Appendices

Inside Cover: Enclosure CD-ROM

Appendices:

A. Technical Data for Existing Robots
B. IDA Calculations
C. Fatigue Life Estimates
D. FTS Screw Connection
E. FEM Verification of FTS
F. Workspace Plots for DC Motors
G. Structural Optimization
H. Loading Spectrum
IJ. Alternative Foot Model
K. Joint Loads from FDA
L. Project Participants
M. Extreme Static Joint Torques
N. Data Applied in FDA
O. Design Overview

Program Source Codes:

PQ. Inverse Dynamics Program
R. Motor & Gear Selection Program
S. Forward Dynamics Program
TU. FTS Calibration Program
VW. Complex Optimization Programs

Technical Documentation:

XYZ. Technical Drawings.

Enclosures:

ÆØÅ. Enclosure A. Battery Specifications
Enclosure B. Commercial Force/Torque Sensors
Enclosure C. Maxon Motor Specifications

Appendix A - Technical Data for Existing Robots

Technical data collected from (Asimo, Online), (Wabian, Online), (Johnnie, Online), (Wollherr, 2005), (Pfeiffer et al., 2002).

	ASIMO	**WABIAN-2**	**Johnnie**
Building year, initial-latest	1986-2005	1966-2006	1998-2003
Weight	54kg	64,5kg	49kg
Height	1300mm	1530mm	1800mm
Width (shoulder)	450mm	512mm	?
Depth	370mm	455mm	?
Walking speed	2,7km/h	1,3km/h <?	2,2km/h
Step length	?	350mm	550mm
Battery	Ni-MH (38,4V/10AH)	Ni-H	-
Battery weight	7,7kg	4,4kg	-
Total DOFs	34	41	17
Leg: hip/knee/ankle	3/1/2	3/1/3	3/1/2
Torso: waist/trunk	1/0	2/2	1/0
Arm: shoulder/elbow/wrist/hand	3/2/2/2	3/1/3/3	2/0/0/0
Head	3	3	0
Servo motor	?	DC servomotor	Maxon DC motors
Reduction	Harmonic drive gear *(HDG)* and timing-belt connection *(TBC)*	HDG and TBC	HDG/ball screw drives and TBC

Appendix B – IDA Calculations

This appendix presents the calculations used in the inverse dynamic analysis. Firstly, the calculations leading to the set up of local coordinate systems. Thereafter, the calculations applied in the kinetic analysis, i.e. the equations of motion.

Local Coordinate Systems

The definition of the local body fixed coordinates, applied in the inverse dynamic analysis (IDA) that indicate the orientation of each body is explained here. These coordinate systems should be orientated in such way that the lock out DOFs that are present in the human test person in the gait experiment, but not in the robot. Furthermore, the coordinate system associated with a body should be aligned with the joint determining the orientation of the next body, as far as possible. Some inaccuracies are introduced in this context, since the coordinate systems have to be defined based on the marker positions from the gait experiment, thus it is difficult aligning them completely with the joints throughout the cycle.

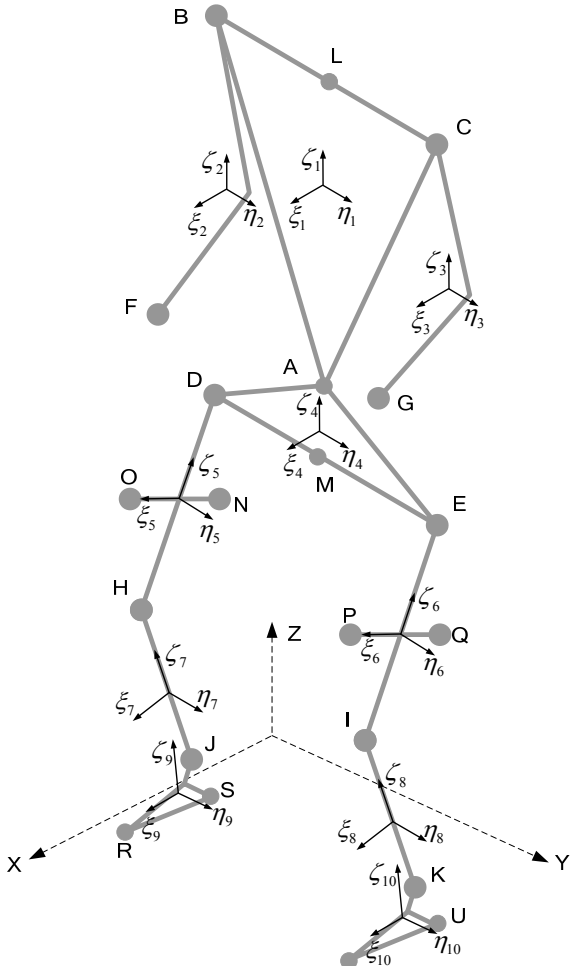


Figure 1 : Local coordinate systems. The coordinates of all points are given from the gait experiment. Please note that the point numbering (the letters) differ from that used in the main report.

Local Coordinate System Orientation

The unit vectors $\underline{\boldsymbol{\xi}}_i$, $\underline{\boldsymbol{\eta}}_i$ and $\underline{\boldsymbol{\zeta}}_i$ defining the axes of the local coordinate systems of body *i* are set up based on the marker positions described in Chapter 5 - Gait Experiment. These of course varies through time as the marker positions vary, therefore new axes have to be calculated for each time step in the input. The unit vectors describing the orientation of coordinate systems are defined here. Based on these vectors one can setup the transformation matrix for body *i* as.

$$\underline{\underline{\boldsymbol{A}}}_i = \left[\underline{\boldsymbol{\xi}}_i \quad \underline{\boldsymbol{\eta}}_i \quad \underline{\boldsymbol{\zeta}}_i\right].$$

If not defined otherwise a vector $\underline{\boldsymbol{r}}_A$ is given by the global coordinates of point *A*, which are available from the gait experiment data.

Body 1: torso

A support point *L* is defined in the middle between points *B* and *C* in Figure 1, with the position

$$\underline{\boldsymbol{r}}_L = \tfrac{1}{2}\left(\underline{\boldsymbol{r}}_B + \underline{\boldsymbol{r}}_C\right)$$

Using *L*, the unit vectors describing orientation of body 1, can be defined as

$$\underline{\boldsymbol{\eta}}_1 = \frac{\underline{\boldsymbol{r}}_C - \underline{\boldsymbol{r}}_B}{\left|\underline{\boldsymbol{r}}_C - \underline{\boldsymbol{r}}_B\right|}, \quad \underline{\boldsymbol{\zeta}}_1 = \frac{\underline{\boldsymbol{r}}_L - \underline{\boldsymbol{r}}_A}{\left|\underline{\boldsymbol{r}}_L - \underline{\boldsymbol{r}}_A\right|} \quad \text{and} \quad \underline{\boldsymbol{\xi}}_1 = \underline{\underline{\tilde{\boldsymbol{\eta}}}}_1 \underline{\boldsymbol{\zeta}}_1.$$

Body 2: right arm

The arms are supposed to swing parallel to the torso, thus $\underline{\boldsymbol{\eta}}_2 = \underline{\boldsymbol{\eta}}_1$. A vector is created from *F* to *B* and projected into the plane defined by applying $\underline{\boldsymbol{\eta}}_2$ as a normal vector.

$$\underline{\boldsymbol{\zeta}}_2 = \frac{(\underline{\boldsymbol{r}}_B - \underline{\boldsymbol{r}}_F) - ((\underline{\boldsymbol{r}}_B - \underline{\boldsymbol{r}}_F)\cdot\underline{\boldsymbol{\eta}}_2)\underline{\boldsymbol{\eta}}_2}{\left|(\underline{\boldsymbol{r}}_B - \underline{\boldsymbol{r}}_F) - ((\underline{\boldsymbol{r}}_B - \underline{\boldsymbol{r}}_F)\cdot\underline{\boldsymbol{\eta}}_2)\underline{\boldsymbol{\eta}}_2\right|} \quad \text{and} \quad \underline{\boldsymbol{\xi}}_2 = \underline{\underline{\tilde{\boldsymbol{\eta}}}}_2 \underline{\boldsymbol{\zeta}}_2.$$

Body 3: left arm

The arms are supposed to swing parallel to the torso, thus $\underline{\boldsymbol{\eta}}_3 = \underline{\boldsymbol{\eta}}_1$. A vector is created from *G* to *C* and projected into the plane defined by applying $\underline{\boldsymbol{\eta}}_3$ as a normal vector.

$$\underline{\boldsymbol{\zeta}}_3 = \frac{(\underline{\boldsymbol{r}}_C - \underline{\boldsymbol{r}}_G) - ((\underline{\boldsymbol{r}}_C - \underline{\boldsymbol{r}}_G)\cdot\underline{\boldsymbol{\eta}}_3)\underline{\boldsymbol{\eta}}_3}{\left|(\underline{\boldsymbol{r}}_C - \underline{\boldsymbol{r}}_G) - ((\underline{\boldsymbol{r}}_C - \underline{\boldsymbol{r}}_G)\cdot\underline{\boldsymbol{\eta}}_3)\underline{\boldsymbol{\eta}}_3\right|} \quad \text{and} \quad \underline{\boldsymbol{\xi}}_3 = \underline{\underline{\tilde{\boldsymbol{\eta}}}}_3 \underline{\boldsymbol{\zeta}}_3.$$

Body 4: pelvis

A support point *M* is defined between *D* and *E*, which yields

$$\underline{\boldsymbol{r}}_M = \tfrac{1}{2}\left(\underline{\boldsymbol{r}}_E + \underline{\boldsymbol{r}}_D\right)$$

$$\underline{\boldsymbol{\eta}}_4 = \frac{\underline{\boldsymbol{r}}_M - \underline{\boldsymbol{r}}_E}{\left|\underline{\boldsymbol{r}}_M - \underline{\boldsymbol{r}}_E\right|} \text{ and } \underline{\boldsymbol{\zeta}}_4 = \frac{\underline{\boldsymbol{r}}_A - \underline{\boldsymbol{r}}_M}{\left|\underline{\boldsymbol{r}}_A - \underline{\boldsymbol{r}}_M\right|} \quad \text{and} \quad \underline{\boldsymbol{\xi}}_4 = \underline{\underline{\tilde{\boldsymbol{\eta}}}}_4 \underline{\boldsymbol{\zeta}}_4$$

Body 5: right thigh

The thighs are defined by

$$\underline{\boldsymbol{\zeta}}_5 = \frac{\underline{\boldsymbol{r}}_H - \underline{\boldsymbol{r}}_D}{\left|\underline{\boldsymbol{r}}_H - \underline{\boldsymbol{r}}_D\right|} \quad \text{and} \quad \underline{\boldsymbol{\eta}}_5 = \frac{\underline{\boldsymbol{r}}_N - \underline{\boldsymbol{r}}_O}{\left|\underline{\boldsymbol{r}}_N - \underline{\boldsymbol{r}}_O\right|} \quad \text{and} \quad \underline{\boldsymbol{\xi}}_5 = \underline{\underline{\tilde{\boldsymbol{\eta}}}}_5 \underline{\boldsymbol{\zeta}}_5$$

Body 6: left thigh

The thighs are defined by

$$\underline{\boldsymbol{\zeta}}_6 = \frac{\underline{\boldsymbol{r}}_I - \underline{\boldsymbol{r}}_E}{\left|\underline{\boldsymbol{r}}_I - \underline{\boldsymbol{r}}_E\right|} \quad \text{and} \quad \underline{\boldsymbol{\eta}}_6 = \frac{\underline{\boldsymbol{r}}_P - \underline{\boldsymbol{r}}_Q}{\left|\underline{\boldsymbol{r}}_P - \underline{\boldsymbol{r}}_Q\right|} \quad \text{and} \quad \underline{\boldsymbol{\xi}}_6 = \underline{\underline{\tilde{\boldsymbol{\eta}}}}_6 \underline{\boldsymbol{\zeta}}_6$$

Body 7: right shin

The shins are both defined by $\underline{\boldsymbol{\eta}}_7 = \underline{\boldsymbol{\eta}}_5$, due to the knee having only one DOF. A vector from the ankle to knee (*J* to *H*) are projected into the plane defined by $\underline{\boldsymbol{\eta}}_7$.

$$\underline{\boldsymbol{\zeta}}_7 = \frac{(\underline{\boldsymbol{r}}_H - \underline{\boldsymbol{r}}_J) - ((\underline{\boldsymbol{r}}_H - \underline{\boldsymbol{r}}_J) \cdot \underline{\boldsymbol{\eta}}_7)\underline{\boldsymbol{\eta}}_7}{\left|(\underline{\boldsymbol{r}}_H - \underline{\boldsymbol{r}}_J) - ((\underline{\boldsymbol{r}}_H - \underline{\boldsymbol{r}}_J) \cdot \underline{\boldsymbol{\eta}}_7)\underline{\boldsymbol{\eta}}_7\right|} \quad \text{and} \quad \underline{\boldsymbol{\xi}}_7 = \underline{\underline{\tilde{\boldsymbol{\eta}}}}_7 \underline{\boldsymbol{\zeta}}_7$$

Body 8: left shin

The shins are both defined by $\underline{\boldsymbol{\eta}}_8 = \underline{\boldsymbol{\eta}}_6$, due to the knee having only one DOF. A vector from the ankle to knee (*K* to *I*) are projected into the plane defined by $\underline{\boldsymbol{\eta}}_8$.

$$\underline{\boldsymbol{\zeta}}_8 = \frac{(\underline{\boldsymbol{r}}_I - \underline{\boldsymbol{r}}_K) - ((\underline{\boldsymbol{r}}_I - \underline{\boldsymbol{r}}_K) \cdot \underline{\boldsymbol{\eta}}_8)\underline{\boldsymbol{\eta}}_8}{\left|(\underline{\boldsymbol{r}}_I - \underline{\boldsymbol{r}}_K) - ((\underline{\boldsymbol{r}}_I - \underline{\boldsymbol{r}}_K) \cdot \underline{\boldsymbol{\eta}}_8)\underline{\boldsymbol{\eta}}_8\right|} \quad \text{and} \quad \underline{\boldsymbol{\xi}}_8 = \underline{\underline{\tilde{\boldsymbol{\eta}}}}_8 \underline{\boldsymbol{\zeta}}_8$$

Body 9: right foot

The feet are kept parallel to the walking direction (global *x*-direction), thus $\underline{\boldsymbol{\eta}}_9 = [0\ 1\ 0]^{\mathrm{T}}$.
A vector is defined from heel to toe (*S* to *R*) and projected into the plane defined by $\underline{\boldsymbol{\eta}}_9$.

$$\underline{\boldsymbol{\xi}}_9 = \frac{(\underline{\boldsymbol{r}}_R - \underline{\boldsymbol{r}}_S) - ((\underline{\boldsymbol{r}}_R - \underline{\boldsymbol{r}}_S) \cdot \underline{\boldsymbol{\eta}}_9)\underline{\boldsymbol{\eta}}_9}{\left|(\underline{\boldsymbol{r}}_R - \underline{\boldsymbol{r}}_S) - ((\underline{\boldsymbol{r}}_R - \underline{\boldsymbol{r}}_S) \cdot \underline{\boldsymbol{\eta}}_9)\underline{\boldsymbol{\eta}}_9\right|} \quad \text{and} \quad \underline{\boldsymbol{\zeta}}_9 = \underline{\underline{\tilde{\boldsymbol{\xi}}}}_9 \underline{\boldsymbol{\eta}}_9$$

Body 10: left foot

The feet are kept parallel to the walking direction (global *x*-direction), thus $\underline{\boldsymbol{\eta}}_{10} = [0\ 1\ 0]^{\mathrm{T}}$.
A vector is defined from heel to toe (*S* to *R*) and projected into the plane defined by $\underline{\boldsymbol{\eta}}_{10}$

$$\underline{\boldsymbol{\xi}}_{10} = \frac{(\underline{\boldsymbol{r}}_T - \underline{\boldsymbol{r}}_U) - ((\underline{\boldsymbol{r}}_T - \underline{\boldsymbol{r}}_U) \cdot \underline{\boldsymbol{\eta}}_{10})\underline{\boldsymbol{\eta}}_{10}}{\left|(\underline{\boldsymbol{r}}_T - \underline{\boldsymbol{r}}_U) - ((\underline{\boldsymbol{r}}_T - \underline{\boldsymbol{r}}_U) \cdot \underline{\boldsymbol{\eta}}_{10})\underline{\boldsymbol{\eta}}_{10}\right|} \quad \text{and} \quad \underline{\boldsymbol{\zeta}}_{10} = \underline{\underline{\tilde{\boldsymbol{\xi}}}}_{10} \underline{\boldsymbol{\eta}}_{10}$$

Equations of Motion

The description applies to the straight walking load case, others are similar, but with minor ad-hoc variations. The total number of bodies is denoted n_b. During the DSP the distribution of reactions between the two feet becomes statically indeterminate. Although there are as many unknowns as equations, the system doesn't decouple, and it is only possible to determine the sum of the reactions.

All reactions are therefore determined using the assumption that the total external reaction can be distributed fairly between the feet in the DSP. Also the position of the applied reaction forces on the feet is based on the assumption that the CoP is coincident to ZMP in SSP.

All vector quantities are calculated positive according to the directions of the global coordinate system.

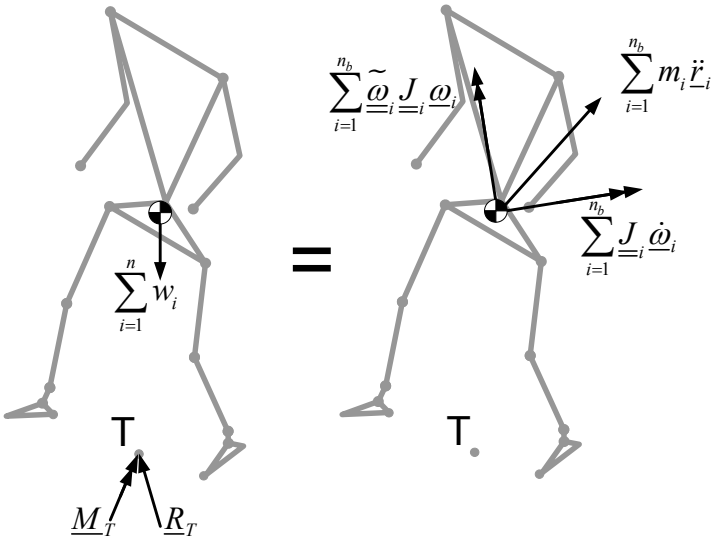


Figure 2: FBD and KD for the entire robot.

The total external reactions caused by the interaction between the robot and the environment are denoted $\underline{\boldsymbol{R}}_T$ and $\underline{\boldsymbol{M}}_T$. These are calculated in the ZMP (point T), why $\underline{\boldsymbol{M}}_T$ only has a vertical component. The position of the ZMP is calculated beforehand, as described in the main report. The calculations are based on summation of the terms for all n_b bodies, where i denotes the body number 1...10. $\underline{\boldsymbol{r}}_{i/T}$ is the vector from T to the i'th body's CoM, $\underline{\boldsymbol{w}}_i$ is the vector of gravity forces on the i'th body. Force and moment equilibrium equations yields

$$\underline{\boldsymbol{R}}_T + \sum_{i=1}^{n_b} \underline{\boldsymbol{w}}_i = \sum_{i=1}^{n_b} m_i \ddot{\underline{\boldsymbol{r}}}_i \;\Leftrightarrow\; \underline{\boldsymbol{R}}_T = -\sum_{i=1}^{n_b} \underline{\boldsymbol{w}}_i + \sum_{i=1}^{n_b} m_i \ddot{\underline{\boldsymbol{r}}}_i$$

$$\underline{\boldsymbol{M}}_T + \sum_{i=1}^{n_b} \tilde{\underline{\boldsymbol{r}}}_{i/T} \underline{\boldsymbol{w}}_i = \sum_{i=1}^{n_b} \tilde{\underline{\boldsymbol{r}}}_{i/T} m_i \ddot{\underline{\boldsymbol{r}}}_i + \sum_{i=1}^{n_b} \underline{\underline{\boldsymbol{J}}}_i \underline{\boldsymbol{\omega}}_i + \sum_{i=1}^{n_b} \underline{\boldsymbol{\omega}} \underline{\underline{\boldsymbol{J}}}_i \underline{\boldsymbol{\omega}}_i \Leftrightarrow$$

$$\underline{\boldsymbol{M}}_T = -\sum_{i=1}^{n_b} \tilde{\underline{\boldsymbol{r}}}_{i/T} \underline{\boldsymbol{w}}_i + \sum_{i=1}^{n_b} \tilde{\underline{\boldsymbol{r}}}_{i/T} m_i \ddot{\underline{\boldsymbol{r}}}_i - \sum_{i=1}^{n_b} \underline{\underline{\boldsymbol{J}}}_i \underline{\boldsymbol{\omega}}_i - \sum_{i=1}^{n_b} \underline{\boldsymbol{\omega}} \underline{\underline{\boldsymbol{J}}}_i \underline{\boldsymbol{\omega}}_i$$

The total external reactions $\underline{\boldsymbol{R}}_T$ and $\underline{\boldsymbol{M}}_T$ is distributed to the feet in the variable points of contact R and S under the right and left foot respectively, as described in the main report. The applied forces and moments are denoted $\underline{\boldsymbol{R}}_R$, $\underline{\boldsymbol{R}}_S$, $\underline{\boldsymbol{M}}_S$ and $\underline{\boldsymbol{M}}_R$. The distribution of these are discussed in the main report.

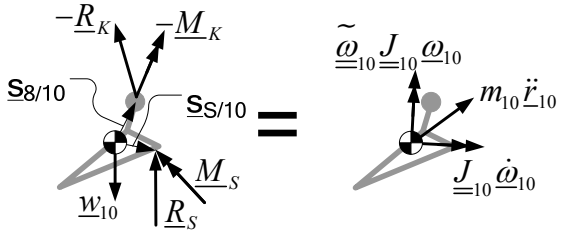


Figure 3: FBD and KD for left foot.

$$\sum \underline{\boldsymbol{F}} : -\underline{\boldsymbol{R}}_K + \underline{\boldsymbol{R}}_S = m_{10}\ddot{\underline{\boldsymbol{r}}}_{10} \Rightarrow \underline{\boldsymbol{R}}_K = \underline{\boldsymbol{R}}_S - m_{10}\ddot{\underline{\boldsymbol{r}}}_{10}$$

$$\sum \underline{\boldsymbol{M}}^G : -\underline{\boldsymbol{M}}_K + \underline{\boldsymbol{M}}_S + \tilde{\underline{\underline{\boldsymbol{s}}}}_{8/10}(-\underline{\boldsymbol{R}}_K) + \tilde{\underline{\underline{\boldsymbol{s}}}}_{S/10}\underline{\boldsymbol{R}}_S = \underline{\underline{\boldsymbol{J}}}_{10}\dot{\underline{\boldsymbol{\omega}}}_{10} + \tilde{\underline{\underline{\boldsymbol{\omega}}}}_{10}\underline{\underline{\boldsymbol{J}}}_{10}\underline{\boldsymbol{\omega}}_{10} \Rightarrow$$
$$\underline{\boldsymbol{M}}_K = \underline{\boldsymbol{M}}_S + \tilde{\underline{\underline{\boldsymbol{s}}}}_{8/10}(-\underline{\boldsymbol{R}}_K) + \tilde{\underline{\underline{\boldsymbol{s}}}}_{S/10}\underline{\boldsymbol{R}}_S - \underline{\underline{\boldsymbol{J}}}_{10}\dot{\underline{\boldsymbol{\omega}}}_{10} - \tilde{\underline{\underline{\boldsymbol{\omega}}}}_{10}\underline{\underline{\boldsymbol{J}}}_{10}\underline{\boldsymbol{\omega}}_{10}$$

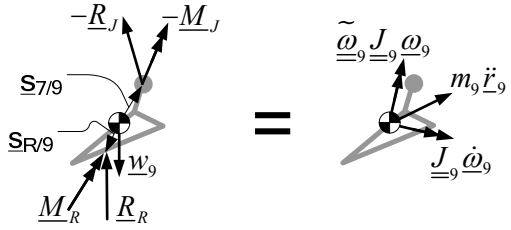


Figure 4: FBD and KD for right foot.

$$\sum \underline{\boldsymbol{F}} : -\underline{\boldsymbol{R}}_J + \underline{\boldsymbol{R}}_R = m_9\ddot{\underline{\boldsymbol{r}}}_9 \Rightarrow \underline{\boldsymbol{R}}_J = \underline{\boldsymbol{R}}_R - m_9\ddot{\underline{\boldsymbol{r}}}_9$$

$$\sum \underline{\boldsymbol{M}}^G : -\underline{\boldsymbol{M}}_J + \underline{\boldsymbol{M}}_R + \tilde{\underline{\underline{\boldsymbol{s}}}}_{7/9}(-\underline{\boldsymbol{R}}_J) + \tilde{\underline{\underline{\boldsymbol{s}}}}_{R/9}\underline{\boldsymbol{R}}_R = \underline{\underline{\boldsymbol{J}}}_9\dot{\underline{\boldsymbol{\omega}}}_9 + \tilde{\underline{\underline{\boldsymbol{\omega}}}}_9\underline{\underline{\boldsymbol{J}}}_9\underline{\boldsymbol{\omega}}_9 \Rightarrow$$
$$\underline{\boldsymbol{M}}_J = \underline{\boldsymbol{M}}_R + \tilde{\underline{\underline{\boldsymbol{s}}}}_{7/9}(-\underline{\boldsymbol{R}}_J) + \tilde{\underline{\underline{\boldsymbol{s}}}}_{R/9}\underline{\boldsymbol{R}}_R - \underline{\underline{\boldsymbol{J}}}_9\dot{\underline{\boldsymbol{\omega}}}_9 - \tilde{\underline{\underline{\boldsymbol{\omega}}}}_9\underline{\underline{\boldsymbol{J}}}_9\underline{\boldsymbol{\omega}}_9$$

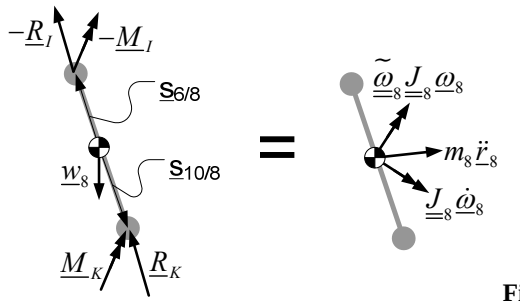


Figure 5: FBD and KD for left shin.

$$\sum \underline{\boldsymbol{F}} : -\underline{\boldsymbol{R}}_I + \underline{\boldsymbol{R}}_K = m_8\ddot{\underline{\boldsymbol{r}}}_8 \Rightarrow \underline{\boldsymbol{R}}_I = \underline{\boldsymbol{R}}_K - m_8\ddot{\underline{\boldsymbol{r}}}_8$$

$$\sum \underline{\boldsymbol{M}}^G : -\underline{\boldsymbol{M}}_I + \underline{\boldsymbol{M}}_K + \tilde{\underline{\underline{\boldsymbol{s}}}}_{6/8}(-\underline{\boldsymbol{R}}_I) + \tilde{\underline{\underline{\boldsymbol{s}}}}_{10/8}\underline{\boldsymbol{R}}_K = \underline{\underline{\boldsymbol{J}}}_8\dot{\underline{\boldsymbol{\omega}}}_8 + \tilde{\underline{\underline{\boldsymbol{\omega}}}}_8\underline{\underline{\boldsymbol{J}}}_8\underline{\boldsymbol{\omega}}_8 \Rightarrow$$
$$\underline{\boldsymbol{M}}_I = \underline{\boldsymbol{M}}_K + \tilde{\underline{\underline{\boldsymbol{s}}}}_{6/8}(-\underline{\boldsymbol{R}}_I) + \tilde{\underline{\underline{\boldsymbol{s}}}}_{10/8}\underline{\boldsymbol{R}}_K - \underline{\underline{\boldsymbol{J}}}_8\dot{\underline{\boldsymbol{\omega}}}_8 - \tilde{\underline{\underline{\boldsymbol{\omega}}}}_8\underline{\underline{\boldsymbol{J}}}_8\underline{\boldsymbol{\omega}}_8$$

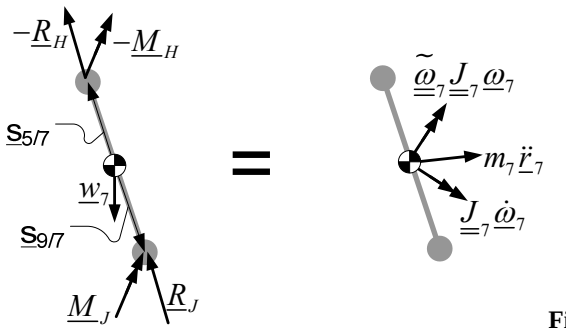


Figure 6: FBD and KD for right shin.

$$\sum \underline{\boldsymbol{F}} : -\underline{\boldsymbol{R}}_H + \underline{\boldsymbol{R}}_J = m_7 \underline{\ddot{\boldsymbol{r}}}_7 \Rightarrow \underline{\boldsymbol{R}}_H = \underline{\boldsymbol{R}}_J - m_7 \underline{\ddot{\boldsymbol{r}}}_7$$

$$\sum \underline{\boldsymbol{M}}^G : -\underline{\boldsymbol{M}}_H + \underline{\boldsymbol{M}}_J + \underline{\tilde{\boldsymbol{s}}}_{5/7}(-\underline{\boldsymbol{R}}_H) + \underline{\tilde{\boldsymbol{s}}}_{9/7}\underline{\boldsymbol{R}}_J = \underline{\underline{\boldsymbol{J}}}_7 \underline{\dot{\boldsymbol{\omega}}}_7 + \underline{\tilde{\boldsymbol{\omega}}}_7 \underline{\underline{\boldsymbol{J}}}_7 \underline{\boldsymbol{\omega}}_7 \Rightarrow$$
$$\underline{\boldsymbol{M}}_H = \underline{\boldsymbol{M}}_J + \underline{\tilde{\boldsymbol{s}}}_{5/7}(-\underline{\boldsymbol{R}}_H) + \underline{\tilde{\boldsymbol{s}}}_{9/7}\underline{\boldsymbol{R}}_J - \underline{\underline{\boldsymbol{J}}}_7 \underline{\dot{\boldsymbol{\omega}}}_7 - \underline{\tilde{\boldsymbol{\omega}}}_7 \underline{\underline{\boldsymbol{J}}}_7 \underline{\boldsymbol{\omega}}_7$$

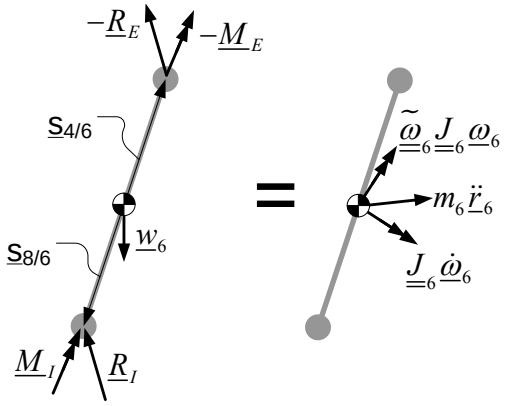


Figure 7: FBD and KD for left thigh.

$$\sum \underline{\boldsymbol{F}} : -\underline{\boldsymbol{R}}_E + \underline{\boldsymbol{R}}_I = m_6 \underline{\ddot{\boldsymbol{r}}}_6 \Rightarrow \underline{\boldsymbol{R}}_E = \underline{\boldsymbol{R}}_I - m_6 \underline{\ddot{\boldsymbol{r}}}_6$$

$$\sum \underline{\boldsymbol{M}}^G : -\underline{\boldsymbol{M}}_E + \underline{\boldsymbol{M}}_I + \underline{\tilde{\boldsymbol{s}}}_{4/6}(-\underline{\boldsymbol{R}}_E) + \underline{\tilde{\boldsymbol{s}}}_{8/6}\underline{\boldsymbol{R}}_I = \underline{\underline{\boldsymbol{J}}}_6 \underline{\dot{\boldsymbol{\omega}}}_6 + \underline{\tilde{\boldsymbol{\omega}}}_6 \underline{\underline{\boldsymbol{J}}}_6 \underline{\boldsymbol{\omega}}_6 \Rightarrow$$
$$\underline{\boldsymbol{M}}_E = \underline{\boldsymbol{M}}_I + \underline{\tilde{\boldsymbol{s}}}_{4/6}(-\underline{\boldsymbol{R}}_E) + \underline{\tilde{\boldsymbol{s}}}_{8/6}\underline{\boldsymbol{R}}_I - \underline{\underline{\boldsymbol{J}}}_6 \underline{\dot{\boldsymbol{\omega}}}_6 - \underline{\tilde{\boldsymbol{\omega}}}_6 \underline{\underline{\boldsymbol{J}}}_6 \underline{\boldsymbol{\omega}}_6$$

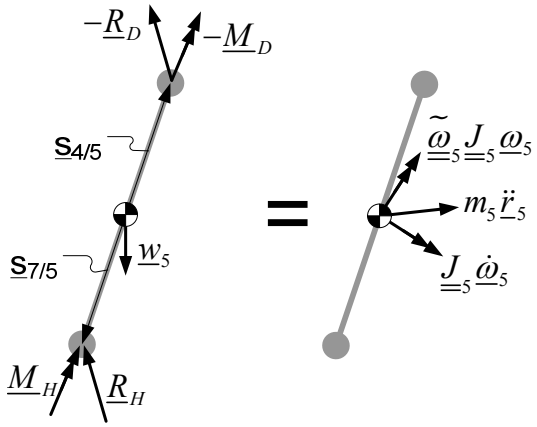


Figure 8: FBD and KD for right thigh.

$$\sum \underline{\boldsymbol{F}} : -\underline{\boldsymbol{R}}_D + \underline{\boldsymbol{R}}_H = m_5 \underline{\ddot{\boldsymbol{r}}}_5 \Rightarrow \underline{\boldsymbol{R}}_D = \underline{\boldsymbol{R}}_H - m_5 \underline{\ddot{\boldsymbol{r}}}_5$$

$$\sum \underline{\boldsymbol{M}}^G : -\underline{\boldsymbol{M}}_D + \underline{\boldsymbol{M}}_H + \underline{\tilde{\boldsymbol{s}}}_{4/5}(-\underline{\boldsymbol{R}}_D) + \underline{\tilde{\boldsymbol{s}}}_{7/5}\underline{\boldsymbol{R}}_H = \underline{\underline{\boldsymbol{J}}}_5 \underline{\dot{\boldsymbol{\omega}}}_5 + \underline{\tilde{\boldsymbol{\omega}}}_5 \underline{\underline{\boldsymbol{J}}}_5 \underline{\boldsymbol{\omega}}_5 \Rightarrow$$
$$\underline{\boldsymbol{M}}_D = \underline{\boldsymbol{M}}_H + \underline{\tilde{\boldsymbol{s}}}_{4/5}(-\underline{\boldsymbol{R}}_D) + \underline{\tilde{\boldsymbol{s}}}_{7/5}\underline{\boldsymbol{R}}_H - \underline{\underline{\boldsymbol{J}}}_5 \underline{\dot{\boldsymbol{\omega}}}_5 - \underline{\tilde{\boldsymbol{\omega}}}_5 \underline{\underline{\boldsymbol{J}}}_5 \underline{\boldsymbol{\omega}}_5$$

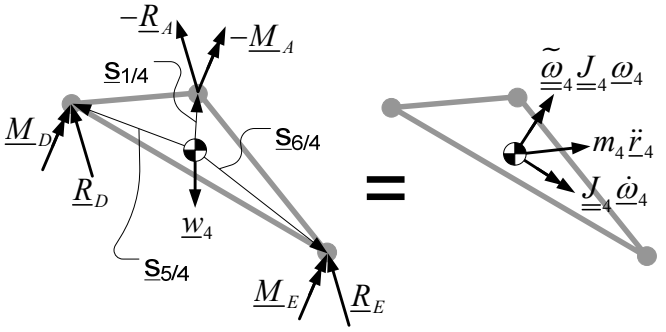


Figure 9: FBD and KD for pelvis.

$$\sum \underline{\boldsymbol{F}} : -\underline{\boldsymbol{R}}_A + \underline{\boldsymbol{R}}_D + \underline{\boldsymbol{R}}_E = m_4 \underline{\ddot{\boldsymbol{r}}}_4 \Rightarrow$$
$$\underline{\boldsymbol{R}}_A = \underline{\boldsymbol{R}}_D + \underline{\boldsymbol{R}}_E - m_4 \underline{\ddot{\boldsymbol{r}}}_4$$

$$\sum \underline{\boldsymbol{M}}^G : -\underline{\boldsymbol{M}}_A + \underline{\boldsymbol{M}}_D + \underline{\boldsymbol{M}}_E + \underline{\tilde{\boldsymbol{s}}}_{1/4}(-\underline{\boldsymbol{R}}_A) + \underline{\tilde{\boldsymbol{s}}}_{5/4}\underline{\boldsymbol{R}}_D + \underline{\tilde{\boldsymbol{s}}}_{6/4}\underline{\boldsymbol{R}}_E = \underline{\underline{\boldsymbol{J}}}_4 \underline{\dot{\boldsymbol{\omega}}}_4 + \underline{\tilde{\boldsymbol{\omega}}}_4 \underline{\underline{\boldsymbol{J}}}_4 \underline{\boldsymbol{\omega}}_4 \Rightarrow$$
$$\underline{\boldsymbol{M}}_A = \underline{\boldsymbol{M}}_D + \underline{\boldsymbol{M}}_E + \underline{\tilde{\boldsymbol{s}}}_{1/4}(-\underline{\boldsymbol{R}}_A) + \underline{\tilde{\boldsymbol{s}}}_{5/4}\underline{\boldsymbol{R}}_D + \underline{\tilde{\boldsymbol{s}}}_{6/4}\underline{\boldsymbol{R}}_E - \underline{\underline{\boldsymbol{J}}}_4 \underline{\dot{\boldsymbol{\omega}}}_4 - \underline{\tilde{\boldsymbol{\omega}}}_4 \underline{\underline{\boldsymbol{J}}}_4 \underline{\boldsymbol{\omega}}_4$$

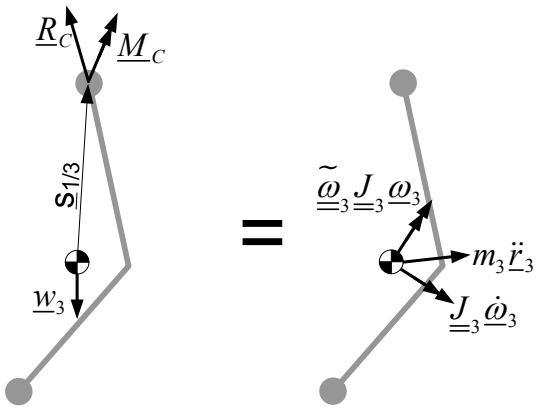


Figure 10: FBD and KD for right arm.

$$\sum \underline{\boldsymbol{F}} : \underline{\boldsymbol{R}}_C = m_3 \underline{\ddot{\boldsymbol{r}}}_3$$

$$\sum \underline{\boldsymbol{M}}^G : \underline{\boldsymbol{M}}_C + \underline{\tilde{\boldsymbol{s}}}_{1/3}\underline{\boldsymbol{R}}_C = \underline{\underline{\boldsymbol{J}}}_3 \underline{\dot{\boldsymbol{\omega}}}_3 + \underline{\tilde{\boldsymbol{\omega}}}_3 \underline{\underline{\boldsymbol{J}}}_3 \underline{\boldsymbol{\omega}}_3 \Rightarrow \underline{\boldsymbol{M}}_C = -\underline{\tilde{\boldsymbol{s}}}_{1/3}\underline{\boldsymbol{R}}_C + \underline{\underline{\boldsymbol{J}}}_3 \underline{\dot{\boldsymbol{\omega}}}_3 - \underline{\tilde{\boldsymbol{\omega}}}_3 \underline{\underline{\boldsymbol{J}}}_3 \underline{\boldsymbol{\omega}}_3$$

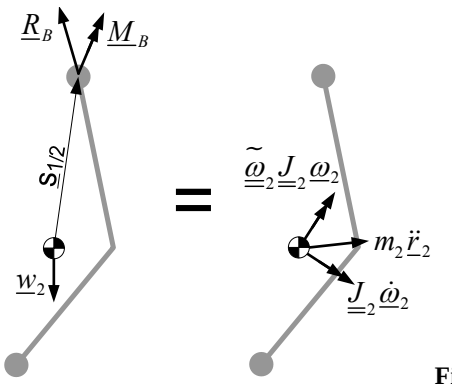


Figure 11: FBD and KD for left arm.

$$\sum \underline{\boldsymbol{F}} : \underline{\boldsymbol{R}}_B = m_2 \ddot{\underline{\boldsymbol{r}}}_2$$

$$\sum \underline{\boldsymbol{M}}^G : \underline{\boldsymbol{M}}_B + \tilde{\underline{\underline{\boldsymbol{s}}}}_{1/2} \underline{\boldsymbol{R}}_B = \underline{\underline{\boldsymbol{J}}}_2 \dot{\underline{\boldsymbol{\omega}}}_2 + \tilde{\underline{\boldsymbol{\omega}}}_2 \underline{\underline{\boldsymbol{J}}}_2 \underline{\boldsymbol{\omega}}_2 \Rightarrow \underline{\boldsymbol{M}}_B = -\tilde{\underline{\underline{\boldsymbol{s}}}}_{1/2} \underline{\boldsymbol{R}}_B + \underline{\underline{\boldsymbol{J}}}_2 \dot{\underline{\boldsymbol{\omega}}}_2 - \tilde{\underline{\boldsymbol{\omega}}}_2 \underline{\underline{\boldsymbol{J}}}_2 \underline{\boldsymbol{\omega}}_2$$

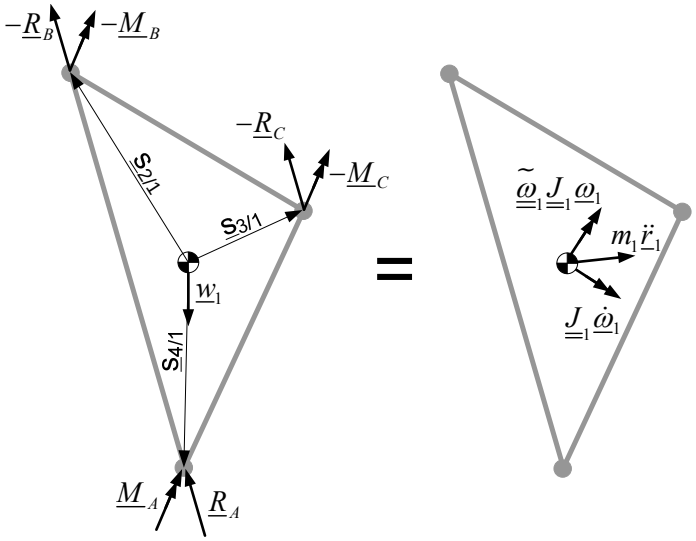


Figure 12: FBD and KD for torso.

$$\sum \underline{\boldsymbol{F}} : \underline{\boldsymbol{R}}_A - \underline{\boldsymbol{R}}_B - \underline{\boldsymbol{R}}_C = m_1 \ddot{\underline{\boldsymbol{r}}}_1 \Rightarrow \underline{\boldsymbol{R}}_A = \underline{\boldsymbol{R}}_B + \underline{\boldsymbol{R}}_C + m_1 \ddot{\underline{\boldsymbol{r}}}_1$$

$$\sum \underline{\boldsymbol{M}}^G : \underline{\boldsymbol{M}}_A - \underline{\boldsymbol{M}}_B - \underline{\boldsymbol{M}}_C + \tilde{\underline{\underline{\boldsymbol{s}}}}_{1/2}(-\underline{\boldsymbol{R}}_B) + \tilde{\underline{\underline{\boldsymbol{s}}}}_{1/3}(-\underline{\boldsymbol{R}}_C) + \tilde{\underline{\underline{\boldsymbol{s}}}}_{4/1} \underline{\boldsymbol{R}}_A = \underline{\underline{\boldsymbol{J}}}_1 \dot{\underline{\boldsymbol{\omega}}}_1 + \tilde{\underline{\boldsymbol{\omega}}}_1 \underline{\underline{\boldsymbol{J}}}_1 \underline{\boldsymbol{\omega}}_1 \Rightarrow$$
$$\underline{\boldsymbol{M}}_A = \underline{\boldsymbol{M}}_B + \underline{\boldsymbol{M}}_C - \tilde{\underline{\underline{\boldsymbol{s}}}}_{1/4} \underline{\boldsymbol{R}}_A - \tilde{\underline{\underline{\boldsymbol{s}}}}_{1/2}(\underline{\boldsymbol{R}}_B) - \tilde{\underline{\underline{\boldsymbol{s}}}}_{1/3}(-\underline{\boldsymbol{R}}_C) + \underline{\underline{\boldsymbol{J}}}_1 \dot{\underline{\boldsymbol{\omega}}}_1 + \tilde{\underline{\boldsymbol{\omega}}}_1 \underline{\underline{\boldsymbol{J}}}_1 \underline{\boldsymbol{\omega}}_1$$

To verify that equilibrium is maintained, the reactions $\underline{\boldsymbol{R}}_A$ and $\underline{\boldsymbol{M}}_A$ calculated from both body 4 and body 1 must be the same. A deviation here, is a consequence of a wrong distribution of feet forces.
All equations are solved for each time step, and the reactions are thus known through time.

Appendix C - Fatigue Life Estimates

This appendix presents the calculation of fatigue life estimates for two representative parts of the robot; the ankle cross axle and the ankle bracket, made from steel and aluminum, respectively. The calculations are based on (Norton, 2000, chapter 6).

Ankle Cross Axle

The duration of one working cycle for the robot is 1.2sec, if it follows the defined straight walking load case. Thus during the expected life of 1000h, it will experience total of approximately 3.600.000 cycles.

Since the exact loading spectrum is unknown due to the unknown control strategy, the fatigue life is estimated, based on the assumption that the loading will be fully reversed.

The maximum alternating stress occurring in the cross axle is determined to σ_a = 214MPa, in *chapter 10 Verification of Mechanical Design* in the main report. The mean stress is set to zero, because of the before mentioned assumption.

The corrected endurance limit, i.e. the stress level under which no fatigue will occur, is determined in the following for the specific steel type applied in the cross axles.

Corrected endurance limit

The endurance limit of the material is corrected for the given loading situation, geometry, environment and desired reliability. The corrected endurance limit S_e can be calculated as

$$S_e = C_{load}\, C_{size}\, C_{surf} C_{temp}\, C_{reliab}\, S_{e'}$$

$S_{e'}$ is the uncorrected endurance limit of the material. $S_{e'}$ can be estimated as

$$S_{e'} = 0.5 S_{ut} \quad , \quad S_{ut} < 1400 MPa$$

Where S_{ut} is the ultimate tensile strength of the material.

C_{load}, is the load factor, which is set to one for rotating bending

$C_{size,}$ is the size factor, which for diameters between 8mm and 250mm is calculated as.

$$C_{size} = 0.869 d^{-0.097}$$

C_{size} is 0.984 for the cross axle, since d=19mm at cross section of the highest stress level.

C_{surf} is the surface factor, which depends on the average surface roughness R_a of the part.

$$C_{surf} = A(S_{ut})^b$$

Where A is the R_a value and b is a corresponding coefficient. R_a for the cross axle is 1.6 µmm and C_{surf} therefore becomes 0.9.

C_{temp} is the temperature factor, which for room temperature is set to one.

C_{reliab} is the reliability factor, which is set to C_{reliab}=0.84 for a reliability of 97.7%.

	C_{load}	C_{size}	C_{surf}	C_{temp}	C_{reliab}	$S_{e'}$
S_e	1	0.984	0.9	1	0.84	700MPa·0.5

Table 1: Values used for correcting the endurance limit.

By multiplying the factors with the uncorrected endurance limit of the material, the corrected endurance limit becomes S_e=260MPa. The corrected endurance limit describes the point of the knee in the Wöhler diagram. Since the maximum occurring stress level is σ_a=214MPa, i.e. less than the corrected endurance limit, the part is dimensioned for infinite life.

Ankle Bracket

The ankle bracket is subjected to a load cycle similar to the cross axle. The ankle bracket is manufactured in aluminum, why the fatigue strength S_f of the material is used instead of the endurance limit. The fatigue strength is defined as the stress level that will yield a fatigue life of $5 \cdot 10^8$ cycles. S_f is found using the same approach as for S_e. The resulting values are listed in Table 2.

	C_{load}	C_{size}	C_{surf}	C_{temp}	C_{reliab}	$S_{f'}$
S_f	1	0.9	0.986	1	0.84	310MPa·0.4

Table 2: Values used for correcting the fatigue strength.

The fatigue strength S_f for the ankle bracket becomes 92MPa. The maximum stress level in the ankle bracket is σ_a=101MPa, and it will thus not last for $5 \cdot 10^8$ cycles. It is therefore necessary to determine the fatigue strength at the required number of cycles, i.e. $3 \cdot 10^6$.

$$S_n = aN^b$$

Where S_n is the fatigue strength at the N number of cycles and a, b are constants defined by the boundary conditions. The constant b can be found as

$$b = \frac{1}{z}\log\left(\frac{S_m}{S_f}\right), \qquad z = \log N_1 - \log N_2$$

Where S_m=0.9S_{ut}, is the estimated material strength at 1000 cycles. The value of the constant z corresponds to the *log* difference between the number of cycles at S_m (N_1 =1000) and S_f ($N_2 = 5 \cdot 10^8$). The value of a can be found by

$$\log(a) = \log(S_m) - b\log(N_1)$$

The value of S_n can then be calculated to 140MPa, which is the stress level that will lead to a fatigue life of the required $3 \cdot 10^6$. Since the maximum occurring stress level in the ankle bracket is below this value, it will not fail due to fatigue within the 1000h of operation.

Appendix D - FTS screw connection

This appendix presents a thorough verification of the FTS screw connection. The calculations in this appendix are based on (Norton, 2000, chapter 14) and (Mouritsen, 2006).

A 2D free body diagram (FBD) of the FTS is shown in Figure 13. The core is cut free from the support ring by the area of connection. It is assumed that the only area of connection is at the effective area, A_m, at each beam.

The FBD of the core is set up in the *xz* and *yz* planes, (top and bottom in Figure 13, respectively). It can be shown that screw 1 is only subjected to moment around the y-axis and the vertical reaction force. It should be noted that the traction forces, in the horizontal plane and the moment around the *z*-axis, are not taken into account yet. These will create a shear force in the connection, which will be examined later. Only the force and moments which can potentially separate the connection is currently investigated.

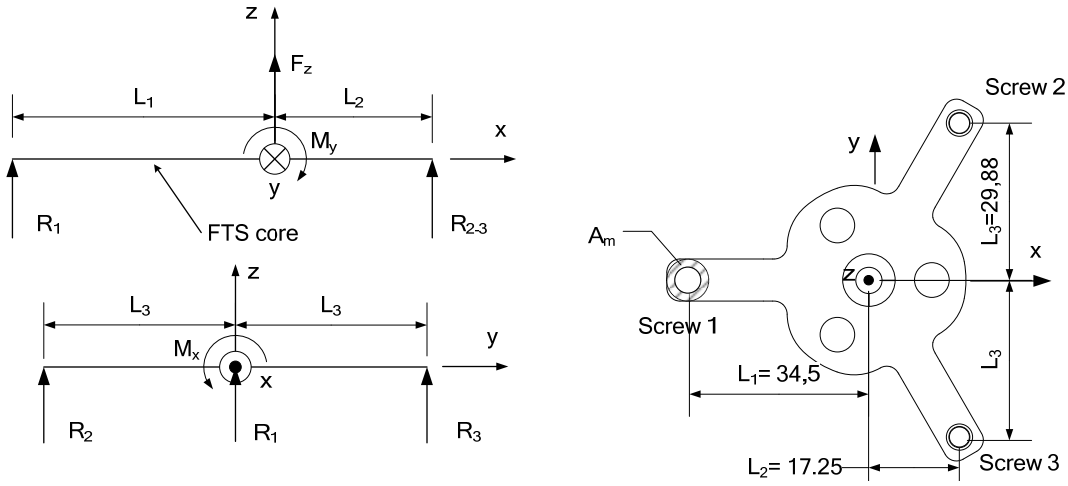


Figure 13: Left: Free body diagrams of the FTS core in the *xz*-plane (top) and *yz*-plane (bottom). Right: FTS core with dimensions.

Overload values:

$$M_y = \pm 250 Nm,\ M_x = \pm 250 Nm$$

Repeated peak loads:

$$F_z = -840 N,\ M_x = 54 Nm,\ M_y = -147 Nm$$

Lengths:

$$L_1 = 34.5 \cdot 10^{-3} m,\ L_2 = 17.25 \cdot 10^{-3} m,\ L_3 = 29.9 \cdot 10^{-3} m$$

Summation of forces in the vertical direction and moment equilibrium around the y-axis yields.

$$\sum F_z = F_z + R_1 + R_{2-3} = 0 \Rightarrow R_1 = -F_z - R_{2-3}$$

$$\sum M_y = M_y + R_1 L_1 - R_{2-3} L_2 = 0 \Rightarrow R_{2-3} = \frac{R_1 L_1 + M_y}{L_2}$$

Reaction force in screw 1

$$R_1 = -F_z - \frac{R_1 L_1 + M_y}{L_2} = \frac{-M_y - F_z L_2}{L_2 + L_1}$$

$$R_{1,\max} = \frac{\pm 250 Nm}{(34.5 + 17.25) \cdot 10^{-3} m} = \pm 4831 N$$

$$R_{1,rep} = \frac{-(-147 Nm) - (-840 N) \cdot 17.25 \cdot 10^{-3} m}{(34.5 + 17.25) \cdot 10^{-3} m} = 3120 N$$

Reactions force in screw 2-3.

$$R_{2-3,\max} = R_{1,\max} = \pm 4831 N$$

$$R_{2-3,rep} = -F_z - R_{1,rep} = -2280 N$$

The reaction forces R_2 and R_3 are determined similarly.

$$\sum F_z = R_2 + R_3 + R_{2-3} = 0 \Rightarrow R_2 = -R_3 - R_{2-3}$$

$$\sum M_x = M_x - R_2 L_3 + R_3 L_3 = 0 \Rightarrow R_3 = \frac{R_2 L_3 - M_x}{L_3}$$

$$R_2 = \frac{R_2 L_3 - M_x}{L_3} - R_{2-3} = \frac{M_x - R_{2-3} L_3}{2 L_3}$$

$$R_{2,\max} = -4180 N, \; R_{2,rep} = 2043 N, \; R_{3,rep} = 237 N$$

Since the reaction force for repeated peak loads in screw 1 is larger than for screw 2 and 3. The further calculations is based on screw 1.

Preloading of screws:

The screws used in this connection are class 12.9 socket head screws. The connection is not to be uncoupled, and a preload of 90 % of the proof strength is therefore used.

$$A_t = 8.78 mm^2, \quad S_p = 830 MPa$$

Where A_t is the area of the cross section of the threaded part of the screw and S_p is the minimum proof strength (Norton, 2000). The preloading force F_i, can then be calculated as

$$F_i = 0.9 \cdot A_t \cdot S_P = 0.9 \cdot 8.78 mm^2 \cdot 970 MPa = 6792 N$$

Criteria set up for a single overloading

$$F_i > R_{\max} \Rightarrow 6792N > 4831N$$

Amount of preload

The torque needed to obtain the preloading force can be calculated for a lubricated thread as

$$T_i = 0.21 \cdot F_i \cdot d = 0.21 \cdot 7665N \cdot 4 \cdot 10^{-3} m = 6.4Nm$$

Where the constant 0.21 is the friction coefficient for a lubricated thread. The surface pressure applied, on the support ring is calculated, first by the head of the screw only, and secondly by using a washer under the screw head.

$$P_{head} = \frac{F_i}{A_{head}} = \frac{6559N}{(7^2 - 4.3^2)mm^2 \frac{\pi}{4}} = 274MPa$$

$$P_{washer} = \frac{F_i}{A_{washers}} = \frac{6559N}{(9^2 - 4.3^2)mm^2 \frac{\pi}{4}} = 134MPa$$

Since the support ring is manufactured in aluminum, a washer is needed in order to lower the surface stress, and avoid yielding.

Support ring and FTS core material properties.

$$E_1 = 71GPa, \quad E_2 = 207GPa$$

Length of compressed material and size of contact area

The length and contact area is illustrated in the main report in *chapter 15 Force Torque Sensor*. It should be noted that the length L_{s2} is 0,5mm longer than according to the figure, because of the thickness of the washer. Therefore the effective area on the support ring is larger as well.

$$L_{s1} = 5.5 \cdot 10^{-3} m, L_{s2} = 7.5 \cdot 10^{-3} m, \; D_{m1} = 9mm, D_{m2} = 8mm, d_m = 4.2mm$$

$$A_{m2} = D_{eff}^2 \cdot \tfrac{\pi}{4} = \left(D_{m1}^2 - d_m^2\right) \cdot \tfrac{\pi}{4} = \left(9^2 - 4.2^2\right) \cdot 10^{-6} m^2 \cdot \tfrac{\pi}{4} = 5 \cdot 10^{-5} m^2$$

$$A_{m2} = D_{eff}^2 \cdot \tfrac{\pi}{4} = \left(D_{m2}^2 - d_m^2\right) \cdot \tfrac{\pi}{4} = \left(8^2 - 4.2^2\right) \cdot 10^{-6} m^2 \cdot \tfrac{\pi}{4} = 3.6 \cdot 10^{-5} m^2$$

Material stiffness

Since the two parts has different material properties, the stiffness is calculated as two springs in series.

$$\frac{1}{k_m} = \frac{L_{s1}}{A_{m1} \cdot E_1} + \frac{L_{s2}}{A_{m2} \cdot E_2} \Downarrow$$

$$\frac{5.5 \cdot 10^{-3} m}{5 \cdot 10^{-5} m^2 \cdot 71GPa} + \frac{7.5 \cdot 10^{-3} m}{3.6 \cdot 10^{-5} m^2 \cdot 207GPa} = 2.56 \cdot 10^{-9} \tfrac{m}{N} \Rightarrow k_m = 3.91 \cdot 10^8 \tfrac{N}{m}$$

Stiffness of screw

L_t is the length of the screw thread and since the thread is the same length as the screw, the stiffness of the screw can be calculated as.

$$\frac{1}{k_b} = \frac{L_t}{A_t \cdot E_b} = \frac{12.5 \cdot 10^{-3} m}{8.78 \cdot 10^{-6} m^2 \cdot 210 GPa} = 6.18 \cdot 10^{-9} \Rightarrow k_b = 1.48 \cdot 10^8$$

The joint stiffness factor for each bolt can be calculated as.

$$C = \frac{k_b}{k_m + k_b} = \frac{1.48 \cdot 10^8 \frac{N}{m}}{3.91 \cdot 10^8 \frac{N}{m} + 1.48 \cdot 10^8 \frac{N}{m}} = 0.27$$

The portion of applied repeated load, felt by the screw and the material can be calculated as.

$$P_{b1} = \left| R_{1,rep} \right| \cdot C = 3120N \cdot 0.27 = 842N$$
$$F_{rep,\max} = F_i + P_{b1} = 6791N + 842N = 7633N$$
$$F_{rep,\min} = F_i - P_{b1} = 6791N - 842N = 5949N$$
$$P_{m1} = (1 - C) \cdot \left| R_{1,rep} \right| = 2277N$$

The nominal amplitude stress of the screw can be calculated as.

$$\sigma_{a,nom} = \frac{P_{b1}}{2 \cdot A_t} = \frac{842N}{2 \cdot 8.78 mm^2} = 48 MPa$$

The fatigue strength can in practice be calculated independently from the screw (Mouritsen, 2006). The endurance limit, with a reliability of 50%, can be calculated as.

$$S_{e,50\%} = \left\{ \frac{150mm}{d} + 45 \right\} \frac{N}{mm^2} = \left\{ \frac{150mm}{4mm} + 45 \right\} = 82.5 MPa$$

For a reliability of 99,999%, C_{reliab} is used (Norton, 2006, p.353)

$$S_{e,99,999\%} = C_{reliab} \cdot S_{e,50\%} = 0.659 \cdot 82.5 MPa = 54.4 MPa$$

The safety factor for fatigue, N_{fat} should be ≥1.

$$N_{fat} = \frac{S_{e,99,999\%}}{\sigma_{a,nom}} = \frac{54,4 MPa}{48 MPa} = 1.13 \geq 1$$

The safety factor of separation is.

$$N_{seperation} = \frac{F_i}{F_{rep\max}(1 - C)} = \frac{6792N}{7633N(1 - 0.27)} = 1.2$$

Slip

The screw connection is tested for slip, generated by the shear force and the moment around the *z*-axis. Figure 14 (left) shows a FBD of the core in the transverse plane. As seen the FTS is statically indeterminate, and a conservative assumption is taken, i.e. that one screw should absorb the two forces in the horizontal plane.

The forces and moment applied are extracted from the IDA results, when M_z peaks.

$$F_x = -210N \ , F_y = 30N, \ M_z = 15Nm$$

$$F_V = \sqrt{F_x^{\ 2} + F_y^{\ 2}} = \sqrt{210^2 + 30^2} = 212N$$

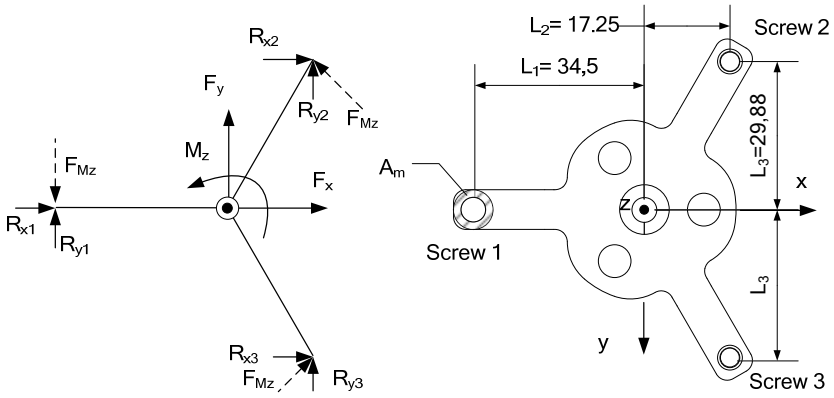


Figure 14: Left: FBD in the horizontal plane of the FTS. Right: FTS seen from above. F_{Mz} is the equivalent force of the moment around the z-axis.

Since the screws are positioned in symmetrically around the vertical axis, the shear force at each screw produced by the moment around the z-axis can be calculated as.

$$F_{Mz} = \frac{M_z \cdot L_1}{\sum_{j=1}^{3} r^2} = \frac{15Nm \cdot 34.5 \cdot 10^{-3} m}{3 \cdot (34.5 \cdot 10^{-3})^2 m^2} = 144N$$

The shear force for one screw can be combined as a vector sum.

$$F_S = \sqrt{F_V^2 + F_{M_z}^2} = 256N$$

The friction between steel and aluminum can be set conservative as $\mu = 0.15$. The friction force can be calculated as.

$$F_f = F_{rep,\min} \cdot \mu = 892N$$

Safety factor against slip

$$N_{slip} = \frac{F_f}{F_S} = 3.5$$

Appendix E – FEM Verification of FTS

A FEM analysis of each load situation of overloading is carried out, to ensure that no permanent deformation occur during overloading the FTS. The results are shown as color plots of the von Mises nodal stresses below. The load and boundary condition are described in the main report.

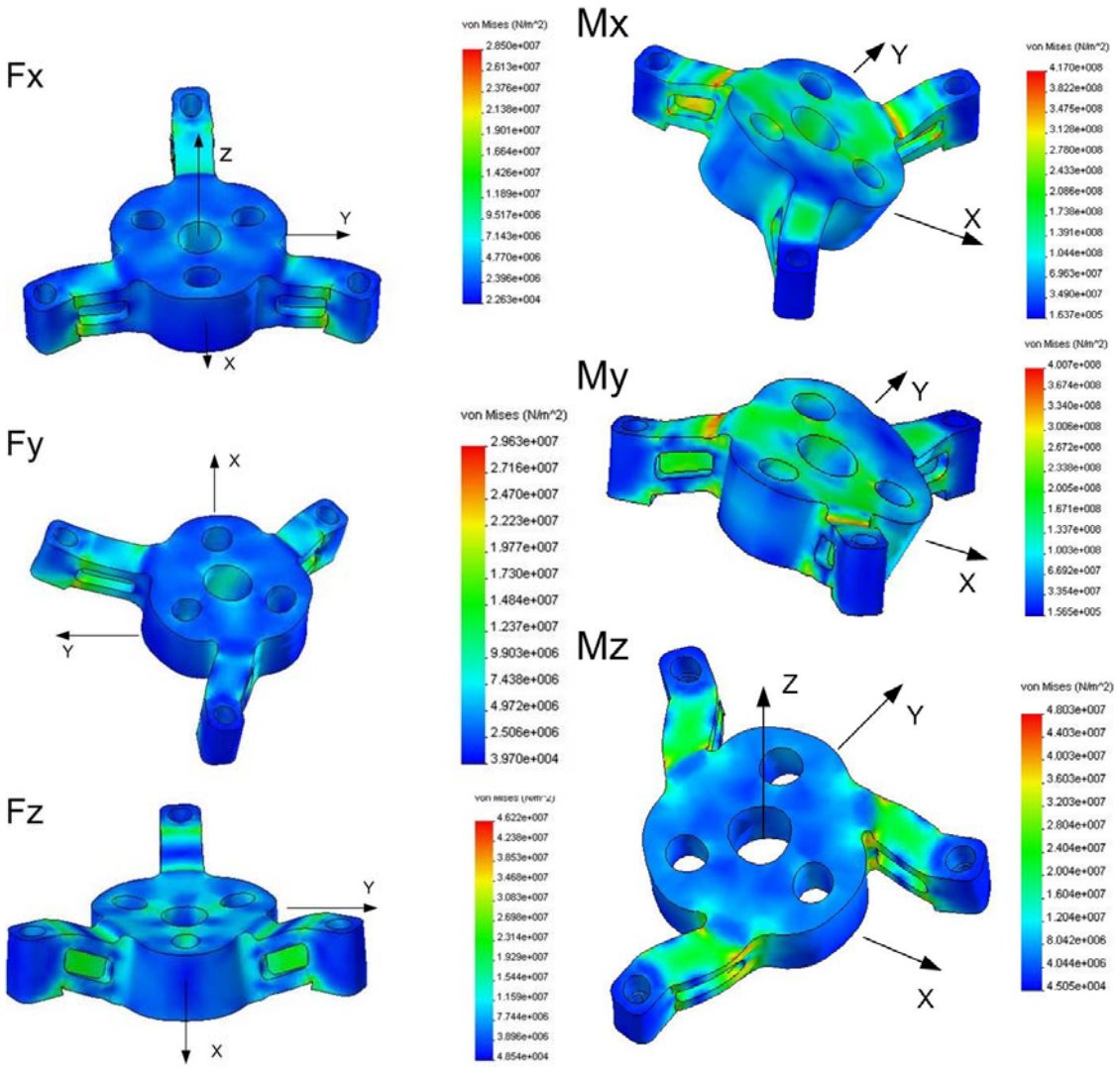


Figure 15: The von Mises stress level in the FTS, when subjected to overloading.

Appendix F – Workspace Plots for DC Motors

The ordered DC motors for the robot. All DC motors have built-on encoders. The type of encoder is; Encoder MR, Type L, 512CPT, 3 Channels, with Line Driver. This encoder has 512 counts per turn and a maximum operation frequency of 160 kHz.

Motor type	Power	Voltage	Weight	Order No.	Number
RE 30	60W	48V	0,238kg	310009	5
RE 35	90W	48V	0,34kg	273755	3
RE 40	150W	48V	0,48kg	148877	15

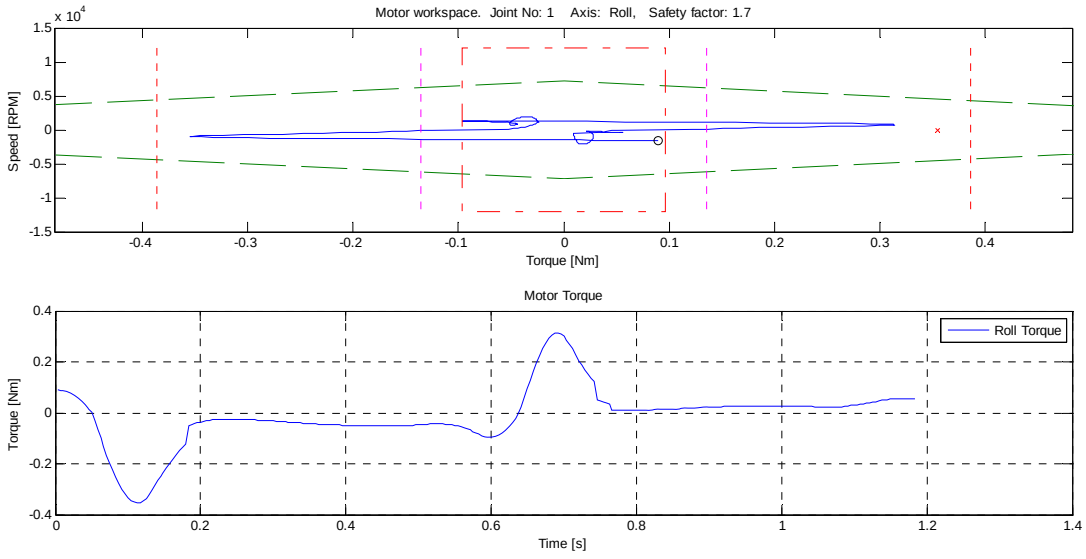
Table 3: The ordered Maxon DC motors.

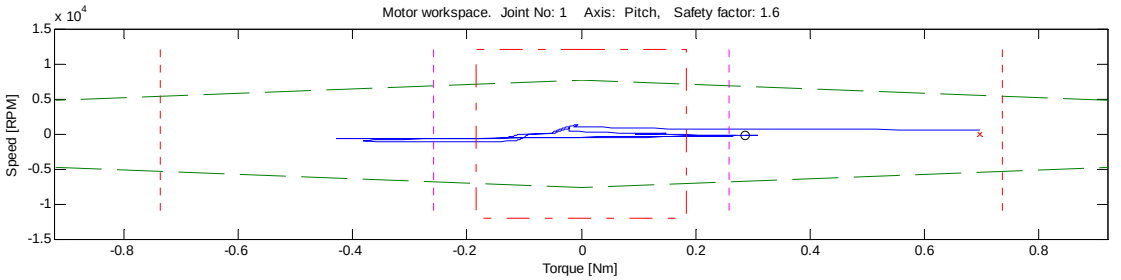
Plots of motor-workspace and torque/time for the selected DC motors are shown in the following. The load data used is based on the inertial properties of the final robot design, scaled up with a safety factor. The title for each plot lists the joint name, number, axis and the safety factor applied.

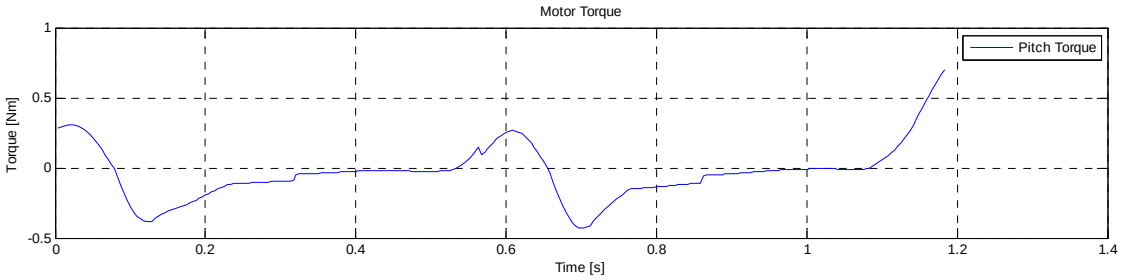
Legend for all plots:

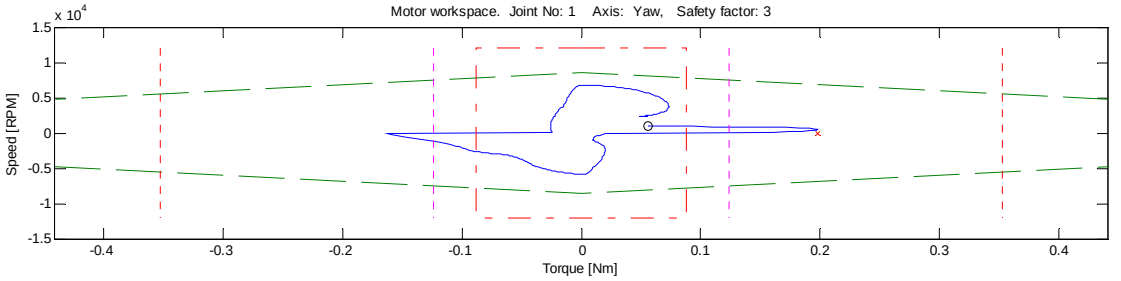
- Motor trajectory
- Max abs Torque
- starting point
- Torque-speed line
- Continuous operation workspace
- 1.4xNominal Torque
- 4xNominal Torque

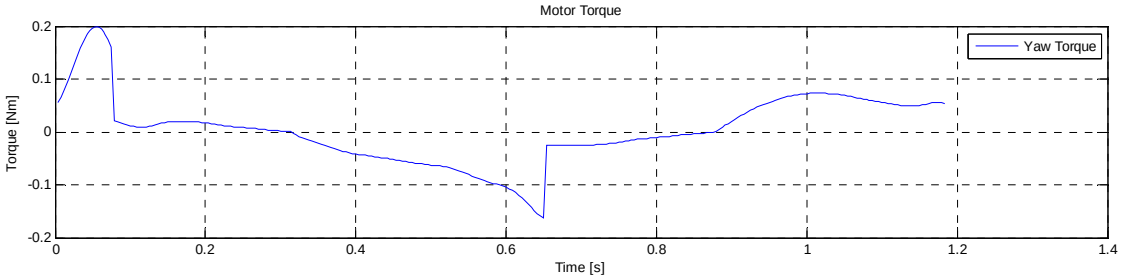
Waist roll-/ pitch-/yaw- DC motor.



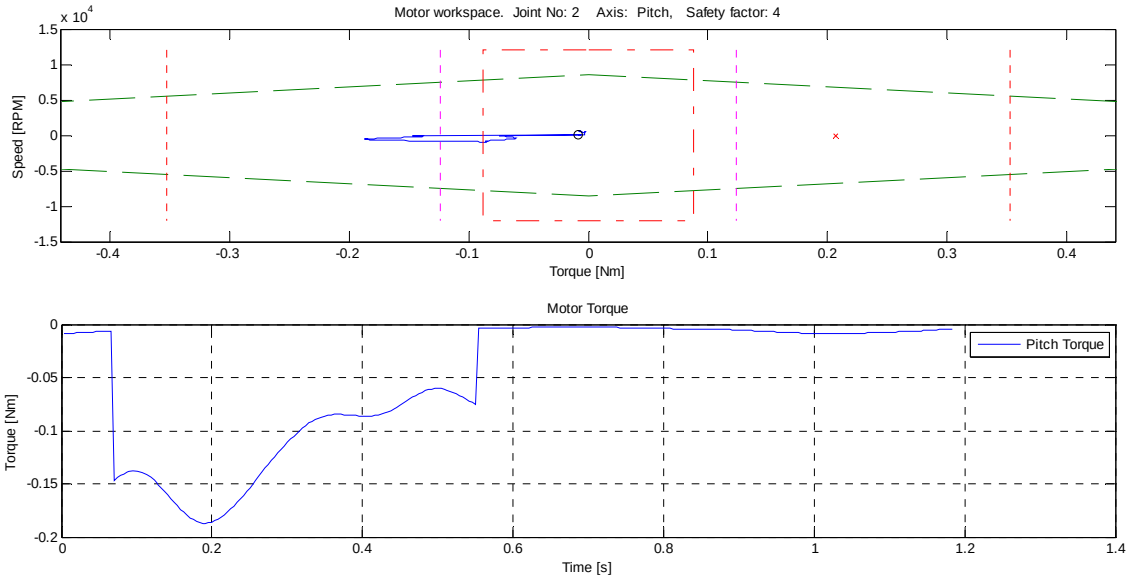

Motor workspace. Joint No: 1 Axis: Pitch, Safety factor: 1.6
x 10^4
Speed [RPM]
Torque [Nm]


Motor Torque
Pitch Torque
Torque [Nm]
Time [s]

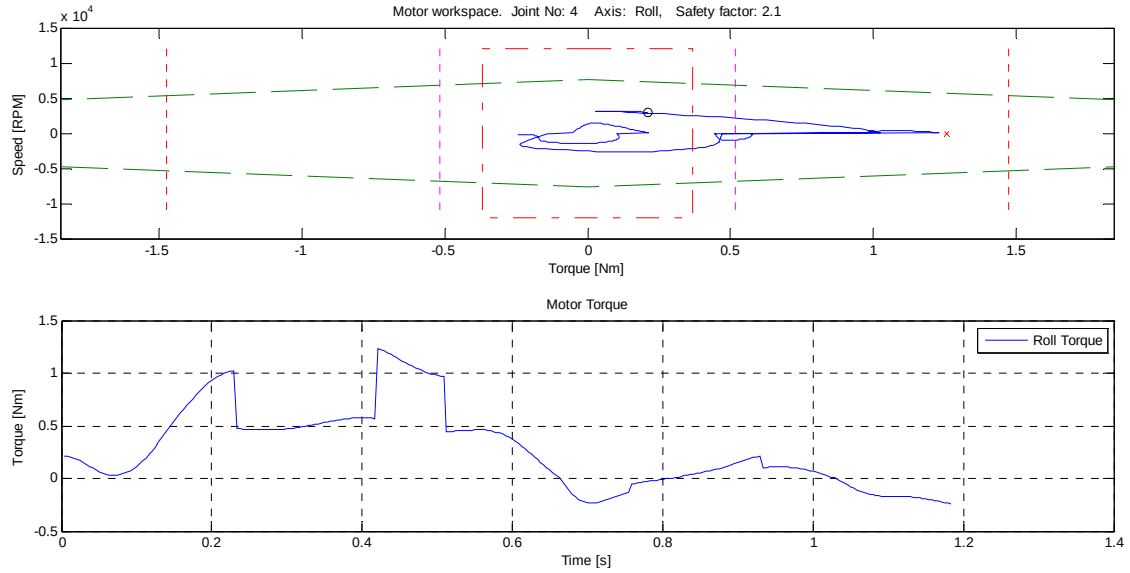

Motor workspace. Joint No: 1 Axis: Yaw, Safety factor: 3
x 10^4
Speed [RPM]
Torque [Nm]

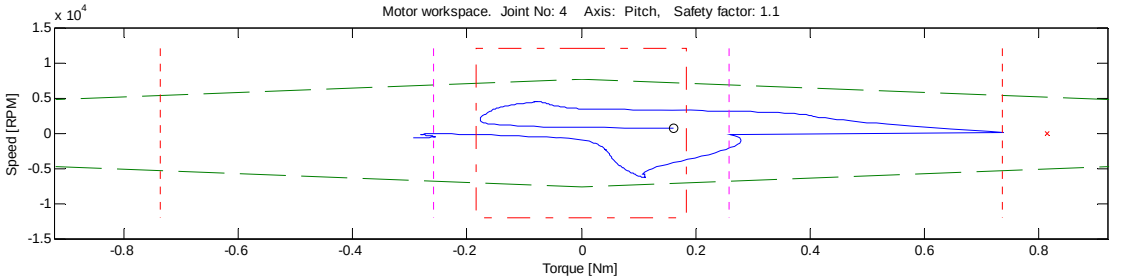

Motor Torque
Yaw Torque
Torque [Nm]
Time [s]

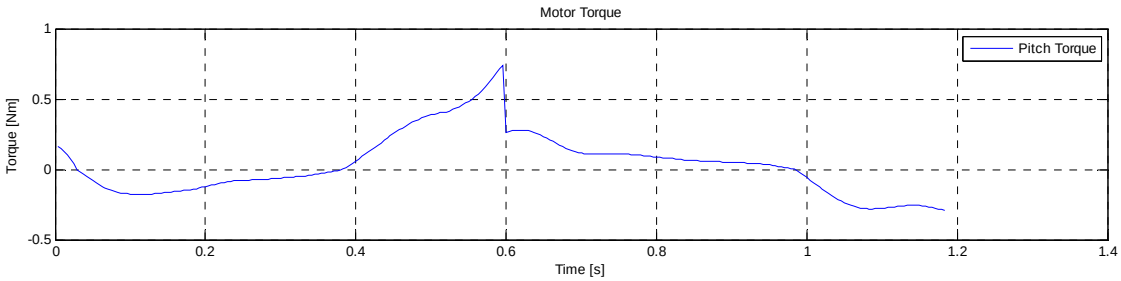
Right- / Left-Shoulder pitch DC motor

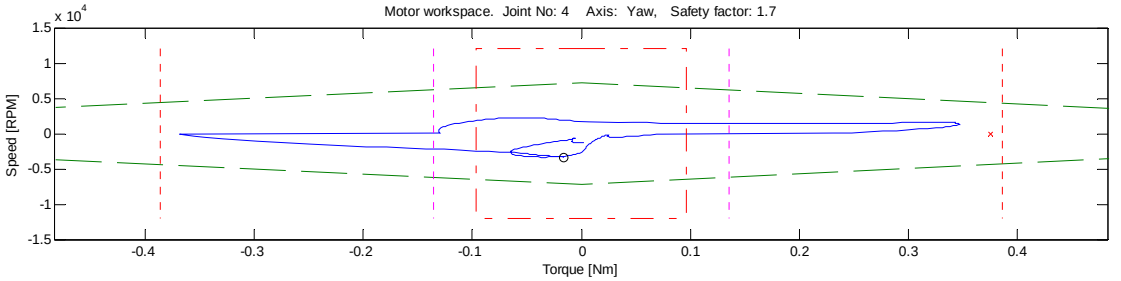


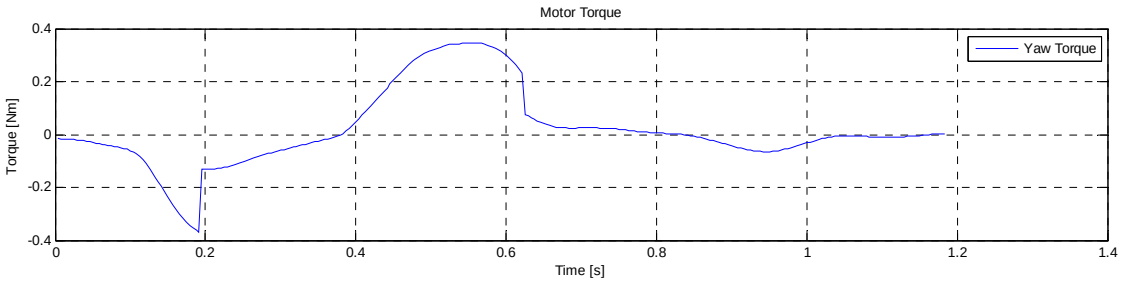
Right-/ Left- Hip pitch- /yaw- DC motor.



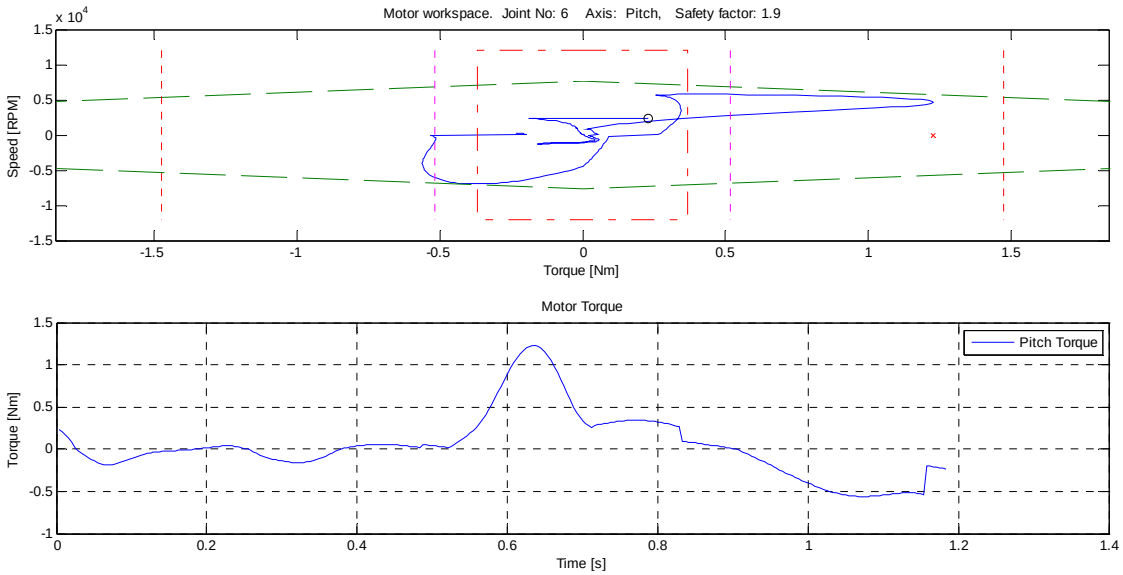

Motor workspace. Joint No: 4 Axis: Pitch, Safety factor: 1.1
x 10^4
Speed [RPM]
Torque [Nm]


Motor Torque
Pitch Torque
Torque [Nm]
Time [s]

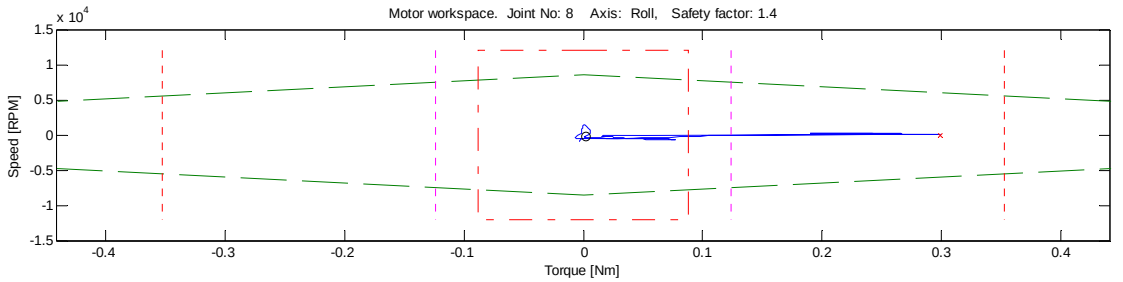

Motor workspace. Joint No: 4 Axis: Yaw, Safety factor: 1.7
x 10^4
Speed [RPM]
Torque [Nm]

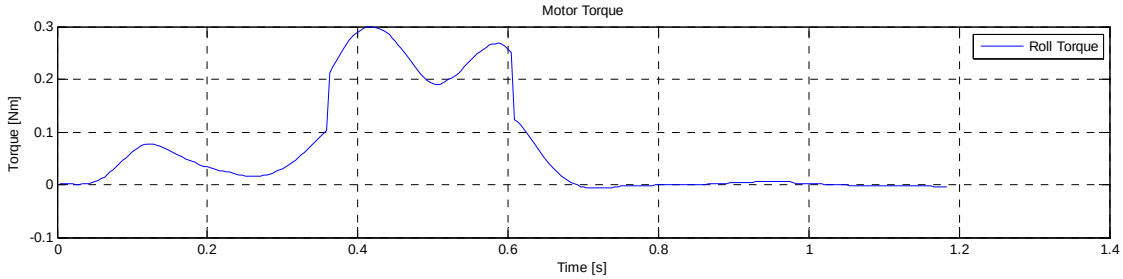

Motor Torque
Yaw Torque
Torque [Nm]
Time [s]

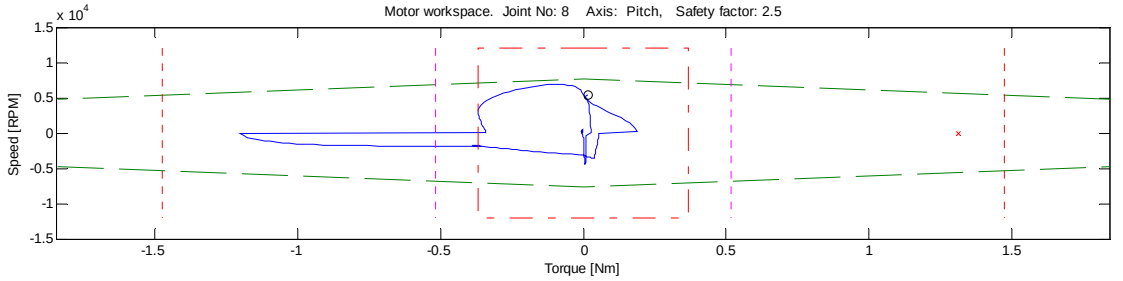
Right / Left Knee pitch- DC motor

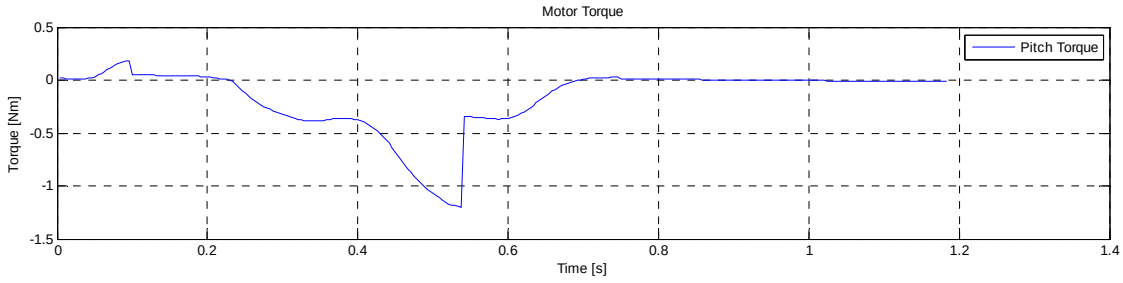


Right / Left Ankle roll-/ pitch- DC motor









Appendix G – Structural Optimization

Supplementary plots of results obtained from FEM analyses, used in the structural optimization.

Analysis of the Beam Model

The height of design variable x_1 is reduced by 1mm in the beam model, suggested as optimum by the complex routine. The variables are x_1= 11.5mm and x_2=2,2mm, see Figure 16.

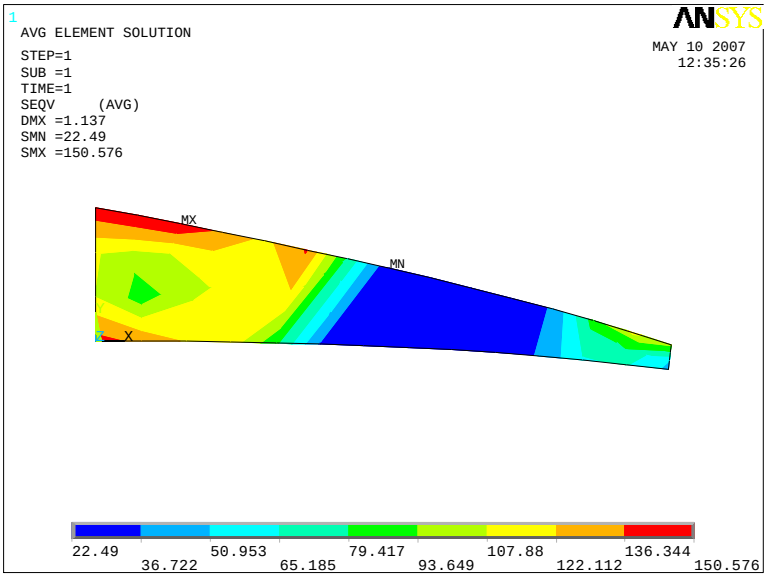
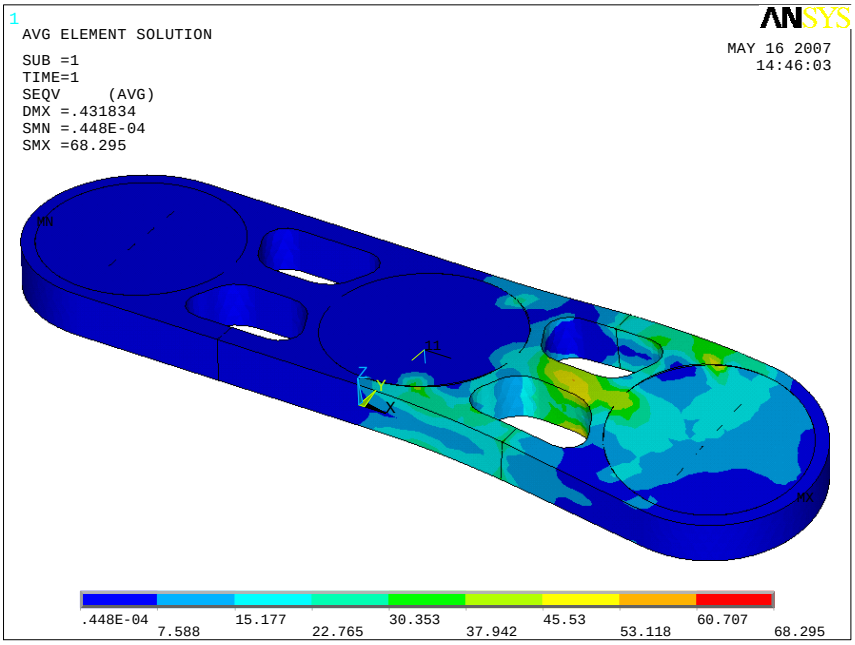


Figure 16: The average von Mises stress plot of each element.

Analysis of Original Pelvis

Figure 17 shows stress and deformation plots of the original pelvis design.



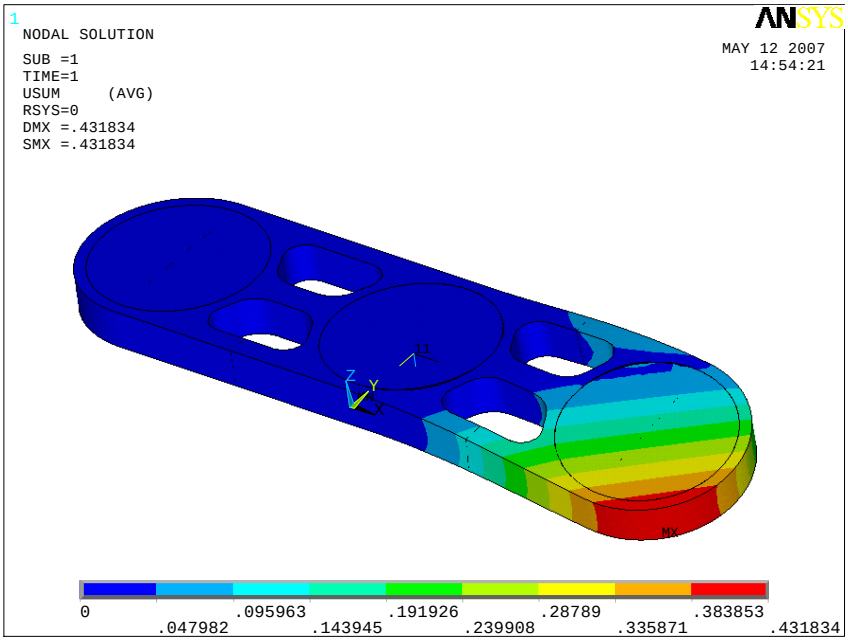


Figure 17: Top: stress plot of the original pelvis design.
Bottom: displacement of the original pelvis design, during peak hip loads.

Analysis of a Quarter of the Original Pelvis

The quarter pelvis-model is only subjected to the half load, of the original. As seen, there is a good agreement between the whole pelvis model and quarter model. Figure 18 shows a deformation plot of the quarter model, which can be compared to Figure 17.

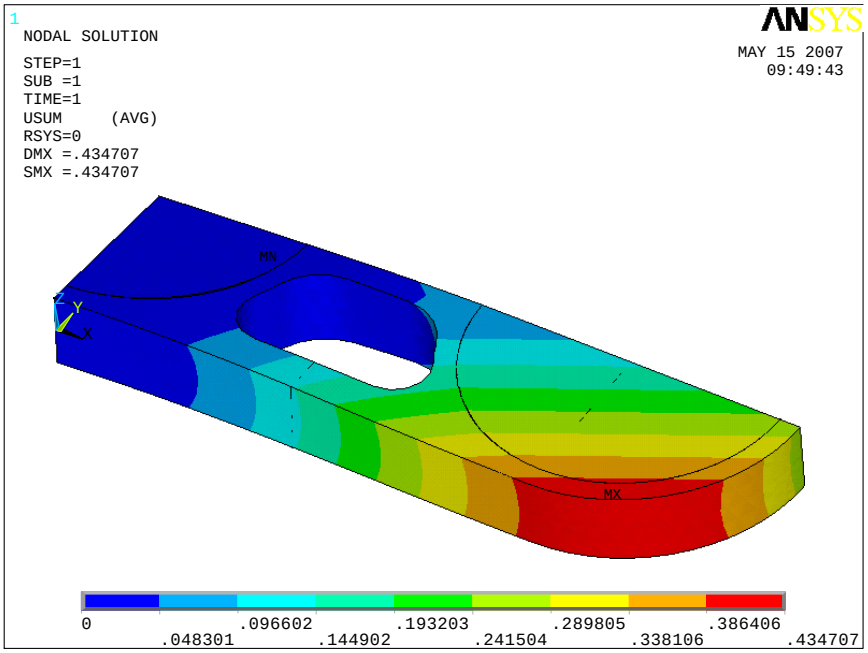


Figure 18: Displacement of the pelvis-model. A symmetry BC is applied to the center line of the model.

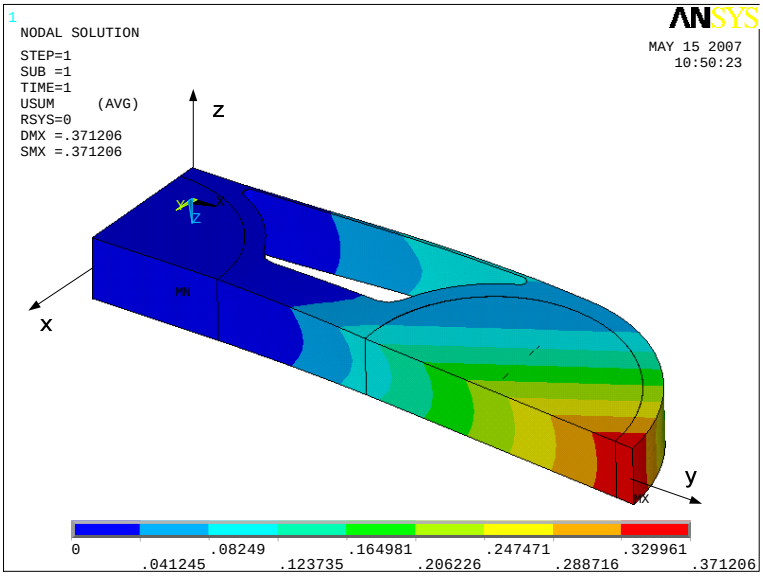


Figure 19: Plot of the deflection of the upper left pelvis-model, with local coordinate system.

To ensure that the optimized design of the lower left side of the pelvis also fits the upper left side and that the stress or displacement limits are not violated, this part is also analyzed. The resulting von Mises stress and displacement plots are shown below.

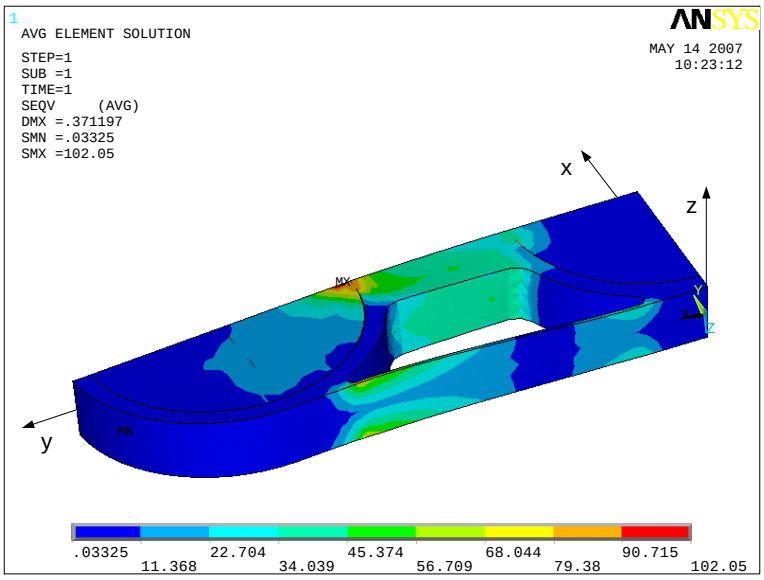


Figure 20: von Mises stress plot of the upper left pelvis model, incl. local coordinate system.

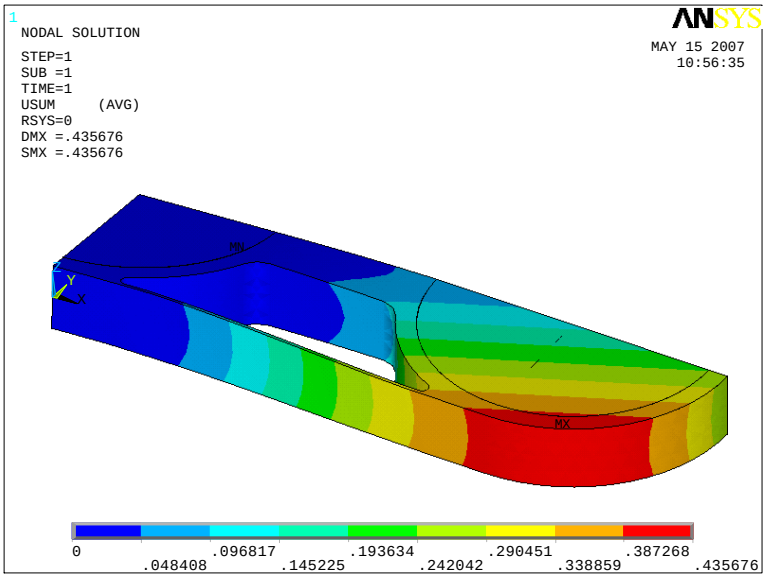


Figure 21: Displacement of the optimized pelvis model.

Figure 22 shows the displacement of the optimized design, without the lower beam.

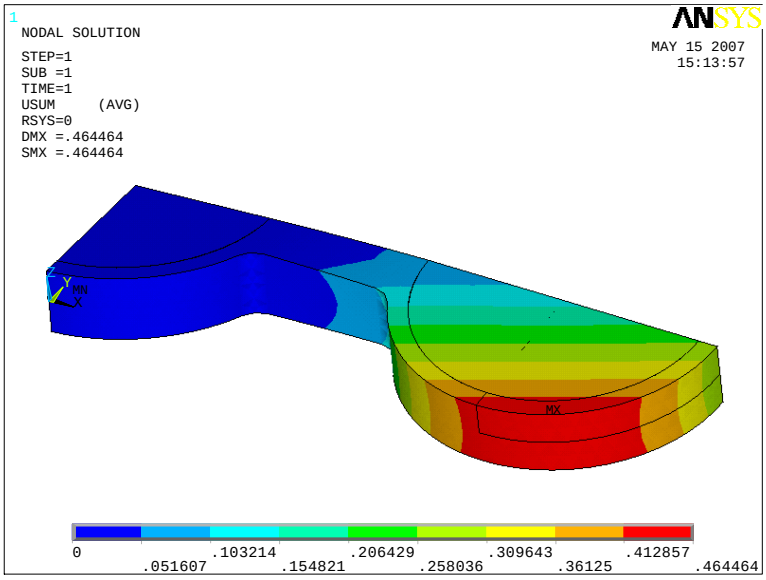


Figure 22: Displacement of the pelvis design without the lower beam.

By increasing the deflection limit 25% the upper beam is reduced further in width. The deflection of the optimum design with this new deflection limit is shown in Figure 23.

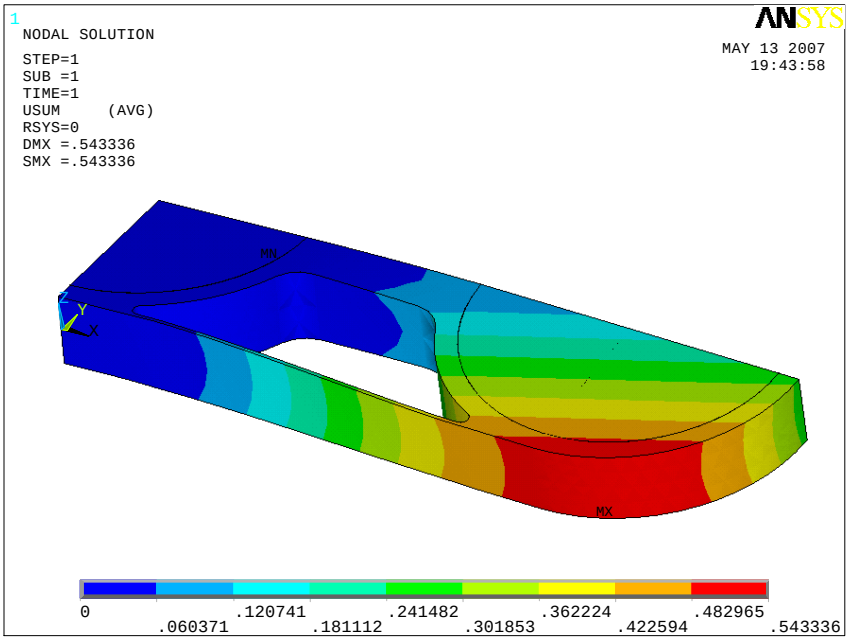


Figure 23: Displacement of the optimized pelvis, if the deflection limit is increased by 25%.

Appendix H - Loading Spectrum

This appendix contains plots of the time-varying loads used in the dimensioning of the robot. These comprise body and joint related kinematics and the reaction forces and moments determined using both the initial human inertial data and the final robot's inertial data. The plots are for the straight walking load case only, since this is the most interesting load case. None of the data is scaled with the factor of 1.8, discussed in the main report.

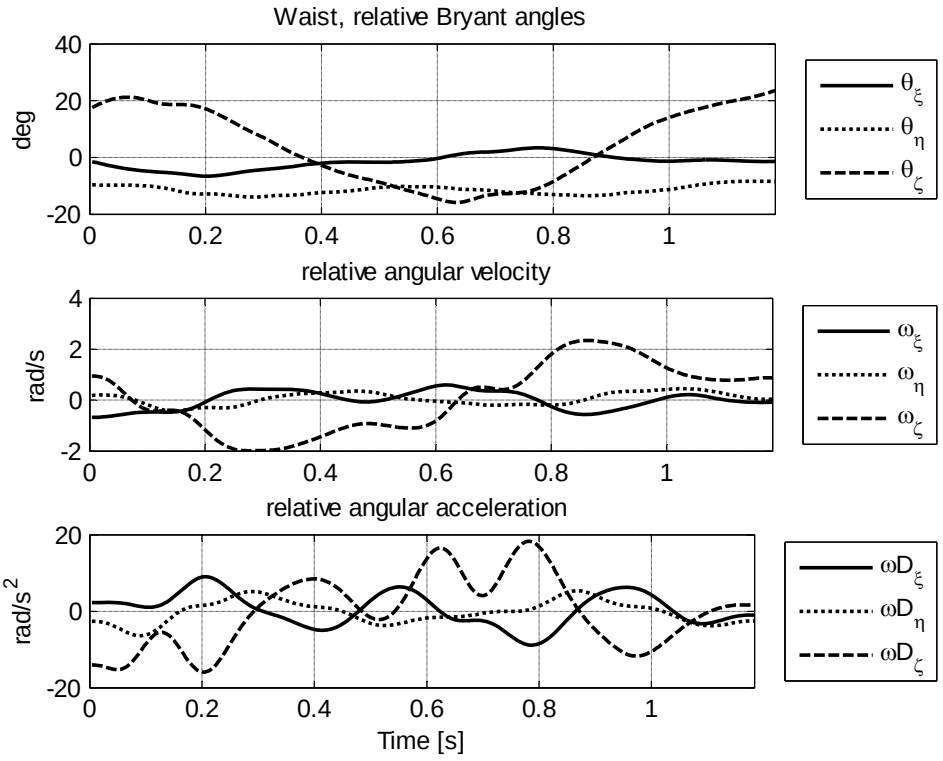
Only the data for the right side of the body is included, since the data for the left hand side is very similar, only shifted in time.

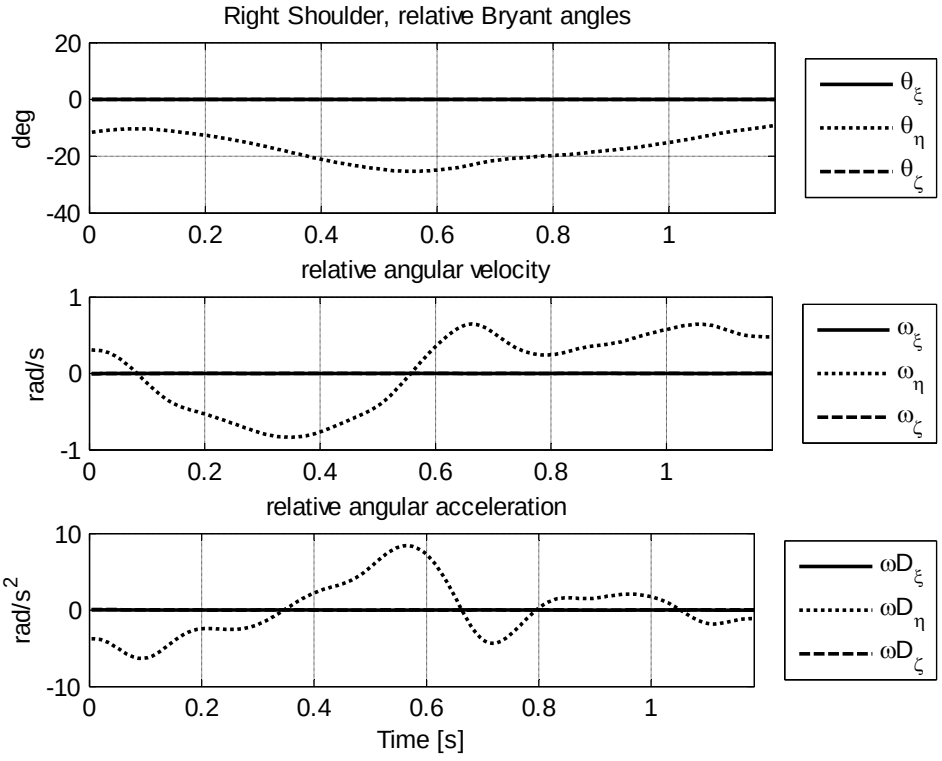
- Joints: waist, right shoulder, right hip, right knee, right ankle.
- Bodies: torso, right arm, pelvis, right thigh, right shin, right foot.

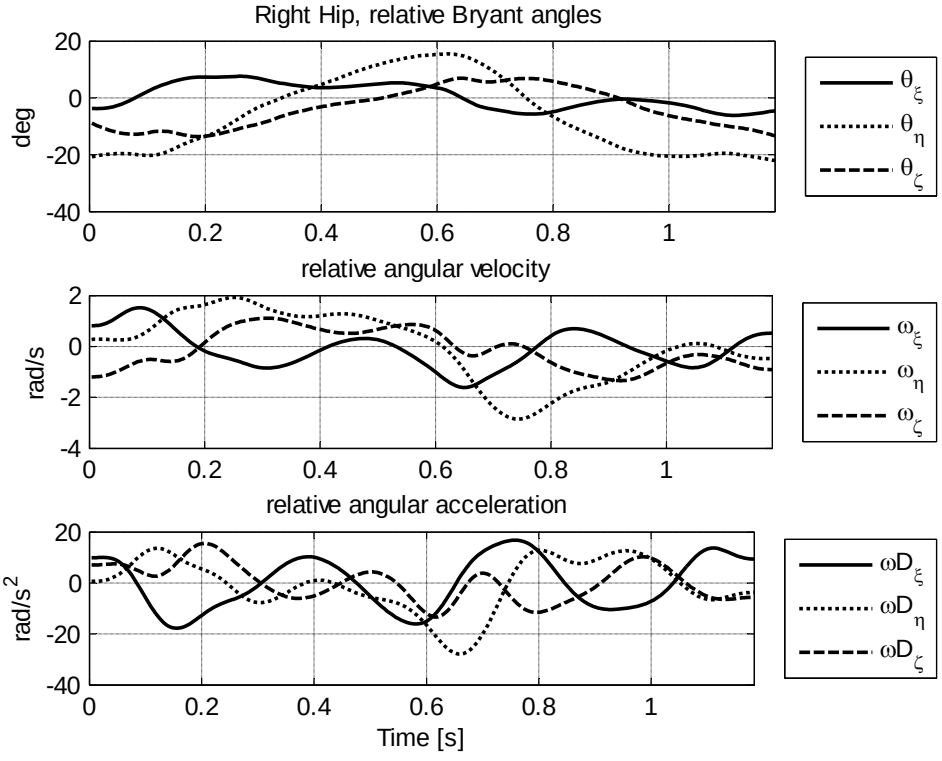
This appendix is arranged as follows:

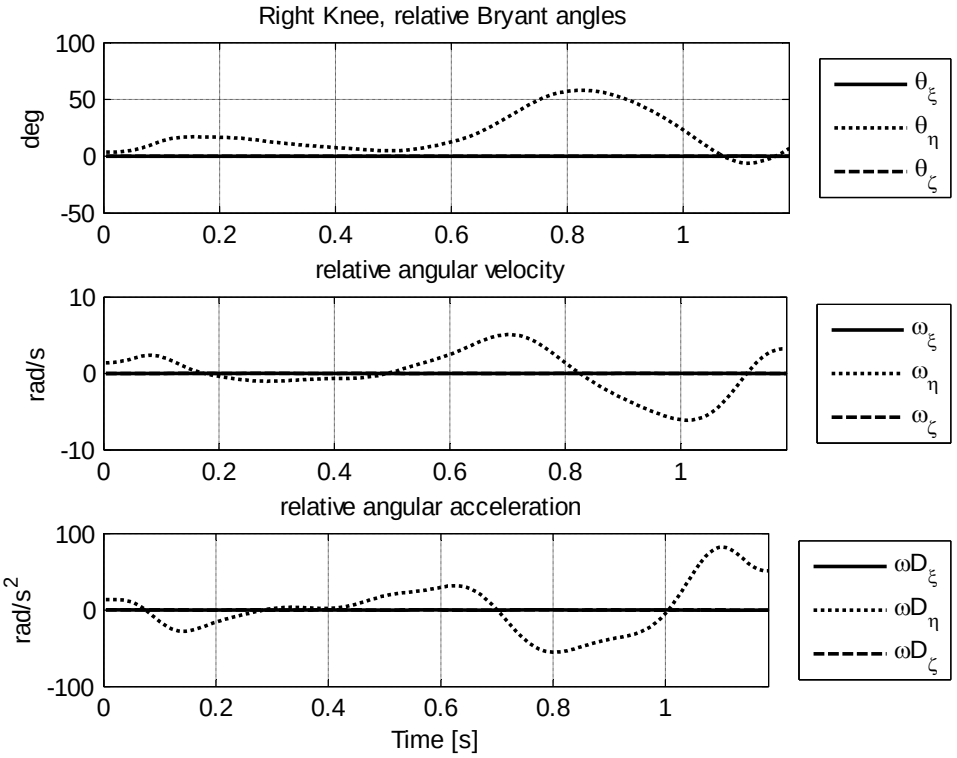
- Plots of joint related kinematics, i.e. relative Bryant angles, relative angular velocity and acceleration.
- Plots of body related kinematics, i.e. position, velocity and acceleration of the body CoM's.
- Plots of reaction forces and moments. Firstly calculated using the initial human inertial data then using the final robot inertial data.

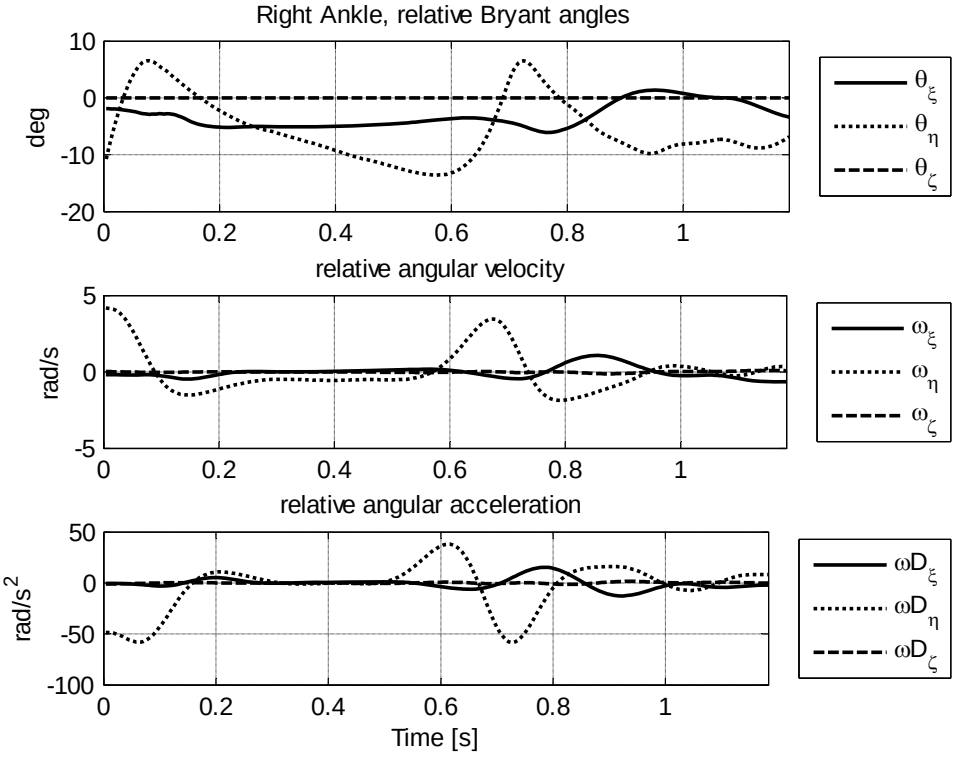
Joint Kinematics



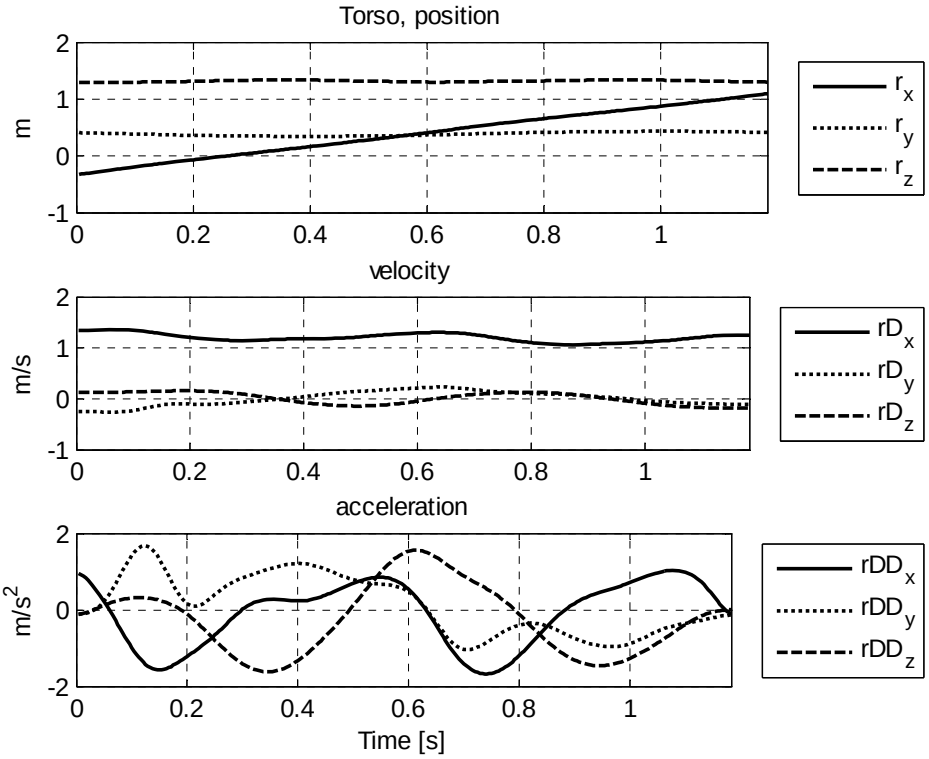

Right Shoulder, relative Bryant angles
deg
relative angular velocity
rad/s
relative angular acceleration
rad/s^2
Time [s]
θξ
θη
θζ
ωξ
ωη
ωζ
ωDξ
ωDη
ωDζ

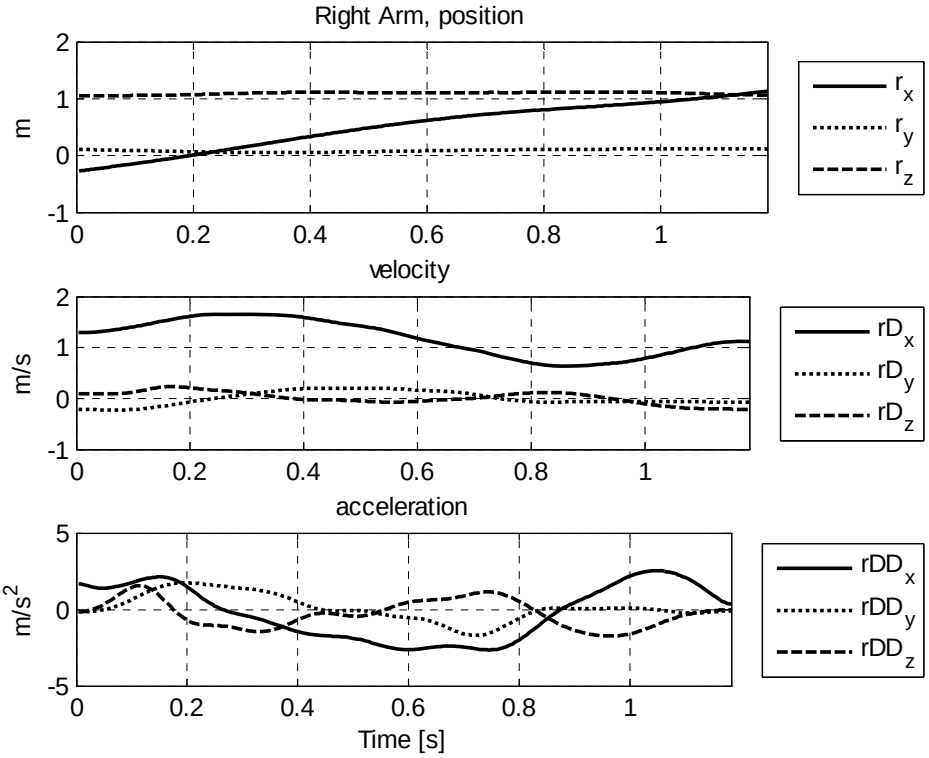

Right Hip, relative Bryant angles
deg
relative angular velocity
rad/s
relative angular acceleration
rad/s^2
Time [s]
θξ
θη
θζ
ωξ
ωη
ωζ
ωDξ
ωDη
ωDζ

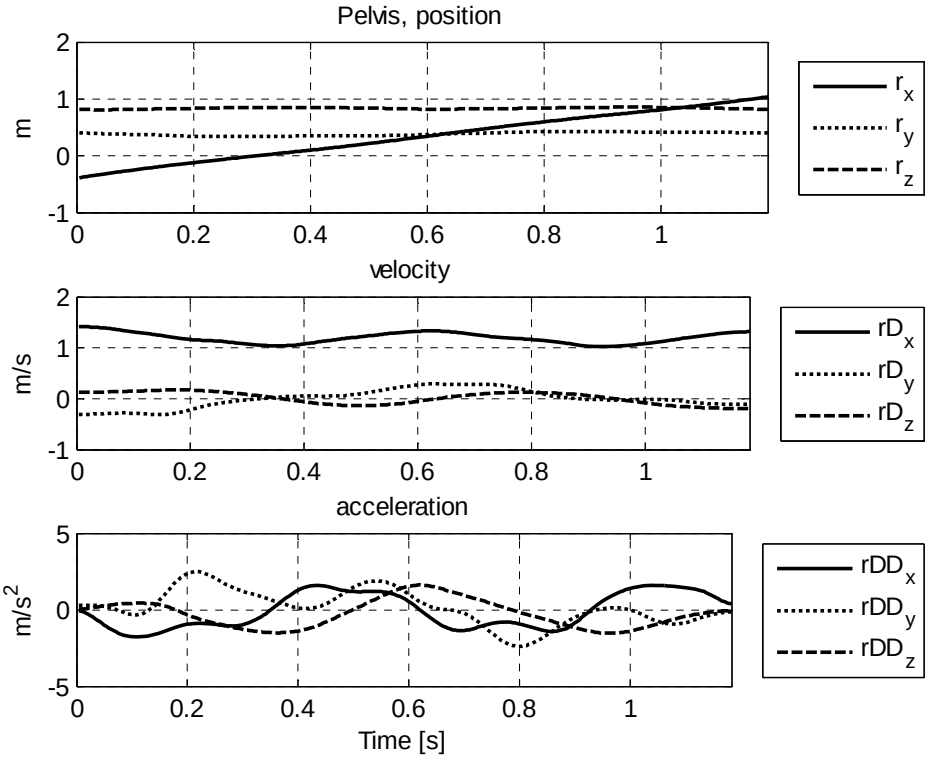

Right Knee, relative Bryant angles
deg
100
50
0
-50
0
0.2
0.4
0.6
0.8
1
θξ
θη
θζ
relative angular velocity
rad/s
10
0
-10
ωξ
ωη
ωζ
relative angular acceleration
rad/s2
100
0
-100
ωDξ
ωDη
ωDζ
Time [s]

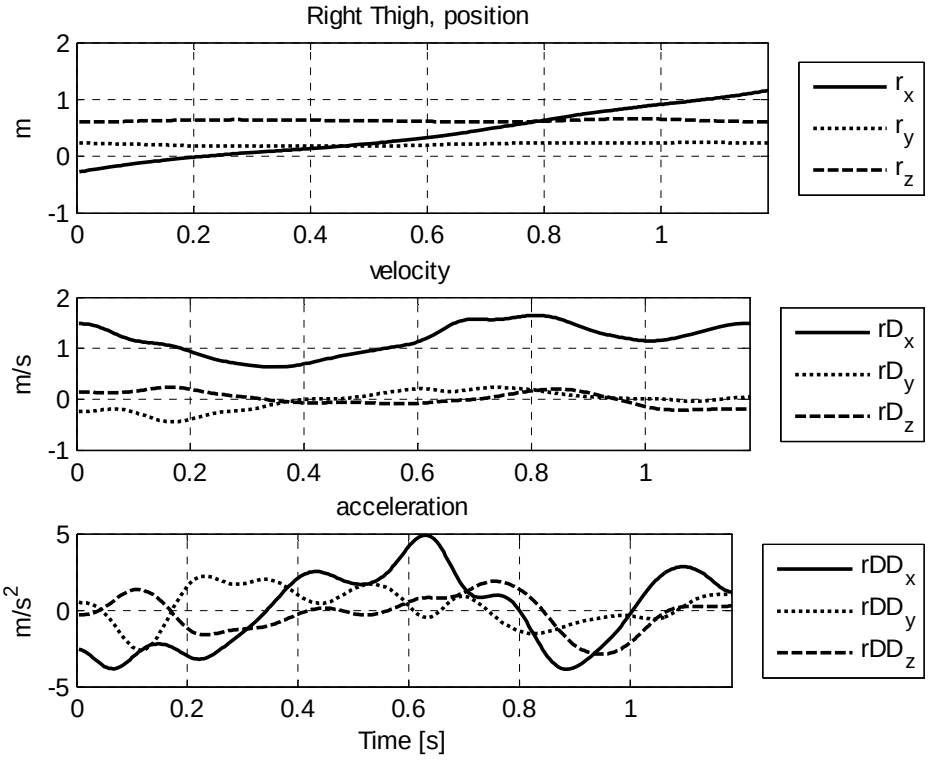

Right Ankle, relative Bryant angles
deg
10
0
-10
-20
0
0.2
0.4
0.6
0.8
1
θξ
θη
θζ
relative angular velocity
rad/s
5
0
-5
ωξ
ωη
ωζ
relative angular acceleration
rad/s2
50
0
-50
-100
ωDξ
ωDη
ωDζ
Time [s]

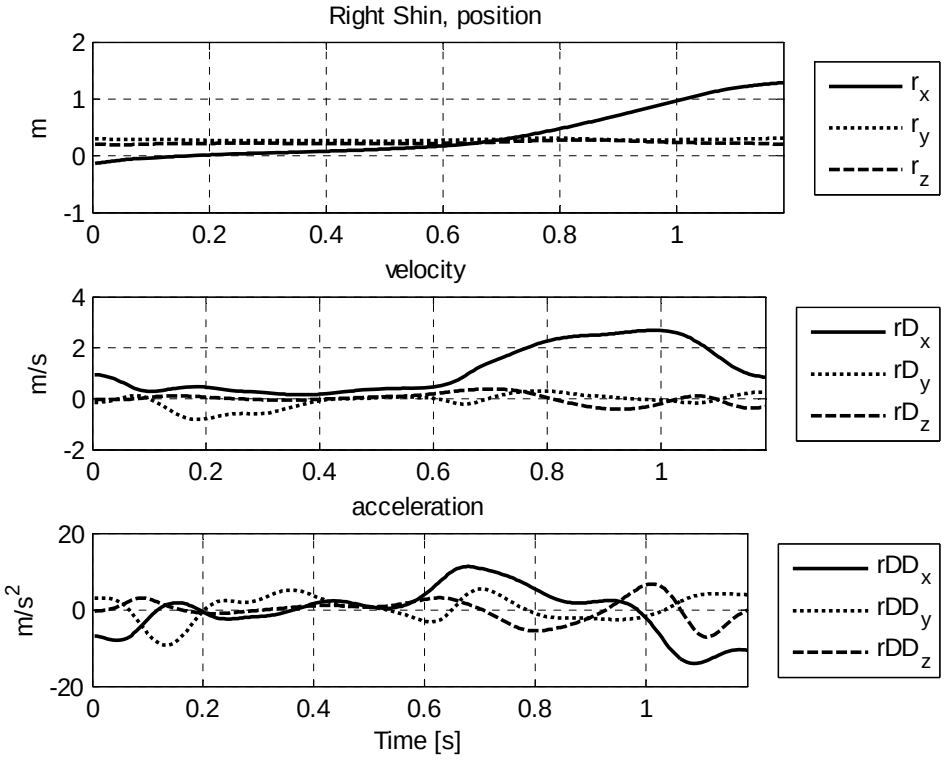
Body Kinematics

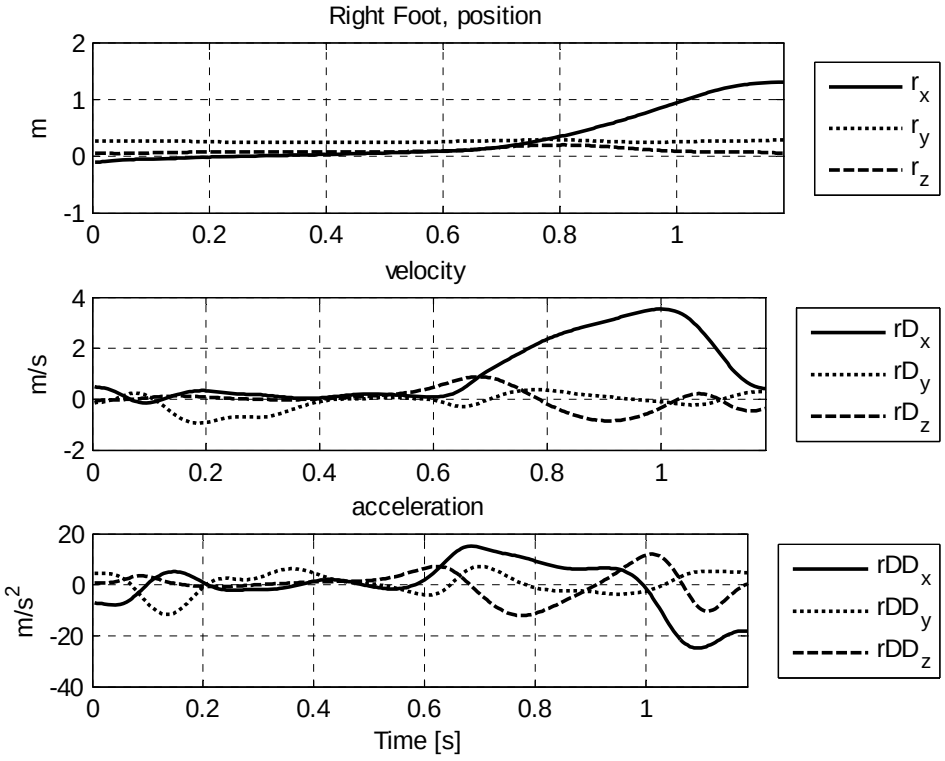




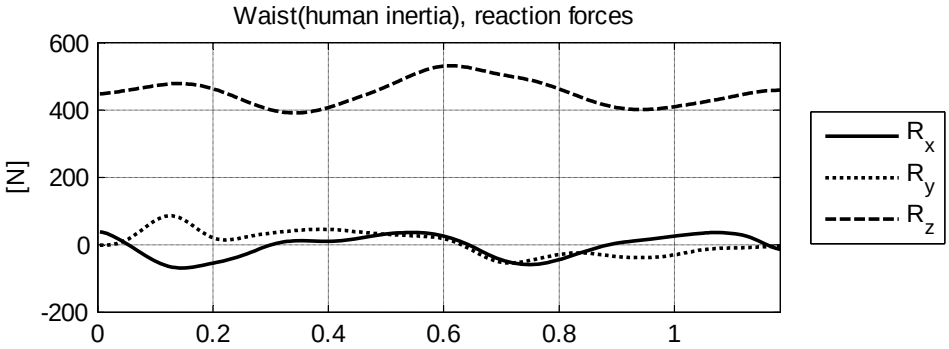

Pelvis, position
m
r_x
r_y
r_z
velocity
m/s
rD_x
rD_y
rD_z
acceleration
m/s^2
rDD_x
rDD_y
rDD_z
Time [s]

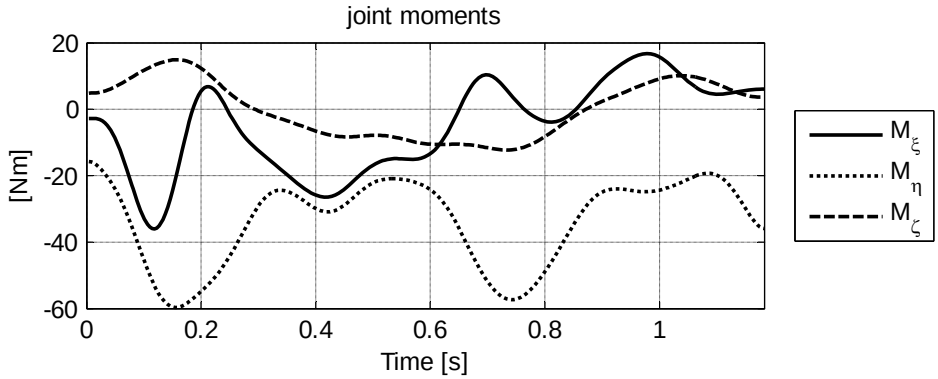

Right Thigh, position
m
r_x
r_y
r_z
velocity
m/s
rD_x
rD_y
rD_z
acceleration
m/s^2
rDD_x
rDD_y
rDD_z
Time [s]

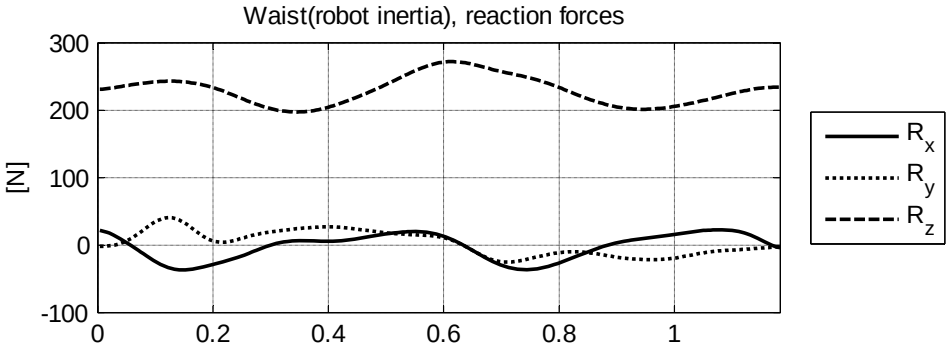

Right Shin, position
m
velocity
m/s
acceleration
m/s2
Time [s]
rx
ry
rz
rDx
rDy
rDz
rDDx
rDDy
rDDz

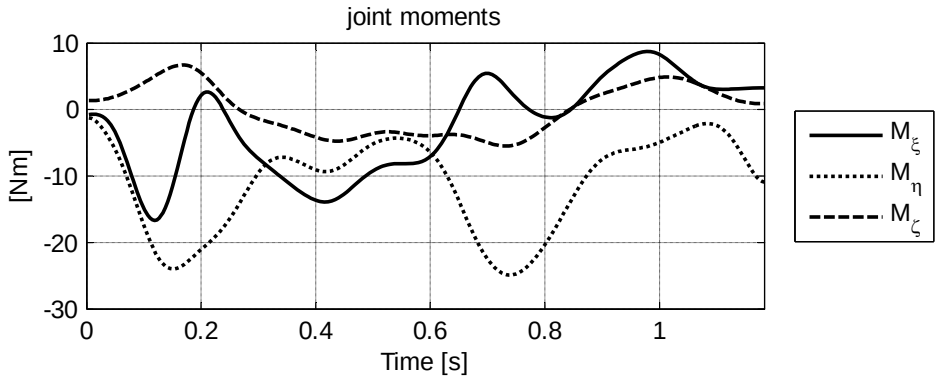

Right Foot, position
m
velocity
m/s
acceleration
m/s2
Time [s]
rx
ry
rz
rDx
rDy
rDz
rDDx
rDDy
rDDz

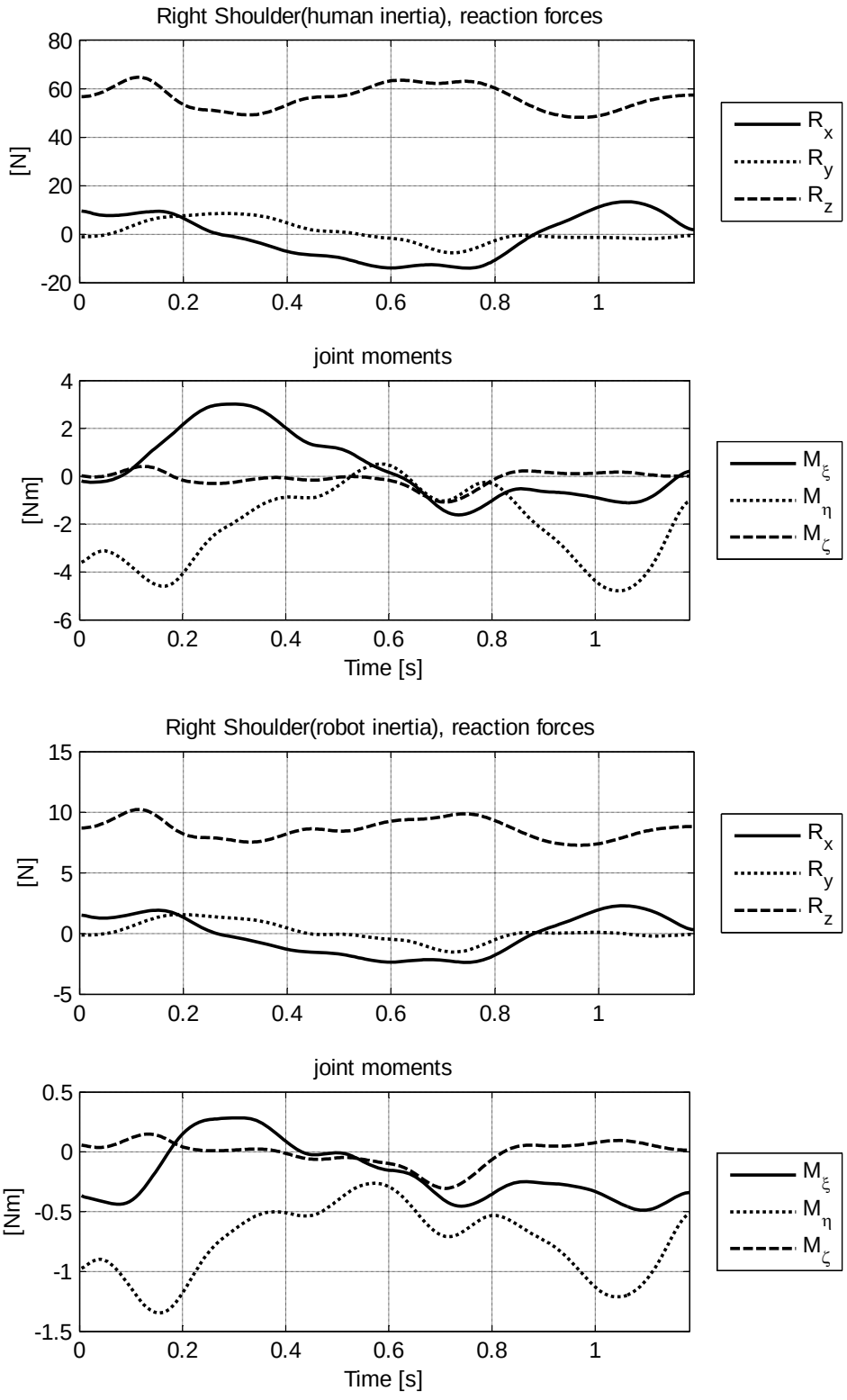
Reaction Forces and Moments

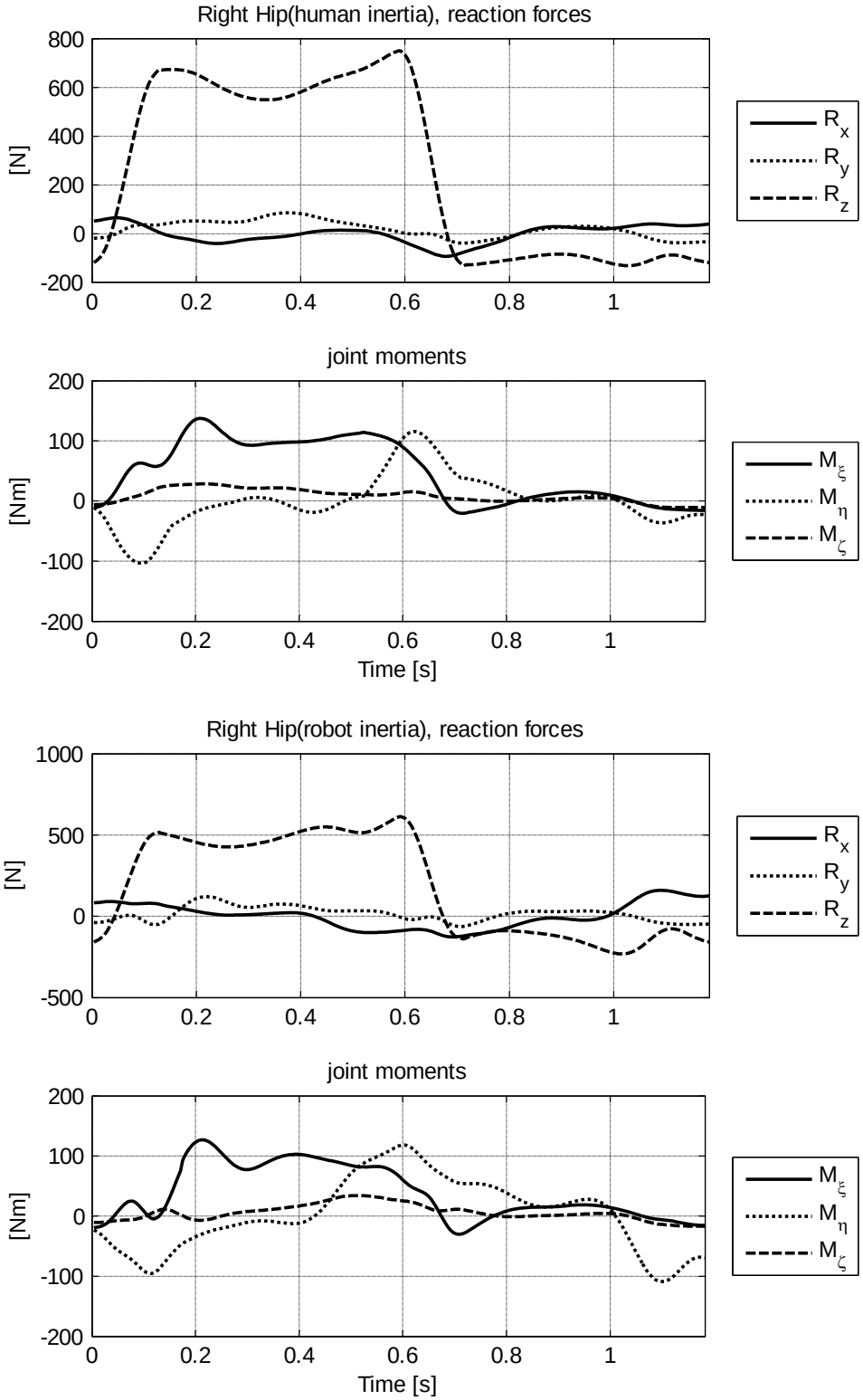


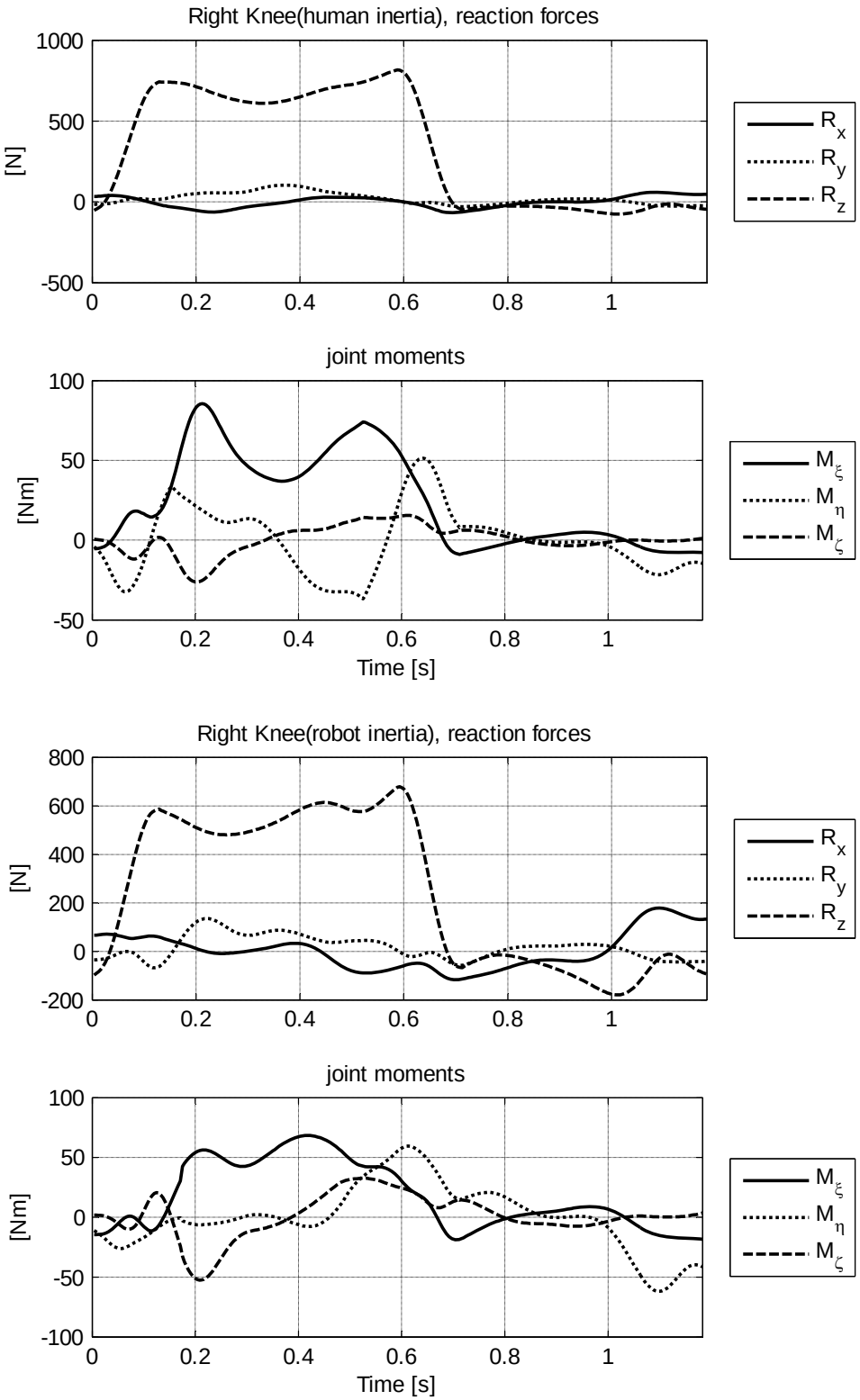


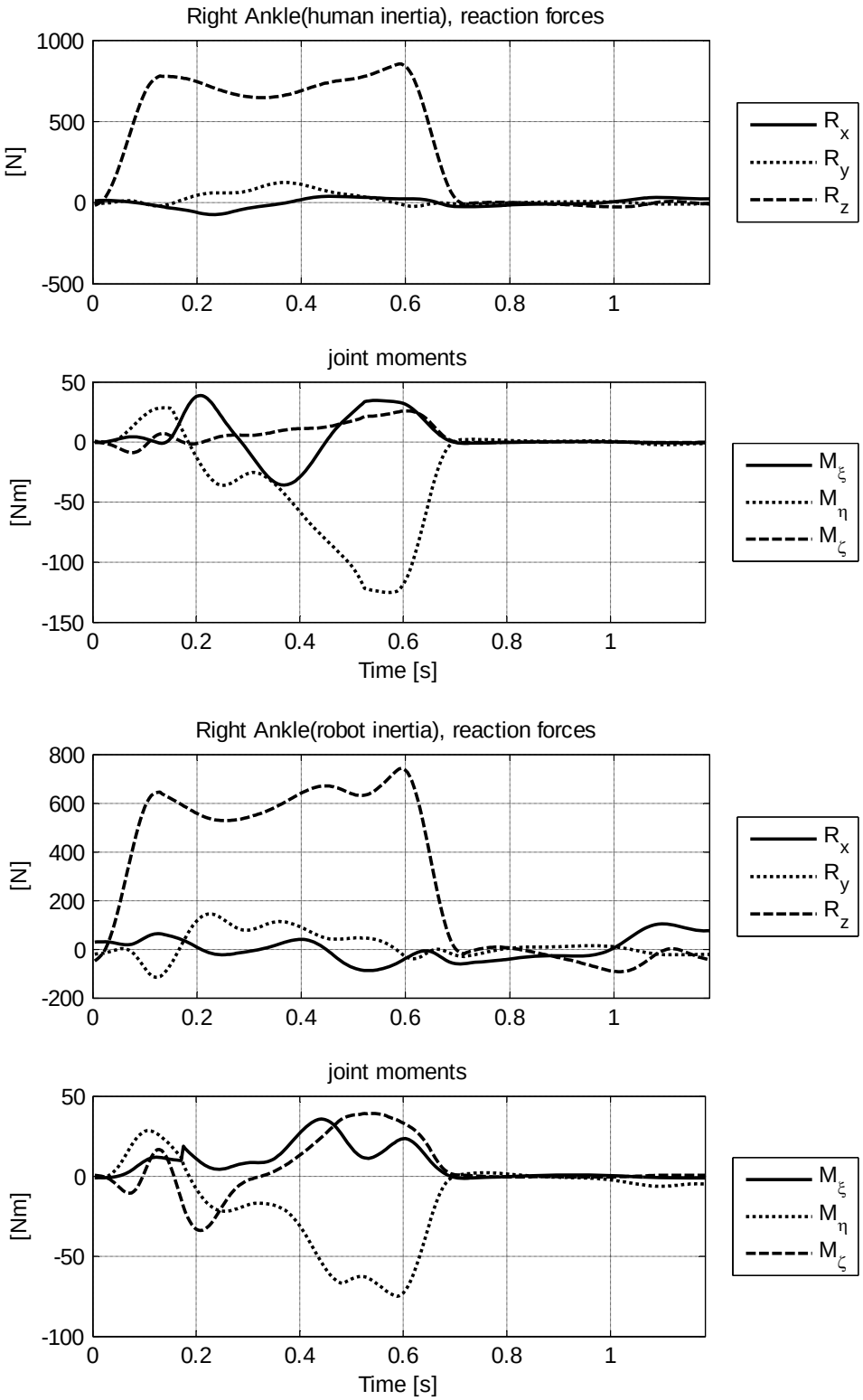





Right Shoulder(human inertia), reaction forces
[N]
80
60
40
20
0
-20
0
0.2
0.4
0.6
0.8
1
R_x
R_y
R_z
joint moments
[Nm]
4
2
0
-2
-4
-6
Time [s]
M_ξ
M_η
M_ζ
Right Shoulder(robot inertia), reaction forces
[N]
15
10
5
0
-5
joint moments
[Nm]
0.5
-0.5
-1
-1.5


Right Hip(human inertia), reaction forces
[N]
800
600
400
200
0
-200
0
0.2
0.4
0.6
0.8
1
R_x
R_y
R_z
joint moments
[Nm]
200
100
0
-100
-200
0
0.2
0.4
0.6
0.8
1
Time [s]
M_ξ
M_η
M_ζ
Right Hip(robot inertia), reaction forces
[N]
1000
500
0
-500
0
0.2
0.4
0.6
0.8
1
R_x
R_y
R_z
joint moments
[Nm]
200
100
0
-100
-200
0
0.2
0.4
0.6
0.8
1
Time [s]
M_ξ
M_η
M_ζ


Right Knee(human inertia), reaction forces
[N]
1000
500
0
-500
0
0.2
0.4
0.6
0.8
1
R_x
R_y
R_z
joint moments
[Nm]
100
50
0
-50
0
0.2
0.4
0.6
0.8
1
Time [s]
M_ξ
M_η
M_ζ
Right Knee(robot inertia), reaction forces
[N]
800
600
400
200
0
-200
0
0.2
0.4
0.6
0.8
1
R_x
R_y
R_z
joint moments
[Nm]
100
50
0
-50
-100
0
0.2
0.4
0.6
0.8
1
Time [s]
M_ξ
M_η
M_ζ


Right Ankle(human inertia), reaction forces
[N]
1000
500
0
-500
0
0.2
0.4
0.6
0.8
1
R_x
R_y
R_z
joint moments
[Nm]
50
0
-50
-100
-150
Time [s]
M_ξ
M_η
M_ζ
Right Ankle(robot inertia), reaction forces
800
600
400
200
joint moments
-100

Appendix IJ - Alternative Foot Model

This appendix presents an alternative foot contact model, which was also implemented and tested in the forward dynamic analysis, but proved inferior when applying large time steps.

Model description

To apply vertical forces from the surroundings, in this case the floor, the foot model is build as illustrated in Figure 24, by simulating springs in each corner of the foot. This way of modeling prevents major forces to occur due to impact between the foot and floor, since the springs gradually slows down the foot prior to impact.

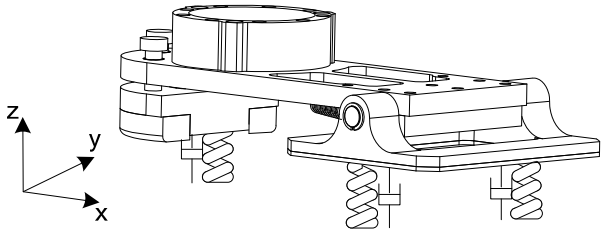


Figure 24: The foot is modeled as a rigid body, with three pairs of parallel coupled springs and dampers.

To apply forces to the foot in the x-y plane, and thereby prevent the robot from sliding on the floor, a suitable friction model is to be applied, which will be described later.

To make it possible for the robot to obtain a human gait pattern, the spring and damper coefficients must be determined with dedication e.g. if the spring constant applied is too large it may cause the model to jump away, and if it is too small it will cause the foot to go through the virtual floor, which is defined as the global x – y plane. A medium must therefore be found between the two extremes.

Springs

The springs are defined as being 10mm long. This means that each individual spring is activated when the vertical distance of each corner of the foot is less than or equal to 10mm. The spring constant k is determined from the assumption that the forces from the three compressed springs should be equal to the normal force acting on the foot while the robot is standing on one leg. The expression for this force was multiplied by 1.5, determined iteratively, to consider the dynamics when the robot is going through a cycle.

$$k = 1.5\frac{1}{3}\frac{m \cdot g}{d}$$

Where d is the length of the springs and m is the total mass of the robot and g is the acceleration due to gravity.

Dampers

The dampers are defined as viscous dampers. The damping coefficient is set to achieve critical damping to avoid any oscillations. The critical damping coefficient is determined as follows (Rao, 2004 p.141).

$$c_c = 2\sqrt{km}$$

The damping coefficient is subsequently determined as a linear function of the compression of the spring and damper, where the damping coefficient consequently will be equal to c_c when the spring and damper is fully compressed. This is to avoid any major forces in the foot on impact with the ground, caused by the damper due to the high velocity of the foot on impact, this way the damping effect is gradually increased. The damping coefficient is updated from the following expression for each time step in the analysis.

$$c = c_c \frac{dL}{L}$$

Where L is the initial length of the spring, and dL is the compression of the spring.

Friction

When applying a friction model it is important that the chosen model is capable of handling the discontinuities that occurs in the stick slip transition phase, which is a major problem in e.g. Coulomb's friction law. (Quinn, 2004) presented a modified model of Coulomb's friction law where the expressions that describe the friction force, is divided in two. One expression describes the friction force when the horizontal velocity v is below a specified value ε, which is set to 0.02m/s. The other describes the friction force when the horizontal velocity is above ε, this is equivalent to Coulomb's friction law.

$$F_\mu = \begin{cases} -\left(\dfrac{\mu F_N}{\varepsilon}\right)v, & |v| \le \varepsilon \\ -\mu F_N \operatorname{sgn}(v), & |v| > \varepsilon \end{cases}$$

This friction force can be plotted as a function of the horizontal velocity, see Figure 25. A consequence of applying this friction model, is that it is necessary to apply very small time steps in the FDA, otherwise oscillations are introduced.

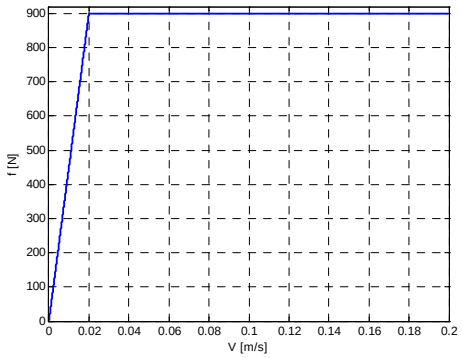


Figure 25 :The friction force fades out as the velocity turns to zero.

Appendix K: Joint Loads from FDA

This appendix presents the joint loads of all revolute joints in the right side of the robot, including the waist, obtained from the forward dynamic analysis. The results are for one step, initiated by heel-strike of the right foot. The oscillations in the results from the waist and the shoulder seem to stem from their reference trajectory being zero.

The results are not directly comparable to the results obtained from the inverse dynamic analysis, since very different gait parameters are applied in the two analyses.

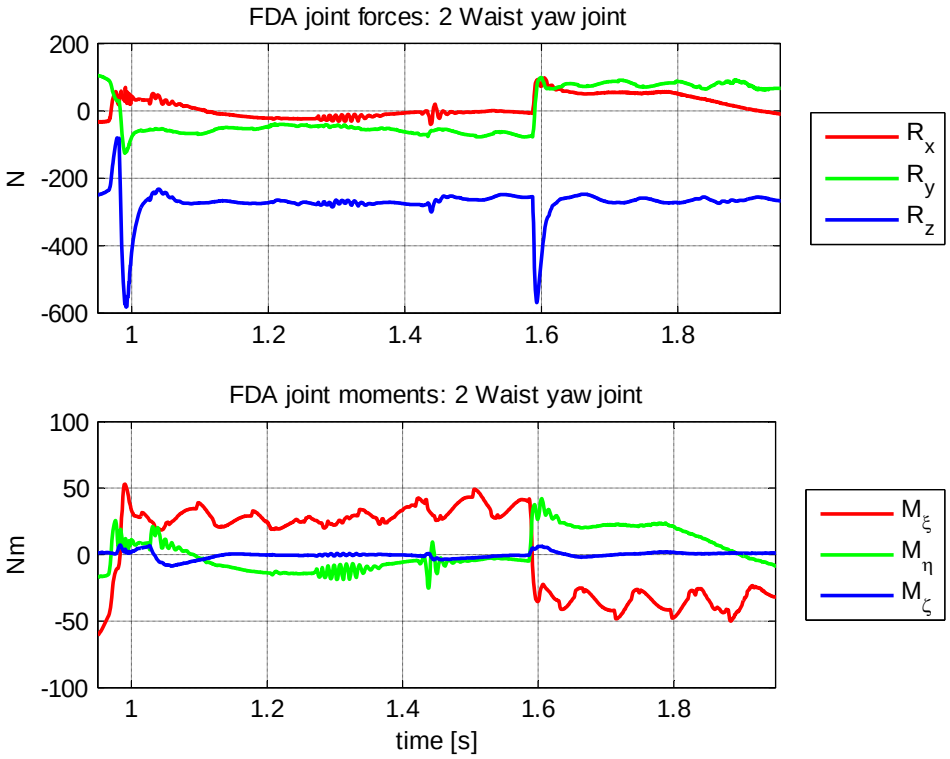


Figure 26

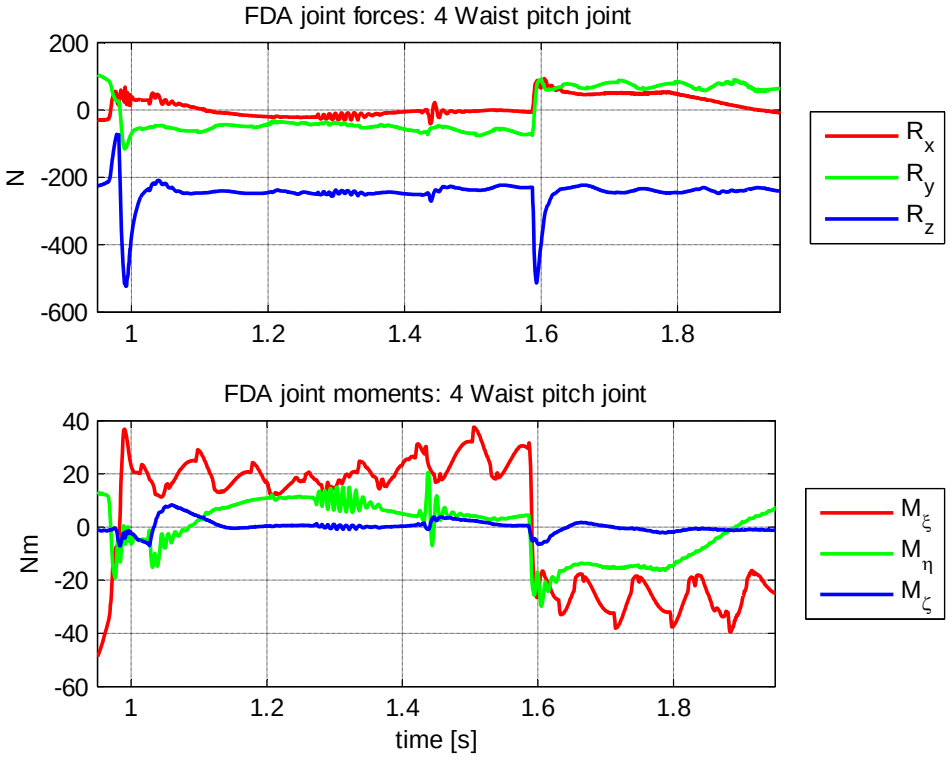


Figure 27

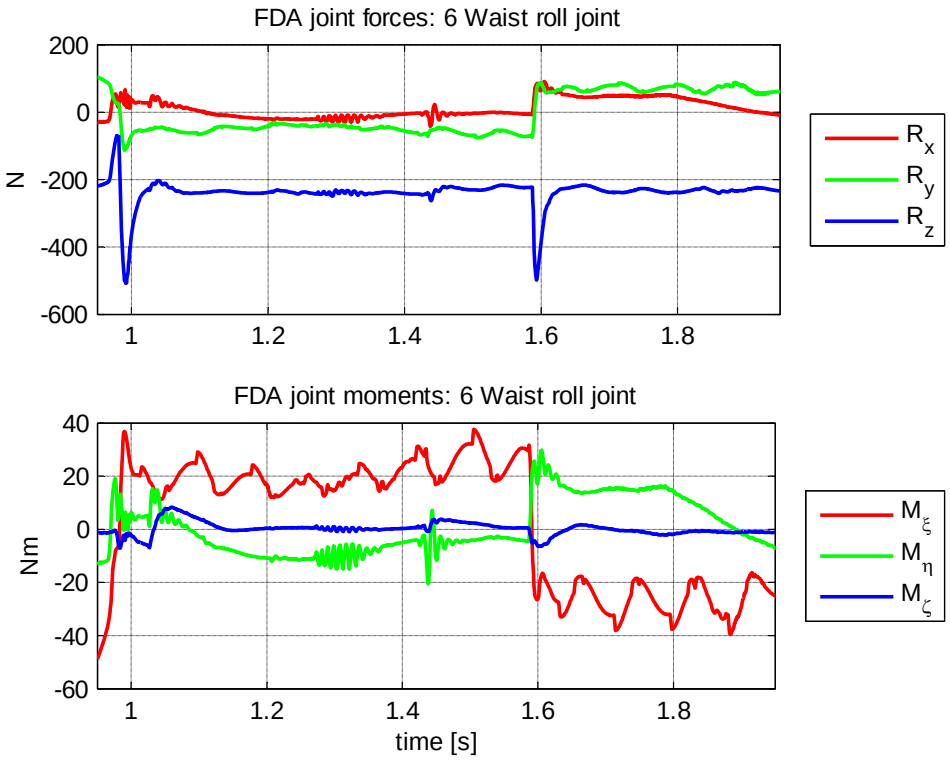


Figure 28

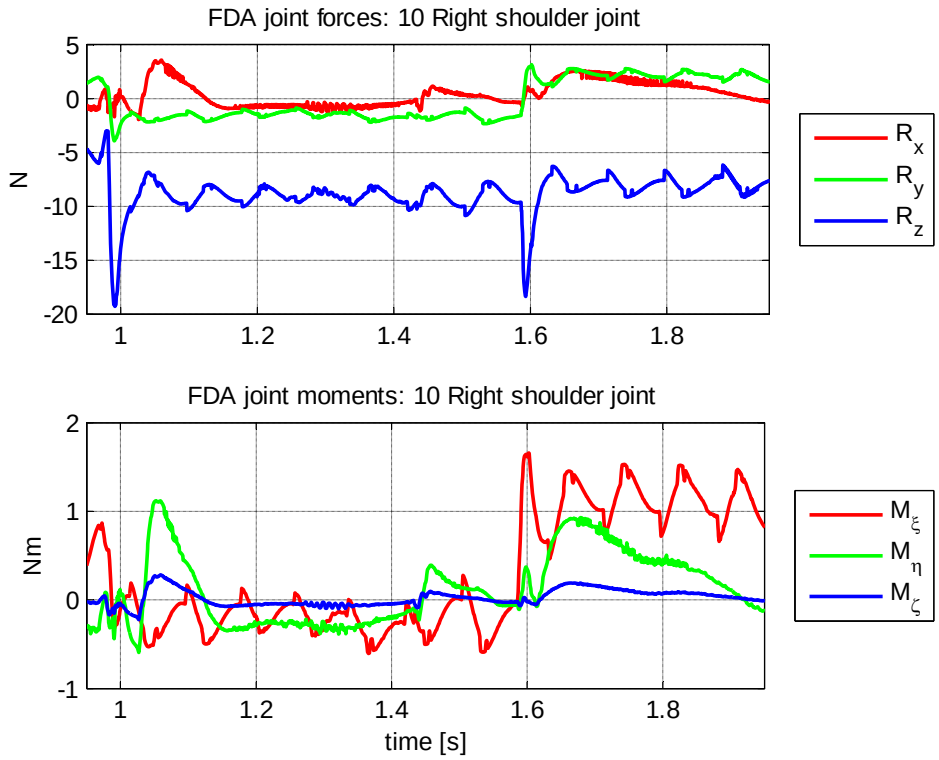


Figure 29

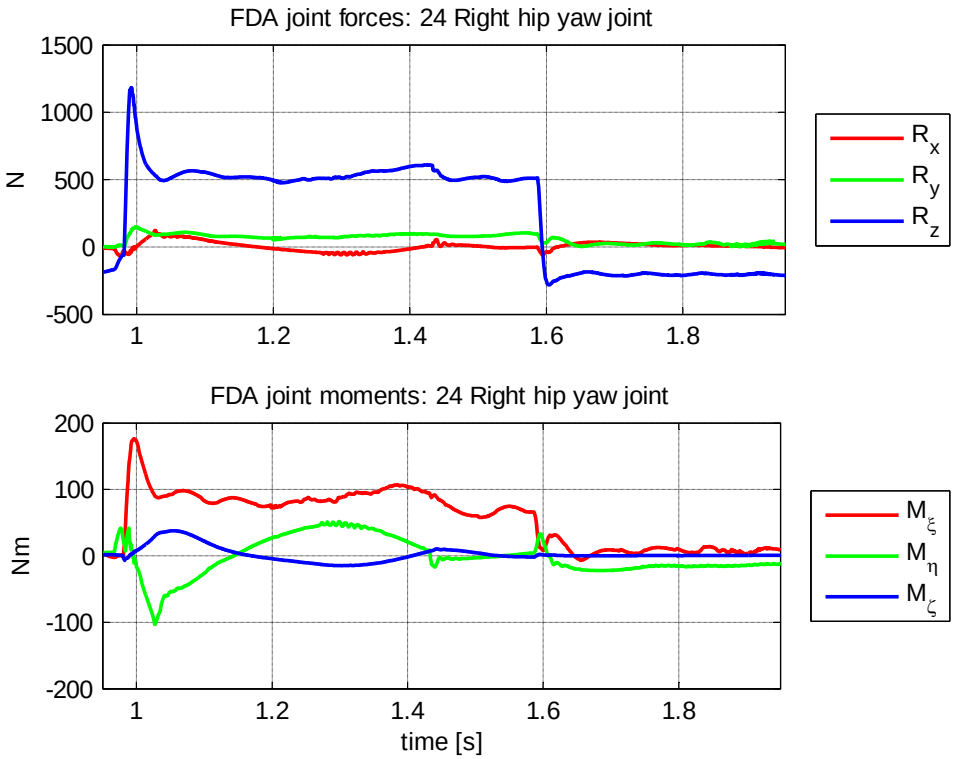


Figure 30

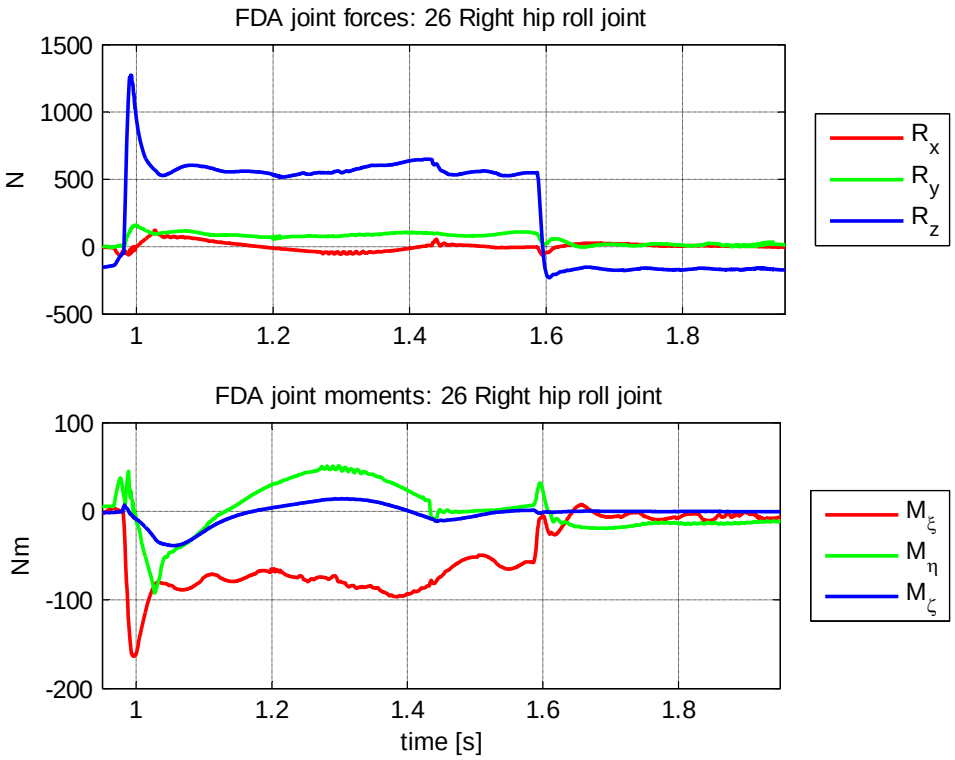
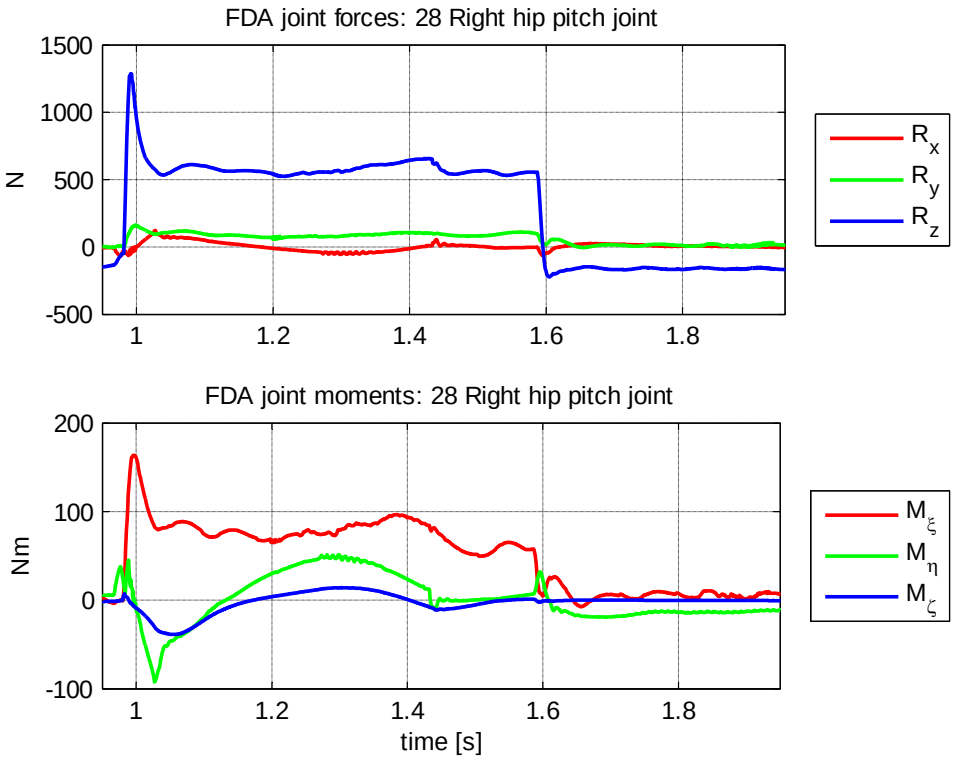

FDA joint forces: 26 Right hip roll joint
N
1500
1000
500
0
-500
1
1.2
1.4
1.6
1.8
R_x
R_y
R_z
FDA joint moments: 26 Right hip roll joint
Nm
100
0
-100
-200
time [s]
M_ξ
M_η
M_ζ

Figure 31


FDA joint forces: 28 Right hip pitch joint
N
1500
1000
500
0
-500
1
1.2
1.4
1.6
1.8
R_x
R_y
R_z
FDA joint moments: 28 Right hip pitch joint
Nm
200
100
0
-100
time [s]
M_ξ
M_η
M_ζ

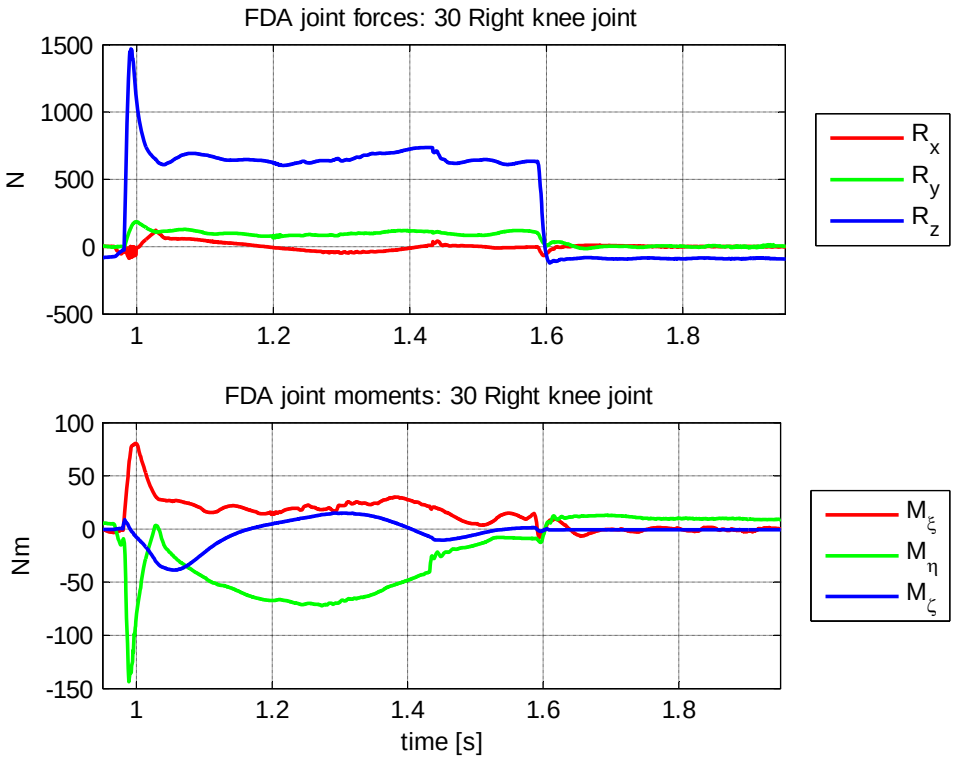

FDA joint forces: 30 Right knee joint
1500
1000
500
0
-500
N
1
1.2
1.4
1.6
1.8
R_x
R_y
R_z
FDA joint moments: 30 Right knee joint
100
50
0
-50
-100
-150
Nm
1
1.2
1.4
1.6
1.8
time [s]
M_ξ
M_η
M_ζ

Figure 32

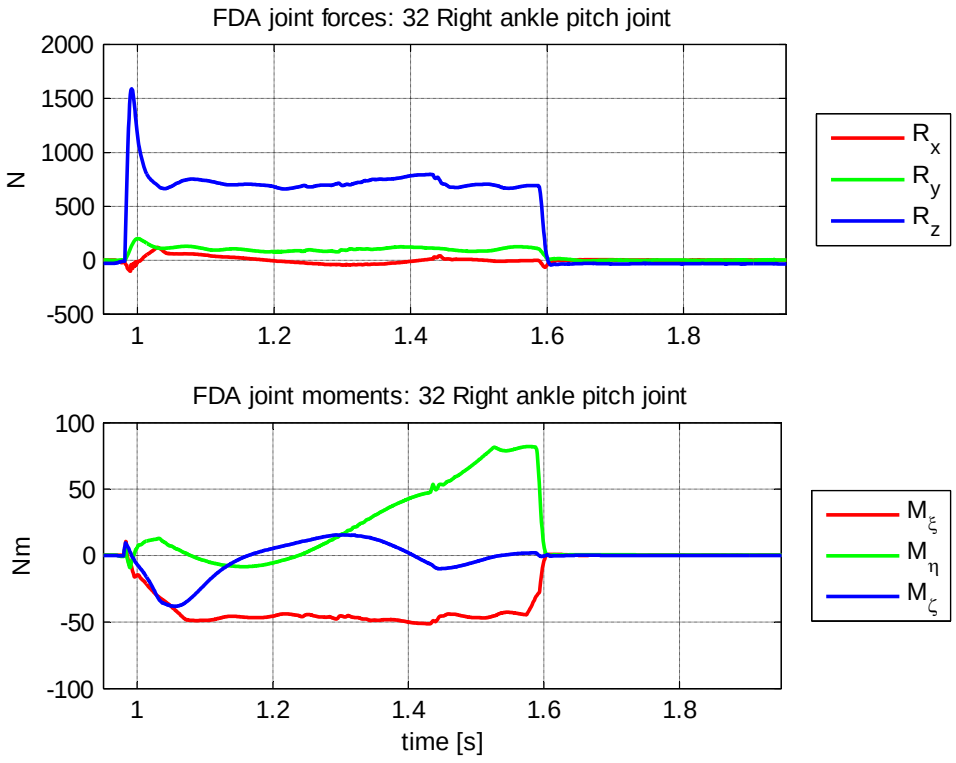


Figure 33

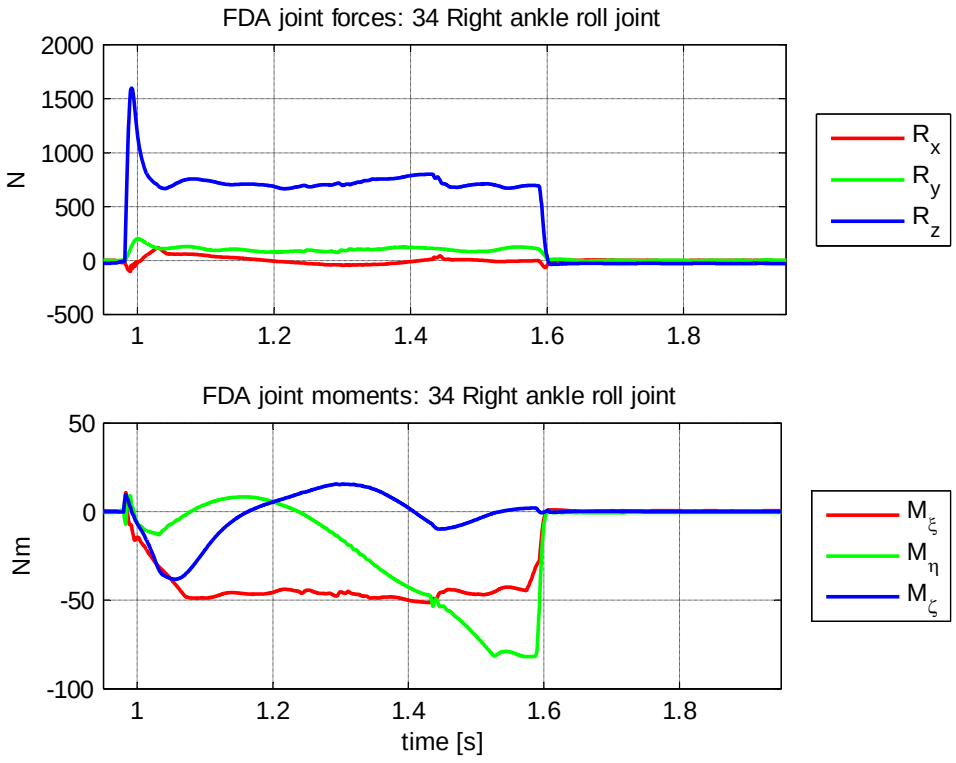


Figure 34

Appendix L – Project Participants

The project will mainly be managed by a steering committee, consisting of DP, JH, JSt, DB, MH, ToA, TDN, OM, & MP, which will meet regularly. Other participants will meet on a less frequent basis.

All participants are listed bellow

Name		Department	Mail	Phone(local)
Jan Helbo	JH	ES/ A&C	jan@control.aau.dk	8738
Dan Banderi	DB	ES/ A&C	danji@control.aau.dk	8756
Ole Madsen	OM	Production	i9om@iprod.aau.dk	8968
Trung Dung Ngo	TDN	CISS	dungnt@cs.aau.dk	7515
Henrik Vie Christensen	HVC	ES/ A&C	vie@control.aau.dk	8767
Stranhinja Dosen	SD	SMI	sdosen@hst.aau.dk	20132276
Mark De Zee	MSZ	SMI/IME	mdz@hst.aau.dk	8818
Pascal Madeleine	PM	SMI/HST	pm@hst.aau.dk	8833
Jacob B. Andersen	JBA	SMI/HST	jaa@hst.aau.dk	8832
Dejan Popovic	DP	SMI/HST	dbp@hst.aau.dk	8726
Michael Rygaard Hansen	MRH	IME	mrh@ime.aau.dk	9635
Jakob Stoustrup	JSt	ES/A&C	jakob@control.aau.dk	8749
Torben O. Andersen	ToA	IET	toa@iet.aau.dk	9269
Ole Borch	OB	ES/A&C	borch@control.aau.dk	8733
Thomas Sinkjær	TS	SMI/HST	ts@hst.aau.dk	8825
John Rasmussen	JR	IME	jr@ime.aau.dk	9307
Mikkel Pedersen	MP	IME	mikkelmelters@gmail.com	51948975
Allan Agerbo Nielsen	AAN	IME	agerboster@gmail.com	23847860
Lars F. Christensen	LFC	IME	lars_fuglsang@hotmail.com	29263901

Appendix M – Extreme Static Joint Torques

This appendix describes the calculation of estimates of the extreme joint torques which can occur. The torques are calculated statically, based on the situations sketched in Figure 35. The mass is determined by summation of the mass of all the body parts positioned above a given joint.

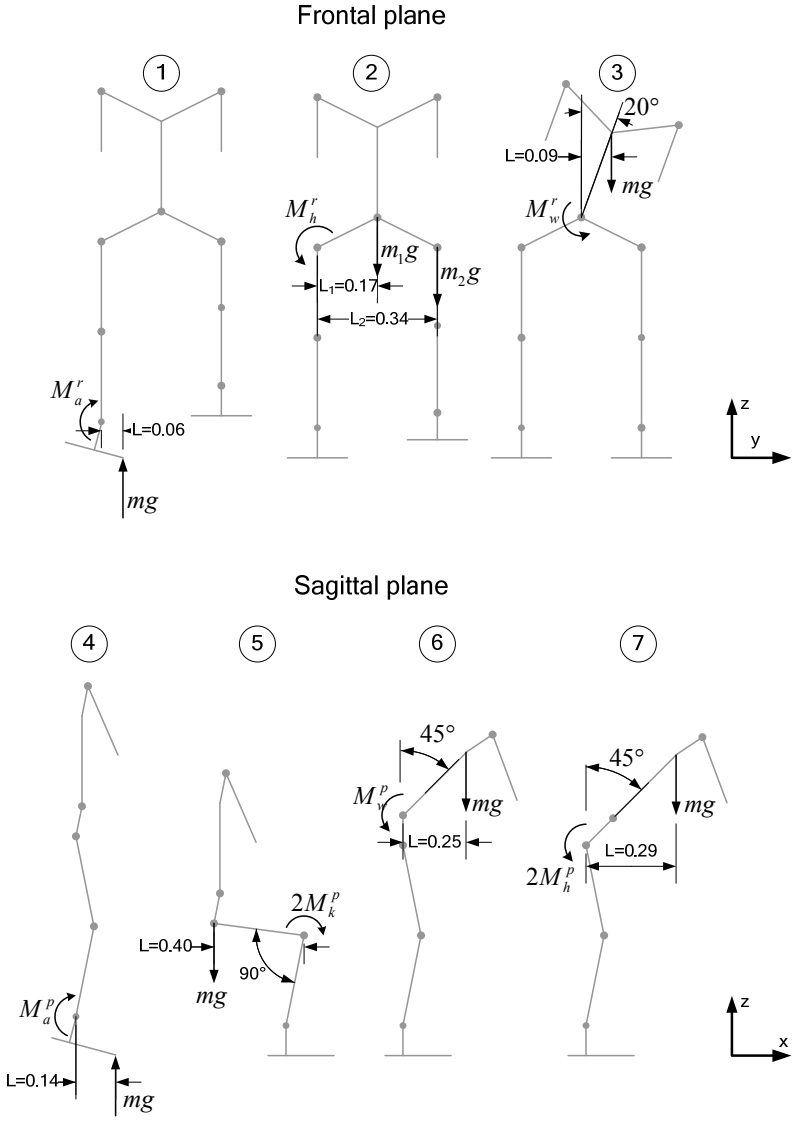


Figure 35: Free body diagrams used for determination of extreme static joint torques. The superscripts denote the axis roll/pitch/yaw and the subscript denotes the joint initial ankle/knee/hip/waist.

$m_{torso} = 35.5kg \quad m_{pelvis} = 7.9kg \quad m_{arm} = 5.8kg$

$m_{thigh} = 6.8kg \quad m_{shin} = 3.6kg \quad m_{foot} = 1.1kg$

$m_{tot} = 78kg$

1 Ankle roll joint

$m = m_{tot} = 78kg$

$L = 0.06m$

$M_a^r = mgL = 46Nm$

2 Hip roll joint

$m_1 = m_{torso} + m_{pelvis} + 2m_{arm} = 37kg \quad m_2 = m_{thigh} + m_{shin} + m_{foot} = 11.5kg$

$L_1 = 0.17m \quad L_2 = 0.34m$

$M_h^r = m_1 g L_1 + m_2 g L_2 = 100Nm$

3 Waist roll joint

$m = m_{torso} = 35.5kg$

$L = 0.12m$

$M_w^r = mgL = 42Nm$

4 Ankle pitch joint

$m = m_{tot} = 78kg$

$L = 0.14m$

$M_a^p = mgL = 107Nm$

5 Knee pitch joint

$m = m_{torso} + 2m_{arm} + m_{pelvis} + \frac{1}{2}2m_{thighs} = 61.8kg$

$L = 0.40m$

$M_k^p = \frac{1}{2}mgL = 121Nm$

6 Waist pitch joint

$m = m_{torso} + 2m_{arm} = 47kg$

$L = 0.25m$

$M_w^p = mgL = 115Nm$

7 Hip pitch joint

$m = m_{torso} + 2m_{arm} + m_{pelvis} = 55kg$

$L = 0.29m$

$M_h^p = \frac{1}{2}mgL = 78Nm$

Appendix N –Data Applied in FDA

This appendix list the inertia properties, geometric specifications and joint/body relations used in the FDA, as well as the total gearing ratio for all joints. All quantities are discussed in the main report.

Inertia properties

Body	Mass [kg]	Inertia [kg*m^2] around CoM		
Pelvis	4.90	$I_{\xi\xi}$ = 0.0547	$I_{\eta\eta}$ = 0.0129	$I_{\zeta\zeta}$ = 0.0657
Waist pitch joint	2.27	$I_{\xi\xi}$ = 0.00988	$I_{\eta\eta}$ = 0.00678	$I_{\zeta\zeta}$ = 0.00587
Waist cross axle	0.75	$I_{\xi\xi}$ = 0.00035	$I_{\eta\eta}$ = 0.000956	$I_{\zeta\zeta}$ = 0.00121
Torso	21.6	$I_{\xi\xi}$ = 0.826	$I_{\eta\eta}$ = 0.586	$I_{\zeta\zeta}$ = 0.502
Arms	0.90	$I_{\xi\xi}$ = 0.0307	$I_{\eta\eta}$ = 0.0419	$I_{\zeta\zeta}$ = 0.0125
Hip roll joints	3.07	$I_{\xi\xi}$ = 0.0161	$I_{\eta\eta}$ = 0.0127	$I_{\zeta\zeta}$ = 0.00932
Hip cross axles	0.64	$I_{\xi\xi}$ = 0.00047	$I_{\eta\eta}$ = 0.000324	$I_{\zeta\zeta}$ = 0.000644
Thighs	6.66	$I_{\xi\xi}$ = 0.099	$I_{\eta\eta}$ = 0.0894	$I_{\zeta\zeta}$ = 0.024
Shins	4.97	$I_{\xi\xi}$ = 0.0726	$I_{\eta\eta}$ = 0.0686	$I_{\zeta\zeta}$ = 0.013
Ankle cross axles	0.47	$I_{\xi\xi}$ = 0.00018	$I_{\eta\eta}$ = 0.000358	$I_{\zeta\zeta}$ = 0.000434
Feet	2.75	$I_{\xi\xi}$ = 0.00789	$I_{\eta\eta}$ = 0.0167	$I_{\zeta\zeta}$ = 0.0145
Toes	0.02	$I_{\xi\xi}$ = 0.000139	$I_{\eta\eta}$ = 0.000426	$I_{\zeta\zeta}$ = 0.000312
Total mass	78.1	$I_{\xi\xi}$ = 10.9	$I_{\eta\eta}$ = 10.1	$I_{\zeta\zeta}$ = 1.27

PT Components	Mass [kg]	Inertia around axis of rotation [kg*m^2]
Maxon RE 60W	0.283	J_{60} = 33.5e-7
Maxon RE 90W	0.340	J_{90} = 67.6e-7
Maxon RE 150W	0.480	J_{150} = 138e-7
HD CPU-S-14	-	J_{14} = 0.091e-4
HD CPU-S-17	-	J_{17} = 0.193e-4
HD CPU-S-20	-	J_{20} = 0.404e-4

Motors/joints	m_{rotor}	$J_{gear} + J_{rotor}$	J_{belt} (est).	u_{belt}	u_{gear}	u_{tot}
Waist yaw	m_{60}	$J_{60} + J_{17}$	0.1080e-4	48/16	100	300
Waist pitch	m_{150}	$J_{150} + J_{17}$	0.3171e-4	56/18	120	373
Waist roll	m_{90}	$J_{90} + J_{14}$	0.2911e-4	62/18	100	344
Shoulder pitch	m_{60}	$J_{60} + J_{14}$	0.0051e-4	20/18	100	111
Hip yaw	m_{60}	$J_{60} + J_{17}$	0.0510e-4	40/16	100	250
Hip roll	2 m_{150}	2 $J_{150} + J_{20}$	0.5576e-4	72/30	120	288
Hip pitch	m_{150}	$J_{150} + J_{20}$	0.2937e-4	60/28	160	342
Knee pitch	2 m_{150}	2 $J_{150} + J_{17}$	0.2111e-4	32/24	100	133
Ankle pitch	2 m_{150}	2 $J_{150} + J_{20}$	0.0421e-4	38/18	100	211
Ankle roll	m_{90}	$J_{90} + J_{14}$	0.0789e-4	44/22	100	200

Joint/body relations

Previous/next body in a RJ is denoted *a* and *b*, relative to the basebody. The axis of rotation is denoted *e*, and given by a unit vector. The last property, drive (*d*) describes a given joint as

1: next body is on a gear output shaft.

2: motor joint.

3: next body is on the gear stator, i.e. opposite of 1.

0: unactuated joint (toes only)

Joint	previous	next	axis	drive
Waist yaw motor	a = 1	b = 2	e = $[0\ 0\ 1]^T$	d = 2
Waist yaw joint	a = 1	b = 3	e = $[0\ 0\ 1]^T$	d = 1
Waist pitch motor	a = 3	b = 4	e = $[0\ 1\ 0]^T$	d = 2
Waist pitch joint	a = 3	b = 5	e = $[0\ 1\ 0]^T$	d = 1
Waist roll motor	a = 7	b = 6	e = $[1\ 0\ 0]^T$	d = 2
Waist roll joint	a = 5	b = 7	e = $[1\ 0\ 0]^T$	d = 3
Left shoulder motor	a = 7	b = 8	e = $[0\ 1\ 0]^T$	d = 2
Left shoulder joint	a = 7	b = 9	e = $[0\ 1\ 0]^T$	d = 1
Right shoulder motor	a = 7	b = 10	e = $[0\ 1\ 0]^T$	d = 2
Right shoulder joint	a = 7	b = 11	e = $[0\ 1\ 0]^T$	d = 1
Left hip yaw motor	a = 1	b = 12	e = $[0\ 0\ 1]^T$	d = 2
Left hip yaw joint	a = 1	b = 13	e = $[0\ 0\ 1]^T$	d = 1
Left hip roll motor	a = 13	b = 14	e = $[1\ 0\ 0]^T$	d = 2
Left hip roll joint	a = 13	b = 15	e = $[1\ 0\ 0]^T$	d = 1
Left hip pitch motor	a = 17	b = 16	e = $[0\ 1\ 0]^T$	d = 2
Left hip pitch joint	a = 15	b = 17	e = $[0\ 1\ 0]^T$	d = 3
Left knee motor	a = 17	b = 18	e = $[0\ 1\ 0]^T$	d = 2
Left knee joint	a = 17	b = 19	e = $[0\ 1\ 0]^T$	d = 1
Left ankle pitch motor	a = 19	b = 20	e = $[0\ 1\ 0]^T$	d = 2
Left ankle pitch joint	a = 19	b = 21	e = $[0\ 1\ 0]^T$	d = 1
Left ankle roll motor	a = 23	b = 22	e = $[1\ 0\ 0]^T$	d = 2
Left ankle roll joint	a = 21	b = 23	e = $[1\ 0\ 0]^T$	d = 3
Right hip yaw motor	a = 1	b = 24	e = $[0\ 0\ 1]^T$	d = 2
Right hip yaw joint	a = 1	b = 25	e = $[0\ 0\ 1]^T$	d = 1
Right hip roll motor	a = 25	b = 26	e = $[1\ 0\ 0]^T$	d = 2
Right hip roll joint	a = 25	b = 27	e = $[1\ 0\ 0]^T$	d = 1
Right hip pitch motor	a = 29	b = 28	e = $[0\ 1\ 0]^T$	d = 2
Right hip pitch joint	a = 27	b = 29	e = $[0\ 1\ 0]^T$	d = 3
Right knee motor	a = 29	b = 30	e = $[0\ 1\ 0]^T$	d = 2
Right knee joint	a = 29	b = 31	e = $[0\ 1\ 0]^T$	d = 1
Right ankle pitch motor	a = 31	b = 32	e = $[0\ 1\ 0]^T$	d = 2
Right ankle pitch joint	a = 31	b = 33	e = $[0\ 1\ 0]^T$	d = 1
Right ankle roll motor	a = 35	b = 34	e = $[1\ 0\ 0]^T$	d = 2
Right ankle roll joint	a = 33	b = 35	e = $[1\ 0\ 0]^T$	d = 3
Left toe	a = 23	b = 36	e = $[0\ 1\ 0]^T$	d = 0
Right toe	a = 35	b = 37	e = $[0\ 1\ 0]^T$	d = 0

Geometry

See main report for numbering information.

s{2,1} = [-80 0 4]'*10^-3
s{3,1} = [22 0 14]'*10^-3
s{24,1} = [-96 -77 4]'*10^-3
s{25,1} = [22 -140 -16]'*10^-3
s{12,1} = [-96 77 4]'*10^-3
s{13,1} = [22 140 -16]'*10^-3
s{1,3} = [4 38 -80]'*10^-3
s{4,3} = [-16 -22 -32]'*10^-3
s{5,3} = [4 38 40]'*10^-3
s{3,5} = [0 0 0]'*10^-3
s{7,5} = [0 0 0]'*10^-3
s{5,7} = [19 0 -341]'*10^-3
s{6,7} = [-54 0 -211]'*10^-3
s{10,7} = [75 -222 84]'*10^-3
s{11,7} = [25 -280 84]'*10^-3
s{8,7} = [75 222 84]'*10^-3
s{9,7} = [25 280 84]'*10^-3
s{7,9} = [-4 -48 347]'*10^-3
s{7,11} = [-4 48 347]'*10^-3
s{1,25} = [38 24 28]'*10^-3
s{26,25} = [-24 -54 77]'*10^-3
s{27,25} = [41 24 -36]'*10^-3
s{1,13} = [38 -24 28]'*10^-3
s{14,13} = [-24 54 77]'*10^-3
s{15,13} = [41 -24 -36]'*10^-3
s{25,27} = [0 0 0]'*10^-3
s{29,27} = [0 0 0]'*10^-3
s{13,15} = [0 0 0]'*10^-3
s{17,15} = [0 0 0]'*10^-3
s{27,29} = [2 41 146]'*10^-3
s{28,29} = [-19 -27 37]'*10^-3
s{30,29} = [13 -27 -34]'*10^-3
s{31,29} = [2 27+14 -165]'*10^-3
s{15,17} = [2 -41 146]'*10^-3
s{16,17} = [-19 27 37]'*10^-3
s{18,17} = [13 27 -34]'*10^-3
s{19,17} = [2 -27-14 -165]'*10^-3
s{29,31} = [-4 -30 252]'*10^-3
s{32,31} = [21 25 -13]'*10^-3
s{33,31} = [-4 -30 -118]'*10^-3
s{17,19} = [-4 30 252]'*10^-3
s{20,19} = [21 -25 -13]'*10^-3
s{21,19} = [-4 30 -118]'*10^-3
s{31,33} = [0 0 0]'*10^-3
s{35,33} = [0 0 0]'*10^-3
s{19,21} = [0 0 0]'*10^-3
s{23,21} = [0 0 0]'*10^-3
s{33,35} = [19 10 46]'*10^-3
s{34,35} = [0 -30 46]'*10^-3
s{37,35} = [132 10 -76]'*10^-3
s{21,23} = [19 -10 46]'*10^-3
s{22,23} = [0 30 46]'*10^-3
s{36,23} = [132 10 -76]'*10^-3
s{35,37} = [0 0 0]'*10^-3
s{23,36} = [0 0 0]'*10^-3
s{1,2}=[0 0 0]'*10^-3
s{3,4}=[0 0 0]'*10^-3
s{7,6}=[0 0 0]'*10^-3
s{7,8}=[0 0 0]'*10^-3
s{7,10}=[0 0 0]'*10^-3
s{1,12}=[0 0 0]'*10^-3
s{13,14}=[0 0 0]'*10^-3
s{17,16}=[0 0 0]'*10^-3
s{17,18}=[0 0 0]'*10^-3
s{19,20}=[0 0 0]'*10^-3
s{23,22}=[0 0 0]'*10^-3
s{1,24}=[0 0 0]'*10^-3
s{25,26}=[0 0 0]'*10^-3
s{29,28}=[0 0 0]'*10^-3
s{29,30}=[0 0 0]'*10^-3
s{31,32}=[0 0 0]'*10^-3
s{35,34}=[0 0 0]'*10^-3

Appendix O – Design Overview

This appendix presents three rendered images of the final robot design; the torso, the pelvis and a leg. These images provides a detailed overview of the position and mounting of the different parts.

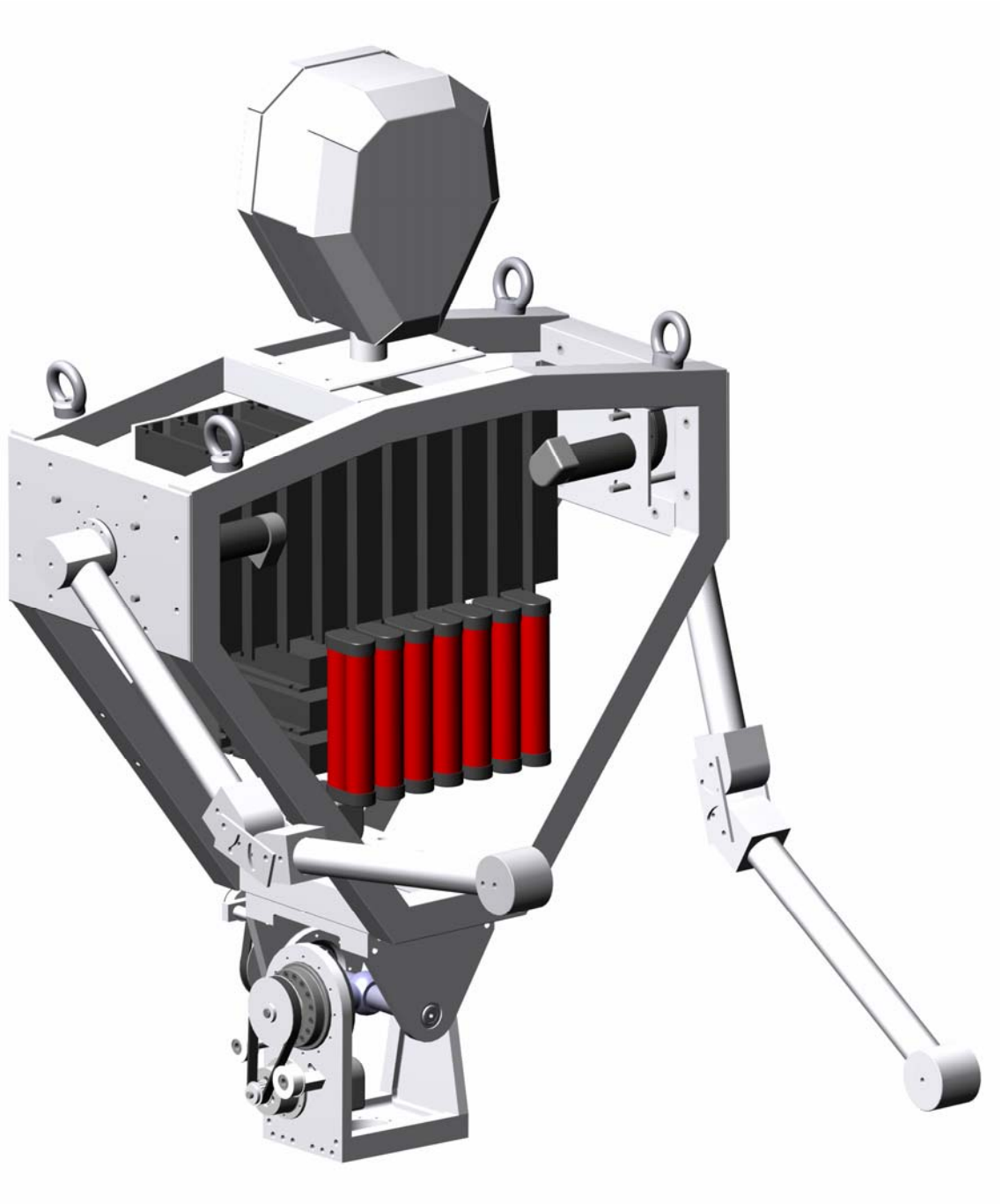


Figure 36: Final design of the torso.

Figure 37: Final design of the pelvis region.

Figure 38: Final leg design.

```
% Inverse dynamic analysis, based on motion capture, by DMS10/gr.43A
clear all; close all; clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% INPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define load case filename: mean_walk, standref, sit, raise, curb
LC='standref';
Data.set='h'; % h= human inertia / r= robot inertia
Dif_precision=0.9999; % Set differentiation precision, 1=max, 0=min:

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialize variables
nBodies = 10;
nJoints =  9;
SampleRate = 240;

% Initialize Body and Joint structs
InitBodyStruct;
InitJointStruct;

% Get input geometry and inertia
if strcmp(Data.set,'h')
    InertiaAndGeometryH;
else
    InertiaAndGeometryR;
end

% Reads recorded motion capture results from files
if strcmp(LC,'mean_walk')
    MeanWalk;  % Read input, take average.
else
    ReadInput; % Read input.
end

% Store constants
FrameSpan = [1:nFrames]';
Tspan = FrameSpan*SampleRate^-1;
Data.nFrames=nFrames;
Data.nJoints=nJoints;
Data.nBodies=nBodies;
Data.LC=LC;
Data.Tspan=Tspan;
Data.FrameSpan=FrameSpan;

% Calculate distribution of forces in feet
ForcePlatform;

% Creates A-matrices for all bodies
CreateAMatrices;

% Smooth rotations (stored in A-matrices)
SmoothRotations;

% Calculate relative transformation matrices
for JointNo = 1:nJoints
```

```
    Joint(JointNo).A_rel = CalcRelTrMatr(Body,Joint,JointNo,nFrames);
end

% Calculate relative rotations in the 3 directions
for JointNo = 1:nJoints
    Joint(JointNo).thetaRel = CalcRelRot(Joint(JointNo).A_rel,nFrames);
end

% Cancel yaw rotation in ankles and add same amount to hips.
% Right leg:
Joint(4).thetaRel(3,:) = Joint(4).thetaRel(3,:) + Joint(8).thetaRel(3,:);
Joint(8).thetaRel(3,:) = 0;
% Left leg:
Joint(5).thetaRel(3,:) = Joint(5).thetaRel(3,:) + Joint(9).thetaRel(3,:);
Joint(9).thetaRel(3,:) = 0;

% Obtain the relative angular velocities
for JointNo = 1:nJoints

    % Differentiate the relative rotations: theta -> thetaDot
    [Joint(JointNo).thetaRelDot junk]= FitDif(Joint(JointNo).thetaRel,Tspan,↙
Dif_precision);

    % Transform to relative angular velocities omega=T(theta)*thetaDot
    for FrameNo=1:nFrames
        Joint(JointNo).omegaRel(:,FrameNo)  = T(Joint(JointNo).thetaRel(:,FrameNo))↙
*Joint(JointNo).thetaRelDot(:,FrameNo);
    end

end

% Obtain the relative angular accelerations by differentiation
for JointNo = 1:nJoints
    [Joint(JointNo).omegaRelDot junk]= FitDif(Joint(JointNo).omegaRel,Tspan,↙
Dif_precision);
end

% Extract extreme values
ExtractValues;

% Calculate absolute angular velocities of joints
CalcAbsAngVel;

% Calculate absolute angular accelerations of joints
CalcAbsAngAcc;

% Calculate absolute position, velocity and acceleration of bodies
CalcAbsPosVelAcc;

% Calculate global Center of Mass
Data.CoM=CalcCoM(Body,nFrames);

% Setup kinetic equation system and implement foot forces
Kinetics;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% OUTPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FirstRun=1;
% Plot BODY kinematics, input body numbers below:
    BodyNos  =[];
    PlotBodyKin(Body,Data,Tspan,BodyNos,1);

% Plot JOINT kinematics, input joint-numbers below:
    JointNos =[];
    PlotJointKin(Joint,Data,Tspan,JointNos,1);

% Plot JOINT kinetics, input joint-numbers below:
    JointNos =[];
    if FirstRun==1 % transform moments to local
        for j=1:length(JointNos)
            J=JointNos(j);
            for i=1:nFrames
                A=Body(Joint(J).RelTo).A(:,:,i);
                M=Joint(J).M(:,i);
                Joint(J).M(:,i)=A'*M;
            end
            FirstRun=0;
        end
    end
    PlotJointReactions(Joint,Tspan,JointNos,Data,0);

% Write results to screen:
    ListJoints(Joint,nJoints);

% % Save results (structs):
%filename = [Data.set '_res_' Data.LC];
%save filename Joint Body Data geo

% Animate results (using Animator.fig)
%   Animator;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Initialize Body struct

% Define body names
Body(1).name  = 'Torso';
Body(2).name  = 'Right Arm';
Body(3).name  = 'Left Arm';
Body(4).name  = 'Pelvis';
Body(5).name  = 'Right Thigh';
Body(6).name  = 'Left Thigh';
Body(7).name  = 'Right Shin';
Body(8).name  = 'Left Shin';
Body(9).name  = 'Right Foot';
Body(10).name = 'Left Foot';

% Insert body numbers to wich a given body is calculated relative to
Body(1).RelTo  = 0;
Body(2).RelTo  = 1;
Body(3).RelTo  = 1;
Body(4).RelTo  = 1;
Body(5).RelTo  = 4;
Body(6).RelTo  = 4;
Body(7).RelTo  = 5;
Body(8).RelTo  = 6;
Body(9).RelTo  = 7;
Body(10).RelTo = 8;

% Insert body numbers to wich a given body is calculated relative to
Body(1).Joint  = 0;
Body(2).Joint  = 2;
Body(3).Joint  = 3;
Body(4).Joint  = 1;
Body(5).Joint  = 4;
Body(6).Joint  = 5;
Body(7).Joint  = 6;
Body(8).Joint  = 7;
Body(9).Joint  = 8;
Body(10).Joint = 9;
```

```
% Initialize Joint struct

% Insert joint names and DOFs
Joint(1).name = 'Waist';
Joint(2).name = 'Right Shoulder';
Joint(3).name = 'Left Shoulder';
Joint(4).name = 'Right Hip';
Joint(5).name = 'Left Hip';
Joint(6).name = 'Right Knee';
Joint(7).name = 'Left Knee';
Joint(8).name = 'Right Ankle';
Joint(9).name = 'Left Ankle';

Joint(10).name = 'Virtual contact point under right foot';
Joint(11).name = 'Virtual contact point under left foot';
Joint(12).name = 'Zero Moment Point';


% Insert joint DOFs
Joint(1).DOF  = 3;
Joint(2).DOF  = 1;
Joint(3).DOF  = 1;
Joint(4).DOF  = 3;
Joint(5).DOF  = 3;
Joint(6).DOF  = 1;
Joint(7).DOF  = 1;
Joint(8).DOF  = 2;
Joint(9).DOF  = 2;

Joint(10).DOF = 0;
Joint(11).DOF = 0;

% Joint(k) connects Body(j) relative to Body(i): connects = [j i]
Joint(1).connects = [4 1];
Joint(2).connects = [2 1];
Joint(3).connects = [3 1];
Joint(4).connects = [5 4];
Joint(5).connects = [6 4];
Joint(6).connects = [7 5];
Joint(7).connects = [8 6];
Joint(8).connects = [9 7];
Joint(9).connects = [10 8];

% Joint letters
Joint(1).letter = 'A';
Joint(2).letter = 'B';
Joint(3).letter = 'C';
Joint(4).letter = 'D';
Joint(5).letter = 'E';
Joint(6).letter = 'H';
Joint(7).letter = 'I';
Joint(8).letter = 'J';
Joint(9).letter = 'K';
Joint(10).letter = 'R';
Joint(11).letter = 'S';
Joint(12).letter = 'T';
```

```
Joint(1).RelTo = 1;
Joint(2).RelTo = 4;
Joint(3).RelTo = 4;
Joint(4).RelTo = 1;
Joint(5).RelTo = 1;
Joint(6).RelTo = 5;
Joint(7).RelTo = 6;
Joint(8).RelTo = 7;
Joint(9).RelTo = 8;
```

```
%Input geometry
geo.s_2_1 = [0 -255 78.9]'*10^-3;
geo.s_3_1 = [0 255 78.9]'*10^-3;
geo.s_4_1 = [0 0 -353]'*10^-3;
geo.s_1_2 = [0 0 280]'*10^-3;
geo.s_1_3 = [0 0 280]'*10^-3;
geo.s_1_4 = [0 0 68]'*10^-3;
geo.s_5_4 = [39 -101-70 -40]'*10^-3;
geo.s_6_4 = [39 101+70 -40]'*10^-3;
geo.s_4_5 = [0 0 203]'*10^-3;
geo.s_7_5 = [0 0 -228]'*10^-3;
geo.s_4_6 = [0 0 203]'*10^-3;
geo.s_8_6 = [0 0 -228]'*10^-3;
geo.s_6_8 = [0 0 174]'*10^-3;
geo.s_10_8= [0 0 -245]'*10^-3;
geo.s_5_7 = [0 0 174]'*10^-3;
geo.s_9_7 = [0 0 -245]'*10^-3;
geo.s_7_9 = [-35 0 71]'*10^-3;
geo.s_8_10= [-35 0 71]'*10^-3;
% Arm to wrist
geo.s_w_2 = [281.97 0 -254.15]'*10^-3;
geo.s_w_3 = [281.97 0 -254.15]'*10^-3;
% Foot geometry
geo.s_t_9 = [135 0 -14]'*10^-3;
geo.s_h_9 = [-70 0 -32]'*10^-3;
geo.s_t_10= [135 0 -14]'*10^-3;
geo.s_h_10= [-70 0 -32]'*10^-3;
geo.foot_width=120*10^-3;

[M Body]=MassMatrix(Body);
Data.M=M;
```

```
% Final definition of mass, inertia and geometrical properties.

m=14.2;
Ixx = 0.3233;       Ixy = -0.003231;    Ixz = 0.00719;
Iyx = -0.003231;    Iyy = 0.08793;      Iyz = -0.009383;
Izx = 0.00719;      Izy = -0.009383;    Izz = 0.2827;
BodyNo=4;
Body(BodyNo).m=m; Body(BodyNo).J=[Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz];
geo.s_1_4=[28.15 7.31 142.25]'*10^-3;
geo.s_6_4=[32.39 149.14 -72.74]'*10^-3;
geo.s_5_4=[32.39 -149.14 -72.74]'*10^-3;

m=22.2;
Ixx = 0.9257;       Ixy = -0.0011;      Ixz = 0.0133;
Iyx = -0.0011;      Iyy = 0.6781;       Iyz = 0.0018;
Izx = 0.0133;       Izy = 0.0018;       Izz = 0.5108;
BodyNo=1;
Body(BodyNo).m=m; Body(BodyNo).J=[Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz];
geo.s_4_1=[19.84 -0.86 -338.43]'*10^-3;
geo.s_2_1=[24.84 -279.14 86.57]'*10^-3;
geo.s_3_1=[24.84 279.14 86.57]'*10^-3;

m=0.9;
Ixx = 0.03082;      Ixy = 9.766e-006;   Ixz = -0.0104;
Iyx = 9.766e-006;   Iyy = 0.03645;      Iyz = -6.16e-005;
Izx = -0.0104;      Izy = -6.16e-005;   Izz = 0.005924;
BodyNo=2;
Body(BodyNo).m=m; Body(BodyNo).J=[Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz];
geo.s_1_2=[-50.06 24.76 308.45]'*10^-3;
geo.s_w_2=[139.58 0 -246.28]'*10^-3;

m=0.9;
Ixx = 0.03082;      Ixy = 9.766e-006;   Ixz = -0.0104;
Iyx = 9.766e-006;   Iyy = 0.03645;      Iyz = -6.16e-005;
Izx = -0.0104;      Izy = -6.16e-005;   Izz = 0.005924;
BodyNo=3;
Body(BodyNo).m=m; Body(BodyNo).J=[Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz];
geo.s_1_3=[-50.06 -24.76 308.45]'*10^-3;
geo.s_w_3=[139.58 0 -246.28]'*10^-3;

m=6.4;
Ixx = 0.09728;      Ixy = -0.0002152;   Ixz = 0.0007405;
Iyx = -0.0002152;   Iyy = 0.08774;      Iyz = -0.003239;
Izx = 0.0007405;    Izy = -0.003239;    Izz = 0.02364;
BodyNo=5;
Body(BodyNo).m=m; Body(BodyNo).J=[Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz];
geo.s_4_5=[-1.37 32.70 143]'*10^-3;
geo.s_7_5=[-1.37 32.70 -167]'*10^-3;

m=6.4;
Ixx = 0.09728;      Ixy = -0.0002152;   Ixz = -0.0007405;
Iyx = -0.0002152;   Iyy = 0.08774;      Iyz = 0.003239;
Izx = -0.0007405;   Izy = 0.003239;     Izz = 0.02364;
BodyNo=6;
Body(BodyNo).m=m; Body(BodyNo).J=[Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz];
geo.s_4_6=[1.37 -32.70 143]'*10^-3;
```

```
geo.s_8_6=[1.37 -32.70 -167]'*10^-3;

m=5.3;
Ixx = 0.07839;      Ixy = 5.029e-005;   Ixz = -5.412e-005;
Iyx = 5.029e-005;   Iyy = 0.07447;      Iyz = -0.01043;
Izx = -5.412e-005;  Izy = -0.01043;     Izz = 0.01354;
BodyNo=7;
Body(BodyNo).m=m; Body(BodyNo).J=[Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz];
geo.s_5_7=[-2.91 -16.28 261.54]'*10^-3;
geo.s_9_7=[-2.92 -26.18 -109.00]'*10^-3;

m=5.3;
Ixx = 0.07839;      Ixy = 5.029e-005;   Ixz = 5.412e-005;
Iyx = 5.029e-005;   Iyy = 0.07447;      Iyz = 0.01043;
Izx = 5.412e-005;   Izy = 0.01043;      Izz = 0.01354;
BodyNo=8;
Body(BodyNo).m=m; Body(BodyNo).J=[Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz];
geo.s_6_8=[2.91 16.28 261.54]'*10^-3;
geo.s_10_8=[2.91 26.18 -109]'*10^-3;

m=4.2;
Ixx = 0.01215;      Ixy = 5.681e-006;   Ixz = -0.006042;
Iyx = 5.681e-006;   Iyy = 0.02128;      Iyz = -7.158e-007;
Izx = -0.006042;    Izy = -7.158e-007;  Izz = 0.01906;
BodyNo=9;
Body(BodyNo).m=m; Body(BodyNo).J=[Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz];
geo.s_7_9=[25.97 0 35.14]'*10^-3;
geo.s_t_9=[144.18+16 0 -65.41]'*10^-3;
geo.s_h_9=[-51.46+16 0 -87.27]'*10^-3;

m=4.2;
Ixx = 0.01215;      Ixy = 5.681e-006;   Ixz = -0.006042;
Iyx = 5.681e-006;   Iyy = 0.02128;      Iyz = -7.158e-007;
Izx = -0.006042;    Izy = -7.158e-007;  Izz = 0.01906;
BodyNo=10;
Body(BodyNo).m=m; Body(BodyNo).J=[Ixx Ixy Ixz; Iyx Iyy Iyz; Izx Izy Izz];
geo.s_8_10=[25.97 0 35.14]'*10^-3;
geo.s_t_10=[144.18+16 0 -65.41]'*10^-3;
geo.s_h_10=[-51.46+16 0 -87.27]'*10^-3;

geo.foot_width=120*10^-3;

Data.TotalMass=0;
for i=1:10
    Data.TotalMass=Data.TotalMass+Body(i).m;
end

clear m Ixx Iyy Izz Ixy Iyx Ixz Izx Iyz Izy BodyNo
```

```
function [Mn Body]=MassMatrix(Body)
% Assembly of the Mass matrix of AAU-BOT

% n is the number of segments
n=length(Body);

% segment 1; Torso + head
Body(1).m =   35.5;%kg
Mass(:,:,1)=[Body(1).m 0 0; 0 Body(1).m 0; 0 0 Body(1).m];
Ixx(1)= 1.9624   ;% kgm^2
Ixy(1)= 0        ;% kgm^2
Ixz(1)= 0        ;% kgm^2
Iyy(1)= 1.7710   ;% kgm^2
Iyz(1)= 0.0001   ;% kgm^2
Izz(1)= 0.3670   ;% kgm^2

% segment 2; Left arm
Body(2).m =   5.88;%kg
Mass(:,:,2)=[Body(2).m 0 0; 0 Body(2).m 0; 0 0 Body(2).m];
Ixx(2)= 0.1307  ;% kgm^2
Ixy(2)= 0       ;% kgm^2
Ixz(2)=-0.052   ;% kgm^2
Iyy(2)= 0.1737  ;% kgm^2
Iyz(2)= 0       ;% kgm^2
Izz(2)= 0.0519  ;% kgm^2

% segment 3; Right arm
Body(3).m =   5.88;%kg
Mass(:,:,3)=[Body(3).m 0 0; 0 Body(3).m 0; 0 0 Body(3).m];
Ixx(3)= 0.1513  ;% kgm^2
Ixy(3)= -0.0072 ;% kgm^2
Ixz(3)=-0.0269  ;% kgm^2
Iyy(3)= 0.1707  ;% kgm^2
Iyz(3)= 0.0176  ;% kgm^2
Izz(3)= 0.0344  ;% kgm^2

% segment 4; Pelvis
Body(4).m =   7.9;%kg
Mass(:,:,4)=[Body(4).m 0 0; 0 Body(4).m 0; 0 0 Body(4).m];
Ixx(4)= 0.0908 ;% kgm^2
Ixy(4)= 0      ;% kgm^2
Ixz(4)= 0      ;% kgm^2
Iyy(4)= 0.0317 ;% kgm^2
Iyz(4)= 0      ;% kgm^2
Izz(4)= 0.0963 ;% kgm^2

% segment 5; Right thigh
Body(5).m = 6.87 ;% kg
Mass(:,:,5)=[Body(5).m 0 0; 0 Body(5).m 0; 0 0 Body(5).m];
Ixx(5)= 0.0873 ;% kgm^2
Ixy(5)= 0      ;% kgm^2
Ixz(5)= 0      ;% kgm^2
Iyy(5)= 0.0857 ;% kgm^2
Iyz(5)= 0      ;% kgm^2
Izz(5)= 0.0212 ;% kgm^2
```

```
% segment 6; Left thigh
Body(6).m = 6.87 ;%kg
Mass(:,:,6)=[Body(6).m 0 0; 0 Body(6).m 0; 0 0 Body(6).m];
Ixx(6)= 0.0873 ;% kgm^2
Ixy(6)= 0      ;% kgm^2
Ixz(6)= 0      ;% kgm^2
Iyy(6)= 0.0857 ;% kgm^2
Iyz(6)= 0      ;% kgm^2
Izz(6)= 0.0212 ;% kgm^2

% segment 7; Right shin
Body(7).m =   3.62;%kg
Mass(:,:,7)=[Body(7).m 0 0 ; 0 Body(7).m 0; 0 0 Body(7).m];
Ixx(7)= 0.0528 ;% kgm^2
Ixy(7)= 0.0001 ;% kgm^2
Ixz(7)= 0      ;% kgm^2
Iyy(7)= 0.0516 ;% kgm^2
Iyz(7)= 0      ;% kgm^2
Izz(7)= 0.0054 ;% kgm^2

% segment 8; Left shin
Body(8).m =   3.62;%kg
Mass(:,:,8)=[Body(8).m 0 0; 0 Body(8).m 0; 0 0 Body(8).m];
Ixx(8)= 0.0528 ;% kgm^2
Ixy(8)= 0.0001 ;% kgm^2
Ixz(8)= 0      ;% kgm^2
Iyy(8)= 0.0516 ;% kgm^2
Iyz(8)= 0      ;% kgm^2
Izz(8)= 0.0054 ;% kgm^2

% segment 9; Right foot
Body(9).m =   1.13;%kg
Mass(:,:,9)=[Body(9).m 0 0; 0 Body(9).m 0; 0 0 Body(9).m];
Ixx(9)= 0.001 ;% kgm^2
Ixy(9)= 0     ;% kgm^2
Ixz(9)=-0.001 ;% kgm^2
Iyy(9)= 0.007 ;% kgm^2
Iyz(9)= 0     ;% kgm^2
Izz(9)= 0.006 ;% kgm^2

% segment 10; Left foot
Body(10).m =  1.1;%kg
Mass(:,:,10)=[Body(10).m 0 0; 0 Body(10).m 0; 0 0 Body(10).m];
Ixx(10)= 0.0012 ;% kgm^2
Ixy(10)= 0      ;% kgm^2
Ixz(10)=-0.0009 ;% kgm^2
Iyy(10)= 0.0066 ;% kgm^2
Iyz(10)= 0      ;% kgm^2
Izz(10)= 0.0065 ;% kgm^2

% Inerta matrix
Mn=zeros(6*n);
for i=1:n
    J(:,:,i)=[Ixx(i),Ixy(i),Ixz(i);Ixy(i),Iyy(i),Iyz(i);Ixz(i),Iyz(i),Izz(i)];
    Body(i).J=J(:,:,i);
    M(:,:,i)=zeros(6);
```

```
    M(1:3,1:3,i)=Mass(:,:,i);
    M(4:6,4:6,i)=J(:,:,i);
    Mn(i*6-5:i*6,i*6-5:i*6)=M(:,:,i);
    BodyMass(i)=Body(i).m;
end

UnitMass(1,1:n)=1;
TotalMass=UnitMass*BodyMass';
```

```
% Calculates the mean values of normwalk input
fprintf('\nReading input, taking averages')

% Loading input
load('Loadcases\walk1.mat')
walk1 = QTMmeasurements;
load('Loadcases\walk2.mat')
walk2 = QTMmeasurements;
load('Loadcases\walk3.mat')
walk3 = QTMmeasurements;
load('Loadcases\walk4.mat')
walk4 = QTMmeasurements;
load('Loadcases\walk5.mat')
walk5 = QTMmeasurements;

% The total lenght of the data set
Frames = 283;
% a correction parameter for the startingpoint
cor = 33;

for i = 1:5
    if i == 1
        name = 'walk1';
        No = 1;
        start = 0+cor;
        nFrames = Frames + start;
    end
    if i == 2
        name = 'walk2';
        No = 2;
        start = 9+cor;
        nFrames = Frames + start;
    end
    if i == 3
        name = 'walk3';
        No = 3;
        start = 3+cor;
        nFrames = Frames + start;
    end
    if i == 4
        name = 'walk4';
        No = 4;
        start = 12+cor;
        nFrames = Frames + start;
    end
    if i == 5
        name = 'walk5';
        No = 5;
        start = 15+cor;
        nFrames = Frames + start;
    end



    for FrameNo = start:nFrames
    %  Import data
```

```
    %  (A,B,C)
    %  A = Bodypart, B = x,y or z coordinate, C = Datapoint number

    % Body coordinates imported from the QTMmeasurements struct is collected
    % in an input struct and transposed to two dimensions. The start of the
    % datapoints is translated so the trajectories are on top of each other

        input(No).neck(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.Data↙
(1,1:3,FrameNo)']);

        input(No).r_shoulder(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.↙
Labeled.Data(2,1:3,FrameNo)']);

        input(No).l_shoulder(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.↙
Labeled.Data(3,1:3,FrameNo)']);

        input(No).r_hip_socket(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.↙
Labeled.Data(4,1:3,FrameNo)']);

        input(No).l_hip_socket(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.↙
Labeled.Data(5,1:3,FrameNo)']);

        input(No).waist(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.Data↙
(6,1:3,FrameNo)']);

        input(No).r_hip(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.Data↙
(7,1:3,FrameNo)']);

        input(No).l_hip(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.Data↙
(8,1:3,FrameNo)']);

        input(No).r_wrist(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(9,1:3,FrameNo)']);

        input(No).l_wrist(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(10,1:3,FrameNo)']);

        input(No).r_thigh_u(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(11,1:3,FrameNo)']);

        input(No).r_thigh_f(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(12,1:3,FrameNo)']);

        input(No).r_thigh_r(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(13,1:3,FrameNo)']);

        input(No).r_thigh_d(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(14,1:3,FrameNo)']);

        input(No).l_thigh_u(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(15,1:3,FrameNo)']);

        input(No).l_thigh_f(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(16,1:3,FrameNo)']);

        input(No).l_thigh_r(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
```

```
Data(17,1:3,FrameNo)']);

        input(No).l_thigh_d(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(18,1:3,FrameNo)']);

        input(No).r_knee(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(19,1:3,FrameNo)']);

        input(No).l_knee(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(20,1:3,FrameNo)']);

        input(No).r_shin_u(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(21,1:3,FrameNo)']);

        input(No).r_shin_f(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(22,1:3,FrameNo)']);

        input(No).r_shin_r(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(23,1:3,FrameNo)']);

        input(No).r_shin_d(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(24,1:3,FrameNo)']);

        input(No).l_shin_u(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(25,1:3,FrameNo)']);

        input(No).l_shin_f(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(26,1:3,FrameNo)']);

        input(No).l_shin_r(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(27,1:3,FrameNo)']);

        input(No).l_shin_d(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(28,1:3,FrameNo)']);

        input(No).r_ankle(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(29,1:3,FrameNo)']);

        input(No).r_heel(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(30,1:3,FrameNo)']);

        input(No).r_toe(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.Data↙
(31,1:3,FrameNo)']);

        input(No).l_ankle(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(32,1:3,FrameNo)']);

        input(No).l_heel(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.↙
Data(33,1:3,FrameNo)']);

        input(No).l_toe(1:3,FrameNo-(start-1)) = eval([name '.Trajectories.Labeled.Data↙
(34,1:3,FrameNo)']);



%frames = 1:nFrames;
```

```
time = Frames/QTMmeasurements.FrameRate;    %[sec]

% Data from force platform 1 and 2

           input(No).F1xInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(1,↙
FrameNo)']);
           input(No).F1yInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(2,↙
FrameNo)']);
           input(No).F1zInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(3,↙
FrameNo)']);
           input(No).M1xInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(4,↙
FrameNo)']);
           input(No).M1yInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(5,↙
FrameNo)']);
           input(No).M1zInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(6,↙
FrameNo)']);
           input(No).F2xInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(7,↙
FrameNo)']);
           input(No).F2yInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(8,↙
FrameNo)']);
           input(No).F2zInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(9,↙
FrameNo)']);
           input(No).M2xInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(10,↙
FrameNo)']);
           input(No).M2yInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(11,↙
FrameNo)']);
           input(No).M2zInput(1,FrameNo-(start-1)) = eval([name '.Analog.Data(12,↙
FrameNo)']);

    end
%%%%%%%%%%% Input is corrected and substituted %%%%%%%%%%

    for FrameNo = 1:Frames+1
        Correction1(:,FrameNo) =  [0,0,-30];
        input(No).r_shoulder(:,FrameNo)  = input(No).r_shoulder(:,FrameNo) +↙
Correction1(:,FrameNo);

        Correction2(:,FrameNo) =  [0,0,-30];
        input(No).l_shoulder(:,FrameNo)  = input(No).l_shoulder(:,FrameNo) +↙
Correction2(:,FrameNo);

        Correction3(:,FrameNo) =  [-50,0,0];
        input(No).r_hip_socket(:,FrameNo) = input(No).r_hip_socket(:,FrameNo) +↙
Correction3(:,FrameNo);

        Correction4(:,FrameNo) =  [-42,0,0];
        input(No).l_hip_socket(:,FrameNo) = input(No).l_hip_socket(:,FrameNo) +↙
Correction4(:,FrameNo);

        Correction5(:,FrameNo) =  [10,60,-5];
        input(No).r_hip(:,FrameNo) = input(No).r_hip(:,FrameNo) + Correction5(:,↙
FrameNo);

        Correction6(:,FrameNo) =  [-10,-60,5];
        input(No).l_hip(:,FrameNo) = input(No).l_hip(:,FrameNo) + Correction6(:,↙
FrameNo);
```

```
        Correction7(:,FrameNo) =  [0,55,0];
        input(No).r_knee(:,FrameNo) = input(No).r_knee(:,FrameNo) + Correction7(:,↙
FrameNo);

        Correction8(:,FrameNo) =  [0,-55,0];
        input(No).l_knee(:,FrameNo) = input(No).l_knee(:,FrameNo) + Correction8(:,↙
FrameNo);

        Correction9(:,FrameNo) =  [0,46.5,0];
        input(No).r_ankle(:,FrameNo) = input(No).r_ankle(:,FrameNo) + Correction9(:,↙
FrameNo);

        Correction10(:,FrameNo) = [0,-1,0];
        input(No).r_toe(:,FrameNo) = input(No).r_toe(:,FrameNo) + Correction10(:,↙
FrameNo);

        Correction11(:,FrameNo) = [0,0,18];
        input(No).r_heel(:,FrameNo) = input(No).r_heel(:,FrameNo) + Correction11(:,↙
FrameNo);

        Correction12(:,FrameNo) =  [0,-39,0];
        input(No).l_ankle(:,FrameNo) = input(No).l_ankle(:,FrameNo) + Correction12(:,↙
FrameNo);

        Correction13(:,FrameNo) = [0,19,0];
        input(No).l_toe(:,FrameNo) = input(No).l_toe(:,FrameNo) + Correction13(:,↙
FrameNo);

        Correction14(:,FrameNo) = [0,0,23];
        input(No).l_heel(:,FrameNo) = input(No).l_heel(:,FrameNo) + Correction14(:,↙
FrameNo);

        Correction15(:,FrameNo) = [0,-11,0];
        input(No).r_thigh_f(:,FrameNo) = input(No).r_thigh_f(:,FrameNo) + Correction15↙
(:,FrameNo);

        Correction16(:,FrameNo) = [0,28,0];
        input(No).l_thigh_f(:,FrameNo) = input(No).l_thigh_f(:,FrameNo) + Correction16↙
(:,FrameNo);

        Correction17(:,FrameNo) = [0,10,0];
        input(No).r_thigh_u(:,FrameNo) = input(No).r_thigh_u(:,FrameNo) + Correction17↙
(:,FrameNo);

        Correction18(:,FrameNo) = [0,-10,0];
        input(No).l_thigh_u(:,FrameNo) = input(No).l_thigh_u(:,FrameNo) + Correction18↙
(:,FrameNo);
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Output %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The mean value of each bodypart and forceplate are calculated

for FrameNo = 1:Frames+1
```

```
    neck(:,FrameNo) = (input(1).neck(:,FrameNo)+input(2).neck(:,FrameNo)+input(3).neck↙
(:,FrameNo)+input(4).neck(:,FrameNo)+input(5).neck(:,FrameNo))/5;

    r_shoulder(:,FrameNo) = (input(1).r_shoulder(:,FrameNo)+input(2).r_shoulder(:,↙
FrameNo)+input(3).r_shoulder(:,FrameNo)+input(4).r_shoulder(:,FrameNo)+input(5).↙
r_shoulder(:,FrameNo))/5;

    l_shoulder(:,FrameNo) = (input(1).l_shoulder(:,FrameNo)+input(2).l_shoulder(:,↙
FrameNo)+input(3).l_shoulder(:,FrameNo)+input(4).l_shoulder(:,FrameNo)+input(5).↙
l_shoulder(:,FrameNo))/5;

    r_hip_socket(:,FrameNo) = (input(1).r_hip_socket(:,FrameNo)+input(2).r_hip_socket↙
(:,FrameNo)+input(3).r_hip_socket(:,FrameNo)+input(4).r_hip_socket(:,FrameNo)+input(5).↙
r_hip_socket(:,FrameNo))/5;

    l_hip_socket(:,FrameNo) = (input(1).l_hip_socket(:,FrameNo)+input(2).l_hip_socket↙
(:,FrameNo)+input(3).l_hip_socket(:,FrameNo)+input(4).l_hip_socket(:,FrameNo)+input(5).↙
l_hip_socket(:,FrameNo))/5;

    waist(:,FrameNo) = (input(1).waist(:,FrameNo)+input(2).waist(:,FrameNo)+input(3).↙
waist(:,FrameNo)+input(4).waist(:,FrameNo)+input(5).waist(:,FrameNo))/5;

    r_hip(:,FrameNo) = (input(1).r_hip(:,FrameNo)+input(2).r_hip(:,FrameNo)+input(3).↙
r_hip(:,FrameNo)+input(4).r_hip(:,FrameNo)+input(5).r_hip(:,FrameNo))/5;

    l_hip(:,FrameNo) = (input(1).l_hip(:,FrameNo)+input(2).l_hip(:,FrameNo)+input(3).↙
l_hip(:,FrameNo)+input(4).l_hip(:,FrameNo)+input(5).l_hip(:,FrameNo))/5;

    r_wrist(:,FrameNo) = (input(1).r_wrist(:,FrameNo)+input(2).r_wrist(:,FrameNo)+input↙
(3).r_wrist(:,FrameNo)+input(4).r_wrist(:,FrameNo)+input(5).r_wrist(:,FrameNo))/5;

    l_wrist(:,FrameNo) = (input(1).l_wrist(:,FrameNo)+input(2).l_wrist(:,FrameNo)+input↙
(3).l_wrist(:,FrameNo)+input(4).l_wrist(:,FrameNo)+input(5).l_wrist(:,FrameNo))/5;

    r_thigh_u(:,FrameNo) = (input(1).r_thigh_u(:,FrameNo)+input(2).r_thigh_u(:,FrameNo)↙
+input(3).r_thigh_u(:,FrameNo)+input(4).r_thigh_u(:,FrameNo)+input(5).r_thigh_u(:,↙
FrameNo))/5;

    r_thigh_f(:,FrameNo) = (input(1).r_thigh_f(:,FrameNo)+input(2).r_thigh_f(:,FrameNo)↙
+input(3).r_thigh_f(:,FrameNo)+input(4).r_thigh_f(:,FrameNo)+input(5).r_thigh_f(:,↙
FrameNo))/5;

    r_thigh_r(:,FrameNo) = (input(1).r_thigh_r(:,FrameNo)+input(2).r_thigh_r(:,FrameNo)↙
+input(3).r_thigh_r(:,FrameNo)+input(4).r_thigh_r(:,FrameNo)+input(5).r_thigh_r(:,↙
FrameNo))/5;

    r_thigh_d(:,FrameNo) = (input(1).r_thigh_d(:,FrameNo)+input(2).r_thigh_d(:,FrameNo)↙
+input(3).r_thigh_d(:,FrameNo)+input(4).r_thigh_d(:,FrameNo)+input(5).r_thigh_d(:,↙
FrameNo))/5;

    l_thigh_u(:,FrameNo) = (input(1).l_thigh_u(:,FrameNo)+input(2).l_thigh_u(:,FrameNo)↙
+input(3).l_thigh_u(:,FrameNo)+input(4).l_thigh_u(:,FrameNo)+input(5).l_thigh_u(:,↙
FrameNo))/5;

    l_thigh_f(:,FrameNo) = (input(1).l_thigh_f(:,FrameNo)+input(2).l_thigh_f(:,FrameNo)↙
```

```
+input(3).l_thigh_f(:,FrameNo)+input(4).l_thigh_f(:,FrameNo)+input(5).l_thigh_f(:,↙
FrameNo))/5;

    l_thigh_r(:,FrameNo) = (input(1).l_thigh_r(:,FrameNo)+input(2).l_thigh_r(:,FrameNo)↙
+input(3).l_thigh_r(:,FrameNo)+input(4).l_thigh_r(:,FrameNo)+input(5).l_thigh_r(:,↙
FrameNo))/5;

    l_thigh_d(:,FrameNo) = (input(1).l_thigh_d(:,FrameNo)+input(2).l_thigh_d(:,FrameNo)↙
+input(3).l_thigh_d(:,FrameNo)+input(4).l_thigh_d(:,FrameNo)+input(5).l_thigh_d(:,↙
FrameNo))/5;

    r_knee(:,FrameNo) = (input(1).r_knee(:,FrameNo)+input(2).r_knee(:,FrameNo)+input↙
(3).r_knee(:,FrameNo)+input(4).r_knee(:,FrameNo)+input(5).r_knee(:,FrameNo))/5;

    l_knee(:,FrameNo) = (input(1).l_knee(:,FrameNo)+input(2).l_knee(:,FrameNo)+input↙
(3).l_knee(:,FrameNo)+input(4).l_knee(:,FrameNo)+input(5).l_knee(:,FrameNo))/5;

    r_shin_u(:,FrameNo) = (input(1).r_shin_u(:,FrameNo)+input(2).r_shin_u(:,FrameNo)↙
+input(3).r_shin_u(:,FrameNo)+input(4).r_shin_u(:,FrameNo)+input(5).r_shin_u(:,↙
FrameNo))/5;

    r_shin_f(:,FrameNo) = (input(1).r_shin_f(:,FrameNo)+input(2).r_shin_f(:,FrameNo)↙
+input(3).r_shin_f(:,FrameNo)+input(4).r_shin_f(:,FrameNo)+input(5).r_shin_f(:,↙
FrameNo))/5;

    r_shin_r(:,FrameNo) = (input(1).r_shin_r(:,FrameNo)+input(2).r_shin_r(:,FrameNo)↙
+input(3).r_shin_r(:,FrameNo)+input(4).r_shin_r(:,FrameNo)+input(5).r_shin_r(:,↙
FrameNo))/5;

    r_shin_d(:,FrameNo) = (input(1).r_shin_d(:,FrameNo)+input(2).r_shin_d(:,FrameNo)↙
+input(3).r_shin_d(:,FrameNo)+input(4).r_shin_d(:,FrameNo)+input(5).r_shin_d(:,↙
FrameNo))/5;

    l_shin_u(:,FrameNo) = (input(1).l_shin_u(:,FrameNo)+input(2).l_shin_u(:,FrameNo)↙
+input(3).l_shin_u(:,FrameNo)+input(4).l_shin_u(:,FrameNo)+input(5).l_shin_u(:,↙
FrameNo))/5;

    l_shin_f(:,FrameNo) = (input(1).l_shin_f(:,FrameNo)+input(2).l_shin_f(:,FrameNo)↙
+input(3).l_shin_f(:,FrameNo)+input(4).l_shin_f(:,FrameNo)+input(5).l_shin_f(:,↙
FrameNo))/5;

    l_shin_r(:,FrameNo) = (input(1).l_shin_r(:,FrameNo)+input(2).l_shin_r(:,FrameNo)↙
+input(3).l_shin_r(:,FrameNo)+input(4).l_shin_r(:,FrameNo)+input(5).l_shin_r(:,↙
FrameNo))/5;

    l_shin_d(:,FrameNo) = (input(1).l_shin_d(:,FrameNo)+input(2).l_shin_d(:,FrameNo)↙
+input(3).l_shin_d(:,FrameNo)+input(4).l_shin_d(:,FrameNo)+input(5).l_shin_d(:,↙
FrameNo))/5;

    r_ankle(:,FrameNo) = (input(1).r_ankle(:,FrameNo)+input(2).r_ankle(:,FrameNo)+input↙
(3).r_ankle(:,FrameNo)+input(4).r_ankle(:,FrameNo)+input(5).r_ankle(:,FrameNo))/5;

    r_heel(:,FrameNo) = (input(1).r_heel(:,FrameNo)+input(2).r_heel(:,FrameNo)+input↙
(3).r_heel(:,FrameNo)+input(4).r_heel(:,FrameNo)+input(5).r_heel(:,FrameNo))/5;

    r_toe(:,FrameNo) = (input(1).r_toe(:,FrameNo)+input(2).r_toe(:,FrameNo)+input(3).↙
```

```
r_toe(:,FrameNo)+input(4).r_toe(:,FrameNo)+input(5).r_toe(:,FrameNo))/5;

    l_ankle(:,FrameNo) = (input(1).l_ankle(:,FrameNo)+input(2).l_ankle(:,FrameNo)+input↙
(3).l_ankle(:,FrameNo)+input(4).l_ankle(:,FrameNo)+input(5).l_ankle(:,FrameNo))/5;

    l_heel(:,FrameNo) = (input(1).l_heel(:,FrameNo)+input(2).l_heel(:,FrameNo)+input↙
(3).l_heel(:,FrameNo)+input(4).l_heel(:,FrameNo)+input(5).l_heel(:,FrameNo))/5;

    l_toe(:,FrameNo) = (input(1).l_toe(:,FrameNo)+input(2).l_toe(:,FrameNo)+input(3).↙
l_toe(:,FrameNo)+input(4).l_toe(:,FrameNo)+input(5).l_toe(:,FrameNo))/5;

%%%%%%%%%%%%%%%%%%%%%%%% Forceplates %%%%%%%%%%%%%%%%%%%%%%%%

    F1xInput(:,FrameNo) = (input(1).F1xInput(:,FrameNo)+input(2).F1xInput(:,FrameNo)↙
+input(3).F1xInput(:,FrameNo)+input(4).F1xInput(:,FrameNo)+input(5).F1xInput(:,↙
FrameNo))/5;
    F1yInput(:,FrameNo) = (input(1).F1yInput(:,FrameNo)+input(2).F1yInput(:,FrameNo)↙
+input(3).F1yInput(:,FrameNo)+input(4).F1yInput(:,FrameNo)+input(5).F1yInput(:,↙
FrameNo))/5;
    F1zInput(:,FrameNo) = (input(1).F1zInput(:,FrameNo)+input(2).F1zInput(:,FrameNo)↙
+input(3).F1zInput(:,FrameNo)+input(4).F1zInput(:,FrameNo)+input(5).F1zInput(:,↙
FrameNo))/5;
    M1xInput(:,FrameNo) = (input(1).M1xInput(:,FrameNo)+input(2).M1xInput(:,FrameNo)↙
+input(3).M1xInput(:,FrameNo)+input(4).M1xInput(:,FrameNo)+input(5).M1xInput(:,↙
FrameNo))/5;
    M1yInput(:,FrameNo) = (input(1).M1yInput(:,FrameNo)+input(2).M1yInput(:,FrameNo)↙
+input(3).M1yInput(:,FrameNo)+input(4).M1yInput(:,FrameNo)+input(5).M1yInput(:,↙
FrameNo))/5;
    M1zInput(:,FrameNo) = (input(1).M1zInput(:,FrameNo)+input(2).M1zInput(:,FrameNo)↙
+input(3).M1zInput(:,FrameNo)+input(4).M1zInput(:,FrameNo)+input(5).M1zInput(:,↙
FrameNo))/5;
    F2xInput(:,FrameNo) = (input(1).F2xInput(:,FrameNo)+input(2).F2xInput(:,FrameNo)↙
+input(3).F2xInput(:,FrameNo)+input(4).F2xInput(:,FrameNo)+input(5).F2xInput(:,↙
FrameNo))/5;
    F2yInput(:,FrameNo) = (input(1).F2yInput(:,FrameNo)+input(2).F2yInput(:,FrameNo)↙
+input(3).F2yInput(:,FrameNo)+input(4).F2yInput(:,FrameNo)+input(5).F2yInput(:,↙
FrameNo))/5;
    F2zInput(:,FrameNo) = (input(1).F2zInput(:,FrameNo)+input(2).F2zInput(:,FrameNo)↙
+input(3).F2zInput(:,FrameNo)+input(4).F2zInput(:,FrameNo)+input(5).F2zInput(:,↙
FrameNo))/5;
    M2xInput(:,FrameNo) = (input(1).M2xInput(:,FrameNo)+input(2).M2xInput(:,FrameNo)↙
+input(3).M2xInput(:,FrameNo)+input(4).M2xInput(:,FrameNo)+input(5).M2xInput(:,↙
FrameNo))/5;
    M2yInput(:,FrameNo) = (input(1).M2yInput(:,FrameNo)+input(2).M2yInput(:,FrameNo)↙
+input(3).M2yInput(:,FrameNo)+input(4).M2yInput(:,FrameNo)+input(5).M2yInput(:,↙
FrameNo))/5;
    M2zInput(:,FrameNo) = (input(1).M2zInput(:,FrameNo)+input(2).M2zInput(:,FrameNo)↙
+input(3).M2zInput(:,FrameNo)+input(4).M2zInput(:,FrameNo)+input(5).M2zInput(:,↙
FrameNo))/5;
end

clear Correction1 Correction2 Correction3 Correction4 Correction5 Correction6↙
Correction7 ...
      Correction8 Correction9 Correction10 Correction11 Correction12 Correction13↙
Correction14 ...
      Correction15 Correction16 Correction17 Correction18
```

```
clear QTMmeasurements

nFrames = Frames+1;
```

```
%Loading workspace with datapoints. The name and location of the workspace
%exported from QTM should be substituted in the path.
fprintf('\nReading input')
load(['Loadcases\' LC '.mat'])

%  Import data
%  (A,B,C)
%  A = Bodypart, B = x,y,or z coordinate, C = Datapoint number

%  1 = neck
%  2 = r_shoulder
%  3 = l_shoulder
%  4 = r_hip_socket
%  5 = l_hip_socket
%  6 = waist
%  7 = r_hip
%  8 = l_hip
%  9 = r_wrist
% 10 = l_wrist
% 11 = r_thigh_u
% 12 = r_thigh_f
% 13 = r_thigh_r
% 14 = r_thigh_d
% 15 = l_thigh_u
% 16 = l_thigh_f
% 17 = l_thigh_r
% 18 = l_thigh_d
% 19 = r_knee
% 20 = l_knee
% 21 = r_shin_u
% 22 = r_shin_f
% 23 = r_shin_r
% 24 = r_shin_d
% 25 = l_shin_u
% 26 = l_shin_f
% 27 = l_shin_r
% 28 = l_shin_d
% 29 = r_ankle
% 30 = r_heel
% 31 = r_toe
% 32 = l_ankle
% 33 = l_heel
% 34 = l_toe


% Determining the number of frames (datapoints)
[i,j,nFrames] = size(QTMmeasurements.Trajectories.Labeled.Data);

% Body coordinates collected from the QTMmeasurements struct
neck(:,:) = QTMmeasurements.Trajectories.Labeled.Data(1,:,:);

r_shoulder(:,:) = QTMmeasurements.Trajectories.Labeled.Data(2,:,:);

l_shoulder(:,:) = QTMmeasurements.Trajectories.Labeled.Data(3,:,:);

r_hip_socket(:,:) = QTMmeasurements.Trajectories.Labeled.Data(4,:,:);
```

```
l_hip_socket(:,:) = QTMmeasurements.Trajectories.Labeled.Data(5,:,:);
waist(:,:) = QTMmeasurements.Trajectories.Labeled.Data(6,:,:);
r_hip(:,:) = QTMmeasurements.Trajectories.Labeled.Data(7,:,:);
l_hip(:,:) = QTMmeasurements.Trajectories.Labeled.Data(8,:,:);
r_wrist(:,:) = QTMmeasurements.Trajectories.Labeled.Data(9,:,:);
l_wrist(:,:) = QTMmeasurements.Trajectories.Labeled.Data(10,:,:);
r_thigh_u(:,:) = QTMmeasurements.Trajectories.Labeled.Data(11,:,:);
r_thigh_f(:,:) = QTMmeasurements.Trajectories.Labeled.Data(12,:,:);
r_thigh_r(:,:) = QTMmeasurements.Trajectories.Labeled.Data(13,:,:);
r_thigh_d(:,:) = QTMmeasurements.Trajectories.Labeled.Data(14,:,:);
l_thigh_u(:,:) = QTMmeasurements.Trajectories.Labeled.Data(15,:,:);
l_thigh_f(:,:) = QTMmeasurements.Trajectories.Labeled.Data(16,:,:);
l_thigh_r(:,:) = QTMmeasurements.Trajectories.Labeled.Data(17,:,:);
l_thigh_d(:,:) = QTMmeasurements.Trajectories.Labeled.Data(18,:,:);
r_knee(:,:) = QTMmeasurements.Trajectories.Labeled.Data(19,:,:);
l_knee(:,:) = QTMmeasurements.Trajectories.Labeled.Data(20,:,:);
r_shin_u(:,:) = QTMmeasurements.Trajectories.Labeled.Data(21,:,:);
r_shin_f(:,:) = QTMmeasurements.Trajectories.Labeled.Data(22,:,:);
r_shin_r(:,:) = QTMmeasurements.Trajectories.Labeled.Data(23,:,:);
r_shin_d(:,:) = QTMmeasurements.Trajectories.Labeled.Data(24,:,:);
l_shin_u(:,:) = QTMmeasurements.Trajectories.Labeled.Data(25,:,:);
l_shin_f(:,:) = QTMmeasurements.Trajectories.Labeled.Data(26,:,:);
l_shin_r(:,:) = QTMmeasurements.Trajectories.Labeled.Data(27,:,:);
l_shin_d(:,:) = QTMmeasurements.Trajectories.Labeled.Data(28,:,:);
r_ankle(:,:) = QTMmeasurements.Trajectories.Labeled.Data(29,:,:);
r_heel(:,:) = QTMmeasurements.Trajectories.Labeled.Data(30,:,:);
r_toe(:,:) = QTMmeasurements.Trajectories.Labeled.Data(31,:,:);
l_ankle(:,:) = QTMmeasurements.Trajectories.Labeled.Data(32,:,:);
```

```
l_heel(:,:) = QTMmeasurements.Trajectories.Labeled.Data(33,:,:);

l_toe(:,:) = QTMmeasurements.Trajectories.Labeled.Data(34,:,:);

frames = [0:nFrames-1];
time = frames/QTMmeasurements.FrameRate;    %[sec]

% Data from force platform 1 and 2

           F1xInput(:,:) = QTMmeasurements.Analog.Data(1,:,:);
           F1yInput(:,:) = QTMmeasurements.Analog.Data(2,:,:);
           F1zInput(:,:) = QTMmeasurements.Analog.Data(3,:,:);
           M1xInput(:,:) = QTMmeasurements.Analog.Data(4,:,:);
           M1yInput(:,:) = QTMmeasurements.Analog.Data(5,:,:);
           M1zInput(:,:) = QTMmeasurements.Analog.Data(6,:,:);
           F2xInput(:,:) = QTMmeasurements.Analog.Data(7,:,:);
           F2yInput(:,:) = QTMmeasurements.Analog.Data(8,:,:);
           F2zInput(:,:) = QTMmeasurements.Analog.Data(9,:,:);
           M2xInput(:,:) = QTMmeasurements.Analog.Data(10,:,:);
           M2yInput(:,:) = QTMmeasurements.Analog.Data(11,:,:);
           M2zInput(:,:) = QTMmeasurements.Analog.Data(12,:,:);

%%%%%%%%%%% Input is corrected and substituted %%%%%%%%%%
for FrameNo = 1:nFrames
    Correction1(:,FrameNo) =  [0,0,-30];
    r_shoulder(:,FrameNo)  = r_shoulder(:,FrameNo) + Correction1(:,FrameNo);

    Correction2(:,FrameNo) =  [0,0,-30];
    l_shoulder(:,FrameNo)  = l_shoulder(:,FrameNo) + Correction2(:,FrameNo);

    Correction3(:,FrameNo) =  [-50,0,0];
    r_hip_socket(:,FrameNo) = r_hip_socket(:,FrameNo) + Correction3(:,FrameNo);

    Correction4(:,FrameNo) =  [-42,0,0];
    l_hip_socket(:,FrameNo) = l_hip_socket(:,FrameNo) + Correction4(:,FrameNo);

    Correction5(:,FrameNo) =  [10,60,-5];
    r_hip(:,FrameNo) = r_hip(:,FrameNo) + Correction5(:,FrameNo);

    Correction6(:,FrameNo) =  [-10,-60,5];
    l_hip(:,FrameNo) = l_hip(:,FrameNo) + Correction6(:,FrameNo);

    Correction7(:,FrameNo) =  [0,55,0];
    r_knee(:,FrameNo) = r_knee(:,FrameNo) + Correction7(:,FrameNo);

    Correction8(:,FrameNo) =  [0,-55,0];
    l_knee(:,FrameNo) = l_knee(:,FrameNo) + Correction8(:,FrameNo);

    Correction9(:,FrameNo) =  [0,46.5,0];
    r_ankle(:,FrameNo) = r_ankle(:,FrameNo) + Correction9(:,FrameNo);

    Correction10(:,FrameNo) = [0,-1,0];
    r_toe(:,FrameNo) = r_toe(:,FrameNo) + Correction10(:,FrameNo);

    Correction11(:,FrameNo) = [0,0,18];
```

```
    r_heel(:,FrameNo) = r_heel(:,FrameNo) + Correction11(:,FrameNo);

    Correction12(:,FrameNo) =  [0,-39,0];
    l_ankle(:,FrameNo) = l_ankle(:,FrameNo) + Correction12(:,FrameNo);

    Correction13(:,FrameNo) = [0,19,0];
    l_toe(:,FrameNo) = l_toe(:,FrameNo) + Correction13(:,FrameNo);

    Correction14(:,FrameNo) = [0,0,23];
    l_heel(:,FrameNo) = l_heel(:,FrameNo) + Correction14(:,FrameNo);

    Correction15(:,FrameNo) = [0,-11,0];
    r_thigh_f(:,FrameNo) = r_thigh_f(:,FrameNo) + Correction15(:,FrameNo);

    Correction16(:,FrameNo) = [0,28,0];
    l_thigh_f(:,FrameNo) = l_thigh_f(:,FrameNo) + Correction16(:,FrameNo);

    Correction17(:,FrameNo) = [0,10,0];
    r_thigh_u(:,FrameNo) = r_thigh_u(:,FrameNo) + Correction17(:,FrameNo);

    Correction18(:,FrameNo) = [0,-10,0];
    l_thigh_u(:,FrameNo) = l_thigh_u(:,FrameNo) + Correction18(:,FrameNo);
end

clear Correction1 Correction2 Correction3 Correction4 Correction5 Correction6↙
Correction7 ...
      Correction8 Correction9 Correction10 Correction11 Correction12 Correction13↙
Correction14 ...
      Correction15 Correction16 Correction17 Correction18

clear QTMmeasurements

if strcmp(LC,'curb')
    RepairCurbClimb;
end
```

```
% Force platform 1, serial number 4009
% Origin: Xo=-1mm Yo=0,28mm Zo=42mm
% Sensitivities
SFx1 = 0.657; % uV/Vo/N
SFy1 = 0.657; % uV/Vo/N
SFz1 = 0.168; % uV/Vo/N
SMx1 = 1.645; % uV/Vo/N-0,15732
SMy1 = 1.639; % uV/Vo/N
SMz1 = 3.278; % uV/Vo/N
Gain1= 1000;  % HZ
Vo   = 10;    % V
micro= 0.000001; % volts/micro-volt
% Output amplifier force platform 1

% Voltage without amplifier (imput to amplifier); F1x, F1y, F1x, M1z M1y,
% M1z
F1x = F1xInput/(micro*SFx1*Vo*Gain1);
F1y = F1yInput/(micro*SFy1*Vo*Gain1);
F1z = F1zInput/(micro*SFz1*Vo*Gain1);
M1x = M1xInput/(micro*SMx1*Vo*Gain1);
M1y = M1yInput/(micro*SMy1*Vo*Gain1);
M1z = M1zInput/(micro*SMz1*Vo*Gain1);

for i=1:nFrames
    Force1(:,i)=[F1x(i) F1y(i) F1z(i)]';
    Moment1(:,i)=[M1x(i) M1y(i) M1z(i)]';
end

% Force platform 2, Serie 3577
% Origin: Xo=-1mm, Yo=0.12mm, Zo=39mm
% Sensitivities
SFx2 = 0.339; % uV/Vo/N
SFy2 = 0.337; % uV/Vo/N
SFz2 = 0.087; % uV/Vo/N
SMx2 = 0.650; % uV/Vo/N-0,15732
SMy2 = 0.649; % uV/Vo/N
SMz2 = 1.331; % uV/Vo/N
Gain2= 2000;  % HZ
Vo   = 10;    % V
micro= 0.000001; % volts/micro-volt
% Output amplifier force platform 2; F2x, F2y, F2z, Mx, My, Mz
% Voltage without amplifier (imput to amplifier)
F2x = F2xInput/(micro*SFx2*Vo*Gain2);
F2y = F2yInput/(micro*SFy2*Vo*Gain2);
F2z = F2zInput/(micro*SFz2*Vo*Gain2);
M2x = M2xInput/(micro*SMx2*Vo*Gain2);
M2y = M2yInput/(micro*SMy2*Vo*Gain2);
M2z = M2zInput/(micro*SMz2*Vo*Gain2);

% Correct Fz2:
if strcmp(LC,'mean_walk')
    F2z=F2z+abs(mean(F2z(1:floor(nFrames/3))));
elseif strcmp(LC,'curb')
    F2z=F2z-abs(mean(F2z(1:floor(nFrames/3))));
else
    F2z=F2z;
```

```
end

for i=1:nFrames
    Force2(:,i)=[F2x(i) F2y(i) F2z(i)]';
    Moment2(:,i)=[M2x(i) M2y(i) M2z(i)]';
end

% Store results
Data.ForceR=Force1;
Data.MomentR=Moment1;
Data.ForceL=Force2;
Data.MomentL=Moment2;

% Setup vectors determining whether there is ground contact under the feet
ContactR(1:nFrames)=0;
ContactL(1:nFrames)=0;

if strcmp(LC,'start_rightforce')
    DeltaF = 10.0; %N
else
    DeltaF = 6.0; %N
end

for FrameNo=1:nFrames
   if F1z(FrameNo) > DeltaF
       ContactR(FrameNo)=1;
   end
   if F2z(FrameNo) > DeltaF
       ContactL(FrameNo)=1;
   end
end


if strcmp(LC,'mean_walk') % Use only for straight walking load case
    % Determine length of double support phase
    for FrameNo=2:nFrames-1
        if (ContactL(FrameNo)==0)&& (ContactL(FrameNo+1)==1)
           begin_DSP=FrameNo;
        end

        if (ContactR(FrameNo)==0)&& (ContactR(FrameNo-1)==1)
            end_DSP=FrameNo;
        end
    end
    LengthDSP=end_DSP-begin_DSP;

    % Adjust Contact-vectors to include DSP where not measured
    ContactL(1:LengthDSP)=1;

    % Determine frame numbers for shifts between phases
    counter=0;
    for FrameNo=2:nFrames
       if ContactR(FrameNo)~=ContactR(FrameNo-1)
           counter=counter + 1;
           PhaseShiftR(counter) = FrameNo;
       end
```

```
    end

    counter=0;
    for FrameNo=2:nFrames
       if ContactL(FrameNo)~=ContactL(FrameNo-1)
           counter=counter + 1;
           PhaseShiftL(counter) = FrameNo;
       end
    end

    Data.PhaseShiftR=PhaseShiftR;
    Data.PhaseShiftL=PhaseShiftL;

    % Determine double support phase number
    DSP=1;
    DSPno(1:nFrames)=0;

    if ContactR(1)==1 && ContactL(1)==1
        DSPno(1)=1;
    end

    for FrameNo=2:nFrames
        if ( ContactR(FrameNo)==1 && ContactL(FrameNo)==1 )
            if (ContactR(FrameNo-1)~=ContactL(FrameNo-1))
            DSP=DSP+1;
            end
            DSPno(FrameNo)=DSP;
        end
    end

    Data.DSPno=DSPno;

% Load cases with both feet on the same force plate:
elseif strcmp(LC,'sit') || strcmp(LC,'raise') || strcmp(LC,'standref')

    LengthDSP=nFrames;
    Data.DSPno(nFrames)=1;
    Data.PhaseShiftR=[];
    Data.PhaseShiftL=[];
    ContactR(1:nFrames)=1;
    ContactL(1:nFrames)=1;

elseif strcmp(LC,'curb')
    % Determine length of double support phase
    for FrameNo=2:nFrames-1
        if (ContactL(FrameNo)==0)&& (ContactL(FrameNo+1)==1)
           begin_DSP=FrameNo;
        end

        if (ContactR(FrameNo)==0)&& (ContactR(FrameNo-1)==1)
            end_DSP=FrameNo;
        end
    end
    LengthDSP=end_DSP-begin_DSP;

    % Adjust Contact-vectors to include DSP where not measured
```

```
    ContactL(1:200)=1;
    ContactR(657:end)=1;


    % Determine frame numbers for shifts between phases
    counter=0;
    for FrameNo=2:nFrames
       if ContactR(FrameNo)~=ContactR(FrameNo-1)
           counter=counter + 1;
           PhaseShiftR(counter) = FrameNo;
       end
    end

    counter=0;
    for FrameNo=2:nFrames
       if ContactL(FrameNo)~=ContactL(FrameNo-1)
           counter=counter + 1;
           PhaseShiftL(counter) = FrameNo;
       end
    end

    Data.PhaseShiftR=PhaseShiftR;
    Data.PhaseShiftL=PhaseShiftL;

    % Determine double support phase number
    DSP=1;
    DSPno(1:nFrames)=0;

    if ContactR(1)==1 && ContactL(1)==1
        DSPno(1)=1;
    end

    for FrameNo=2:nFrames
        if ( ContactR(FrameNo)==1 && ContactL(FrameNo)==1 )
            if (ContactR(FrameNo-1)~=ContactL(FrameNo-1))
            DSP=DSP+1;
            end
            DSPno(FrameNo)=DSP;
        end
    end

    Data.DSPno=DSPno;

elseif strcmp(LC,'start_rightforce')

    % Right leg treads on both force plates
    ContactR=ContactR+ContactL;
    ContactL(:)=0;
    % Set contact for left leg
    ContactL(1:272)=1;
    % Correct


    % Determine frame numbers for shifts between phases
    counter=0;
    for FrameNo=2:nFrames
```

```
       if ContactR(FrameNo)~=ContactR(FrameNo-1)
           counter=counter + 1;
           PhaseShiftR(counter) = FrameNo;
       end
    end

    counter=0;
    for FrameNo=2:nFrames
       if ContactL(FrameNo)~=ContactL(FrameNo-1)
           counter=counter + 1;
           PhaseShiftL(counter) = FrameNo;
       end
    end

    Data.PhaseShiftR=PhaseShiftR;
    Data.PhaseShiftL=PhaseShiftL;

    % Determine double support phase number
    DSP=1;
    DSPno(1:nFrames)=0;

    if ContactR(1)==1 && ContactL(1)==1
        DSPno(1)=1;
    end

    for FrameNo=2:nFrames
        if ( ContactR(FrameNo)==1 && ContactL(FrameNo)==1 )
            if (ContactR(FrameNo-1)~=ContactL(FrameNo-1))
            DSP=DSP+1;
            end
            DSPno(FrameNo)=DSP;
        end
    end

    Data.DSPno=DSPno;

    LengthDSP=PhaseShiftL(1)-PhaseShiftR(2);

else % Other load cases
    LengthDSP=nFrames;
end


% Return results
Data.ContactR=ContactR;
Data.ContactL=ContactL;
Data.LengthDSP=LengthDSP;



% Plot force platform 1-2; Fx,Fy,Fz
if 1==0
   figure('name','Force platform results')
   subplot(3,1,1);
   plot(1:length(F1x),F1x,'-k',1:length(F1x),F1y,':k',1:length(F1x),F1z,'--k')
   xlabel('Frames')
```

```
   ylabel('Force [N]')
   title('Force platform 1')
   legend('F_x','F_y','F_z')
   grid on

   subplot(3,1,2);
   plot(1:length(F1x),F2x,'-k',1:length(F1x),F2y,':k',1:length(F1x),F2z,'--k');
   xlabel('Frames')
   ylabel('Force [N]')
   title('Force platform 2')
   legend('F_x','F_y','F_z')
   grid on

   subplot(3,1,3)
   plot(1:nFrames,ContactR,'r',1:nFrames,ContactL,'b')
   xlabel('Frames')
   title('Contact')
   legend('Right foot','Left foot')
   ylim([-1 2]);
end
```

```
% The local coordinate systems will be setup and transformation matrices
% will be created, based on Bryant angles (x-y-z) rotations.


%%%%%%%%%%%%% Initialize transformation matrices %%%%%%%%%%%%%%%%
A_0(1:3,1:3,1:nFrames)  = 0;
A_1(1:3,1:3,1:nFrames)  = 0;
A_2(1:3,1:3,1:nFrames)  = 0;
A_3(1:3,1:3,1:nFrames)  = 0;
A_4(1:3,1:3,1:nFrames)  = 0;
A_5(1:3,1:3,1:nFrames)  = 0;
A_6(1:3,1:3,1:nFrames)  = 0;
A_7(1:3,1:3,1:nFrames)  = 0;
A_8(1:3,1:3,1:nFrames)  = 0;
A_9(1:3,1:3,1:nFrames)  = 0;
A_10(1:3,1:3,1:nFrames) = 0;



for FrameNo = 1:nFrames
%%%%%%%%%%%%% A-matrix for global coordinate system %%%%%%%%%%%%%
    A_0(:,:,FrameNo);
    eta_0   = [1 0 0]';
    xi_0    = [0 1 0]';
    zeta_0  = [0 0 1]';

    A_0(:,:,FrameNo) = [xi_0,eta_0,zeta_0];


%%%%%%%%%%%%%%%%%%%%%%% A-matrix for torso (1)%%%%%%%%%%%%%%%%%%%%%
    r_3_2  = [l_shoulder(:,FrameNo) - r_shoulder(:,FrameNo)];
    eta_1  = unit(r_3_2);
    r_10_1 = [neck(:,FrameNo) - waist(:,FrameNo)];
    zeta_1 = unit(r_10_1);
    xi_1   = unit(skew(eta_1)*zeta_1);

    A_1(:,:,FrameNo) = [xi_1,eta_1,zeta_1];

%%%%%%%%%%%%%%%%%%%%%%% A-matrix for right arm (2)%%%%%%%%%%%%%%%%%
    eta_2       = eta_1;

    % The vector r_12_2 is projected in the plane normal to the unit vector eta_2
    r_12_2      = [r_shoulder(:,FrameNo) - r_wrist(:,FrameNo)];
    r_12_2_proj = r_12_2 - (dot(r_12_2,eta_2))/norm(eta_2)^2*eta_2;
    zeta_2      = unit(r_12_2_proj);
    xi_2        = unit(skew(eta_2)*zeta_2);

    A_2(:,:,FrameNo) = [xi_2,eta_2,zeta_2];

%%%%%%%%%%%%%%%%%%%%%%% A-matrix for left arm (3)%%%%%%%%%%%%%%%%%%
    eta_3       = eta_1;

    % The vector r_12_3 is projected in the plane normal to the unit vector eta_3
    r_11_3      = [l_shoulder(:,FrameNo) - l_wrist(:,FrameNo)];
    r_11_3_proj = r_11_3 - (dot(r_11_3,eta_3))/norm(eta_3)^2*eta_3;
    zeta_3      = unit(r_11_3_proj);
```

```
    xi_3        = unit(skew(eta_3)*zeta_3);

    A_3(:,:,FrameNo) = [xi_3,eta_3,zeta_3];

%%%%%%%%%%%%%%%%%%%%%%% A-matrix for pelvis (4)%%%%%%%%%%%%%%%%%%
    r_5_4   = [l_hip(:,FrameNo) - r_hip(:,FrameNo)];
    eta_4   = unit(r_5_4);

    % Determines the centerpoint of the vectors from left to right hip and the
    % center of the vector from left to right hip socket, and connects the
    % two centerpoints by a vector

    r_13    = 0.5*[l_hip(:,FrameNo) + r_hip(:,FrameNo)];
    r_14    = 0.5*[l_hip_socket(:,FrameNo) + r_hip_socket(:,FrameNo)];
    r_1_13  = [r_14 - r_13];
    zeta_4  = unit(r_1_13);
    xi_4    = unit(skew(eta_4)*zeta_4);

    A_4(:,:,FrameNo) = [xi_4,eta_4,zeta_4];

%%%%%%%%%%%%%%%%%%%%%% A-matrix for right thigh (5) %%%%%%%%%%%%%%%%%%
    %%%%%%%%%% Local coordinate system for right thigh %%%%%%%%%%
    r_4_6        = r_hip(:,FrameNo) - r_knee(:,FrameNo) - [0 -5 0]';
    zeta_5       = unit(r_4_6);
    r_thigh_hor  = r_thigh_f(:,FrameNo) - r_thigh_r(:,FrameNo);
    xi_r_thigh   = unit(r_thigh_hor);
    eta_5        = unit(skew(zeta_5)*xi_r_thigh);
    xi_5         = unit(skew(eta_5)*zeta_5);


    A_5(:,:,FrameNo) = [xi_5,eta_5,zeta_5];

%%%%%%%%%%%%%%%%%%%%%% A-matrix for left thigh (6) %%%%%%%%%%%%%%%%%%
    %%%%%%%%%% Local coordinate system for left thigh %%%%%%%%%%
    r_5_7        = l_hip(:,FrameNo) - l_knee(:,FrameNo)-[0 -5 0]';
    zeta_6       = unit(r_5_7);
    l_thigh_hor  = l_thigh_f(:,FrameNo) - l_thigh_r(:,FrameNo);
    xi_l_thigh   = unit(l_thigh_hor);
    eta_6        = unit(skew(zeta_6)*xi_l_thigh);
    xi_6         = unit(skew(eta_6)*zeta_6);


    A_6(:,:,FrameNo) = [xi_6,eta_6,zeta_6];

%%%%%%%%%%%%%%%%%%%%%% A-matrix for right shin (7) %%%%%%%%%%%%%%%%%%

% The vector from ankle to knee gets projected in to the plane normal to
% eta_7

    eta_7      = eta_5;
    r_6_8      = r_knee(:,FrameNo) - r_ankle(:,FrameNo);
    r_6_8_proj = r_6_8 - (dot(r_6_8,eta_7))/norm(eta_7)^2*eta_7;
    zeta_7     = unit(r_6_8_proj);
    xi_7       = unit(skew(eta_7)*zeta_7);
```

```
    A_7(:,:,FrameNo) = [xi_7,eta_7,zeta_7];

%%%%%%%%%%%%%%%%%%%% A-matrix for left shin (8) %%%%%%%%%%%%%%%%%%

% The vector from ankle to knee gets projected in to the plane normal to
% eta_8

    eta_8      = eta_6;
    r_7_9      = l_knee(:,FrameNo) - l_ankle(:,FrameNo);
    r_7_9_proj = r_7_9 - (dot(r_7_9,eta_8))/norm(eta_8)^2*eta_8;
    zeta_8  = unit(r_7_9_proj);
    xi_8    = unit(skew(eta_8)*zeta_8);


    A_8(:,:,FrameNo) = [xi_8,eta_8,zeta_8];

%%%%%%%%%%%%%%%%%%%% A-matrix for right foot (9) %%%%%%%%%%%%%%%%%%

    % a vector from heel to toe is made r_14_15
    r_14_15      = r_toe(:,FrameNo) - r_heel(:,FrameNo);
    % a vector from toe to ankle is made
    r_16_14      = r_ankle(:,FrameNo) - r_toe(:,FrameNo);
    % the vector r_14_16 is projected to a point on r_14_15
    r_16_14_proj = dot(r_16_14,unit(r_14_15))*unit(r_14_15);
    % the vector from the point to the old vector is made
    r_proj_16    = r_16_14 - r_16_14_proj;

    eta_9        = [0 1 0]';
    r_16_projproj= r_proj_16 - (dot(r_proj_16,eta_9))/norm(eta_9)^2*eta_9;
    zeta_9       = unit(r_16_projproj);
    xi_9         = unit(skew(eta_9)*zeta_9);



    A_9(:,:,FrameNo) = [xi_9,eta_9,zeta_9];

%%%%%%%%%%%%%%%%%%%% A-matrix for left foot (10) %%%%%%%%%%%%%%%%%%

    % a vector from heel to toe is made r_14_15
    r_17_18       = l_toe(:,FrameNo) - l_heel(:,FrameNo);
    % a vector from toe to ankle is made
    r_19_17       = l_ankle(:,FrameNo) - l_toe(:,FrameNo);
    % the vector r_19_17 is projected to r_17_18
    r_19_17_proj  = dot(r_19_17,unit(r_17_18))*unit(r_17_18);
    % the vector from the end of the projected vector to the old vector is made
    r_proj_19     = r_19_17 - r_19_17_proj;

    eta_10        = [0 1 0]';
    r_19_projproj = r_proj_19 - (dot(r_proj_19,eta_10))/norm(eta_10)^2*eta_10;
    zeta_10       = unit(r_19_projproj);
    xi_10         = unit(skew(eta_10)*zeta_10);



    A_10(:,:,FrameNo) = [xi_10,eta_10,zeta_10];
end
```

```
% Local coordinate system is plottet relative to the global coordinate
% system.
if 1 == 0;
    figure
    plot3([0 1],[0 0],[0 0],'r',[0 0],[0 1],[0 0,],'g',[0 0],[0 0],[0 1],'b',...
          [0 xi_6(1,1)],  [0 xi_6(2,1)],  [0 xi_6(3,1)],'-.r*',...
         [0 eta_6(1,1)], [0 eta_6(2,1)], [0 eta_6(3,1)],'-.g*',...
        [0 zeta_6(1,1)],[0,zeta_6(2,1)],[0 zeta_6(3,1)],'-.b*')
    legend('\xi','\eta','\zeta')
    axis equal
    title('Local coordinate system relative to the global coordinate system')
    grid on
end

clear r_10_1 r_11_3 r_11_3_proj r_12_2 r_12_2_proj r_13 r_14 r_14_15 r_16_14↙
r_16_14_proj...
      r_17_18 r_19_17 r_19_17_proj r_1_13 r_3_2 r_4_6 r_5_4 r_5_7 r_6_8 r_7_9 r_proj_16↙
r_proj_19

clear xi_1 xi_2 xi_3 xi_4 xi_5 xi_6 xi_7 xi_8 xi_9 xi_10 ...
      eta_1 eta_2 eta_3 eta_4 eta_5 eta_6 eta_7 eta_8 eta_9 eta_10 ...
      zeta_1 zeta_2 zeta_3 zeta_4 zeta_5 zeta_6 zeta_7 zeta_8 zeta_9 zeta_10

clear QTMfileNum l_thigh_hor l_thigh_ver r_thigh_hor r_thigh_ver xi_l_thigh xi_r_thigh↙
zeta_l_thigh zeta_r_thigh

% Insert A-matrices
Body(1).A  = A_1;
Body(2).A  = A_2;
Body(3).A  = A_3;
Body(4).A  = A_4;
Body(5).A  = A_5;
Body(6).A  = A_6;
Body(7).A  = A_7;
Body(8).A  = A_8;
Body(9).A  = A_9;
Body(10).A = A_10;
```

```
% Smooths the rotations stored in the A-matrices for all bodies
fprintf('\nSmoothing input')

% Initialize variables
xi(1:3,1:nFrames)=0;
eta(1:3,1:nFrames)=0;
zeta(1:3,1:nFrames)=0;
smoothed_xi(1:3,1:nFrames)=0;
smoothed_eta(1:3,1:nFrames)=0;
smoothed_zeta(1:3,1:nFrames)=0;
smoothed_A(1:3,1:3,1:nFrames)=0;

for BodyNo=1:nBodies


    for FrameNo=1:nFrames
        xi(:,FrameNo)=Body(BodyNo).A(:,1,FrameNo);
        eta(:,FrameNo)=Body(BodyNo).A(:,2,FrameNo);
        zeta(:,FrameNo)=Body(BodyNo).A(:,3,FrameNo);
    end

    smoothed_xi(1,:)=smooth(xi(1,:),0.1,'rloess');
    smoothed_xi(2,:)=smooth(xi(2,:),0.1,'rloess');
    smoothed_xi(3,:)=smooth(xi(3,:),0.1,'rloess');

    smoothed_eta(1,:)=smooth(eta(1,:),0.1,'rloess');
    smoothed_eta(2,:)=smooth(eta(2,:),0.1,'rloess');
    smoothed_eta(3,:)=smooth(eta(3,:),0.1,'rloess');

    smoothed_zeta(1,:)=smooth(zeta(1,:),0.1,'rloess');
    smoothed_zeta(2,:)=smooth(zeta(2,:),0.1,'rloess');
    smoothed_zeta(3,:)=smooth(zeta(3,:),0.1,'rloess');

    for FrameNo=1:nFrames
        smoothed_A(1:3,1:3,FrameNo)=...
            [smoothed_xi(1,FrameNo) smoothed_eta(1,FrameNo) smoothed_zeta(1,FrameNo);
             smoothed_xi(2,FrameNo) smoothed_eta(2,FrameNo) smoothed_zeta(2,FrameNo);
             smoothed_xi(3,FrameNo) smoothed_eta(3,FrameNo) smoothed_zeta(3,FrameNo)];
    end

    Body(BodyNo).A=smoothed_A;
end

fprintf('\nPerforming kinematic analysis')
```

```
function [A_rel]=CalcRelTrMatr(Body,Joint,JointNo,nFrames)
% Calculates the relative transformation matrices based on the absolute
% transformation matrices stored in Body.

%Initialize variables
A_rel(1:3,1:3,1:nFrames)=0;

j=Joint(JointNo).connects(1);
i=Joint(JointNo).connects(2);

% Get input
A_i=Body(i).A;
A_j=Body(j).A;

% Compute
for FrameNo=1:nFrames
   A_rel(:,:,FrameNo) = A_i(:,:,FrameNo)'*A_j(:,:,FrameNo);
end
```

```
function phi=CalcRelRot(A_j_i,nFrames)
% Calculates the relative rotations (phi) in the 3 directions in each
% frame defined by the relative transformation matrix A_j_i (Nikravesh p.352).


for FrameNo=1:nFrames

    % Extract transformation matrix associated with FrameNo
    A=A_j_i(:,:,FrameNo);

    % Calculate sine and cosine to the angles defined by A_j_i
    sTheta2 = A(1,3);
    cTheta2 = sqrt(1-sTheta2^2);

    cTheta3 =  A(1,1)/cTheta2;
    sTheta3 = -A(1,2)/cTheta2;

    sTheta1 = -A(2,3)/cTheta2;
    cTheta1 =  A(3,3)/cTheta2;

    % Return result
    phi(1,FrameNo) = atan2(sTheta1,cTheta1);
    phi(2,FrameNo) = atan2(sTheta2,cTheta2);
    phi(3,FrameNo) = atan2(sTheta3,cTheta3);
end
```

```
function [phiDot phiDotDot]=FitDif(phi,Tspan,precision)
% Calculates 1st derivative based on fitted smoothingsplines.

nFrames=length(phi(1,:));

if precision <=1
    % SmoothingParam(SP)=[0;1] determines the level of smoothing applied.
    % 1: No smoothing, 0: Very smooth.
    if precision==1
        SP(1)=0.9995;
        SP(2)=0.99;
        SP(3)=0.995;
    else
        SP(1:3)=precision;
    end
    % Fit curves using splines
    fit_phi1 = fit(Tspan,phi(1,:)', 'smoothingspline','SmoothingParam',SP(1));
    fit_phi2 = fit(Tspan,phi(2,:)', 'smoothingspline','SmoothingParam',SP(2));
    fit_phi3 = fit(Tspan,phi(3,:)', 'smoothingspline','SmoothingParam',SP(3));

    % Differentiate fitted objects
    [phiDot1 phiDotDot1]=differentiate(fit_phi1,Tspan);
    [phiDot2 phiDotDot2]=differentiate(fit_phi2,Tspan);
    [phiDot3 phiDotDot3]=differentiate(fit_phi3,Tspan);
else
    % Uses 5 point numerical differentiation, twice
    phiDot1=smooth(nd5p(phi(1,:),Tspan(1),nFrames),0.1,'rloess');
    phiDot2=smooth(nd5p(phi(2,:),Tspan(1),nFrames),0.1,'rloess');
    phiDot3=smooth(nd5p(phi(3,:),Tspan(1),nFrames),0.1,'rloess');

    phiDotDot1=nd5p(phiDot1,Tspan(1),nFrames);
    phiDotDot2=nd5p(phiDot2,Tspan(1),nFrames);
    phiDotDot3=nd5p(phiDot3,Tspan(1),nFrames);
end

% Combine and return results
phiDot(1,:)=phiDot1;
phiDot(2,:)=phiDot2;
phiDot(3,:)=phiDot3;

phiDotDot(1,:)=phiDotDot1;
phiDotDot(2,:)=phiDotDot2;
phiDotDot(3,:)=phiDotDot3;
```

```
function [T]=T(phi)
% Returns the angular velocities, based on the time derivative of the
% Bryant angle in phiDot and the Bryant angles themselves in phi.

% Transformation matrix T (Nikravesh p.352)
T=[ cos(phi(1))*cos(phi(3)) sin(phi(3)) 0;
   -cos(phi(2))*sin(phi(3)) cos(phi(3)) 0;
               sin(phi(2))           0 1];
```

```
% Extract angular rotation span + max relative velocity and acceleration
for JointNo = 1:nJoints
    Joint(JointNo).thetaRelSpanMaxX = max(max(Joint(JointNo).thetaRel(1,:)) - min(Joint↙
(JointNo).thetaRel(1,:)));
    Joint(JointNo).thetaRelSpanMaxY = max(max(Joint(JointNo).thetaRel(2,:)) - min(Joint↙
(JointNo).thetaRel(2,:)));
    Joint(JointNo).thetaRelSpanMaxZ = max(max(Joint(JointNo).thetaRel(3,:)) - min(Joint↙
(JointNo).thetaRel(3,:)));

    Joint(JointNo).omegaRelMax = max(max(abs(Joint(JointNo).omegaRel)));
    Joint(JointNo).omegaRelDotMax = max(max(abs(Joint(JointNo).omegaRelDot)));
end

% Extract angular rotation, min and max in each joint
for JointNo = 1:nJoints
    Joint(JointNo).thetaRelMinX = min(Joint(JointNo).thetaRel(1,:));
    Joint(JointNo).thetaRelMinY = min(Joint(JointNo).thetaRel(2,:));
    Joint(JointNo).thetaRelMinZ = min(Joint(JointNo).thetaRel(3,:));
    Joint(JointNo).thetaRelMaxX = max(Joint(JointNo).thetaRel(1,:));
    Joint(JointNo).thetaRelMaxY = max(Joint(JointNo).thetaRel(2,:));
    Joint(JointNo).thetaRelMaxZ = max(Joint(JointNo).thetaRel(3,:));
end
```

```
% Calculates the absolute angular velocities fot JointNo.
% Based on (Damsgaard, eq.248a)
% omega_j = omega_i + omega_j/i
%         = omega_i + A_i * omega_j/i^(i)
%
% Dimensions: omega=[1:3,1:nFrames]


% Calculate the absolute angular velocity of body1 (torso)
% omega_1 = omega_0 + A_0 * omega_1_0^(0)
A_1_0        = Body(1).A;
theta_1_0    = CalcRelRot(A_1_0,nFrames);
thetaDot_1_0 = FitDif(theta_1_0,Tspan,Dif_precision);
omega_1_0    = T(theta_1_0)*thetaDot_1_0;
omegaDot_1_0 = FitDif(omega_1_0,Tspan,Dif_precision);
Body(1).omega = omega_1_0; % since omega_0=0 and A_0=I(3)


% Calculate the absolute angular velocity of remaining bodies
for j=2:nBodies % j=BodyNo

    % Initialize variables
    i=Body(j).RelTo;
    k=Body(j).Joint;

    % Extract needed input data:
    omega_i=Body(i).omega;
    A_i=Body(i).A;
    omega_i_j=Joint(k).omegaRel;

    % Compute for all frames
    for FrameNo=1:nFrames
        omega_j(:,FrameNo) = omega_i(:,FrameNo) + A_i(:,:,FrameNo)*omega_i_j(:,↙
FrameNo);
    end

    % Return results to Body-struct
    Body(j).omega=omega_j;
end

% clear variables
clear A_1_0 A_i theta_1_0 thetaDot_1_0 omega_i omega_j omega_i_j...
      omegaDot_1_0 omegaDot_i omegaDot_j omegaDot_i_j
```

```
%function []=CalcAbsAngAcc()
% Calculates the absolute angular acceleration
% Based on (Damsgaard, eq.249a)
%
% omegaDot_j = omegaDot_i + omegaDot_j_i
%            = omegaDot_i + A_i*omegaDot_j_i^(i) + skew(omega_i)*A_i*omega_j_i^(i)


% Calculate the absolute angular acceleration of body 1
A_1_0            = Body(1).A;
theta_1_0        = CalcRelRot(A_1_0,nFrames);
thetaDot_1_0     = FitDif(theta_1_0,Tspan,Dif_precision);
omega_1_0        = T(theta_1_0)*thetaDot_1_0;
omegaDot_1_0     = FitDif(omega_1_0,Tspan,Dif_precision);
Body(1).omegaDot = omegaDot_1_0;


% Calculate the absolute angular velocity of body2 (Right arm), i=1, j=2
omega_i=Body(1).omega;
omegaDot_i=Body(1).omegaDot;
A_i=Body(1).A;
omega_i_j=Joint(2).omegaRel;
omegaDot_i_j=Joint(2).omegaRelDot;

for FrameNo=1:nFrames
    omegaDot_j(:,FrameNo) = omegaDot_i(:,FrameNo) + A_i(:,:,FrameNo)*omegaDot_i_j(:,↙
FrameNo)...
                          + skew(omega_i(:,FrameNo))*A_i(:,:,FrameNo)*omega_i_j(:,↙
FrameNo);
end

Body(2).omegaDot=omegaDot_j;


% Calculate the absolute angular acceleration of body3 (Left arm), i=1 ,j=3
omega_i=Body(1).omega;
omegaDot_i=Body(1).omegaDot;
A_i=Body(1).A;
omega_i_j=Joint(3).omegaRel;
omegaDot_i_j=Joint(3).omegaRelDot;

for FrameNo=1:nFrames
    omegaDot_j(:,FrameNo) = omegaDot_i(:,FrameNo) + A_i(:,:,FrameNo)*omegaDot_i_j(:,↙
FrameNo)...
                          + skew(omega_i(:,FrameNo))*A_i(:,:,FrameNo)*omega_i_j(:,↙
FrameNo);
end

Body(3).omegaDot=omegaDot_j;

% Calculate the absolute angular acceleration of body4 (Pelvis), i=1 ,j=4
omega_i=Body(1).omega;
omegaDot_i=Body(1).omegaDot;
A_i=Body(1).A;
omega_i_j=Joint(1).omegaRel;
omegaDot_i_j=Joint(1).omegaRelDot;
```

```
for FrameNo=1:nFrames
    omegaDot_j(:,FrameNo) = omegaDot_i(:,FrameNo) + A_i(:,:,FrameNo)*omegaDot_i_j(:,↙
FrameNo)...
                            + skew(omega_i(:,FrameNo))*A_i(:,:,FrameNo)*omega_i_j(:,↙
FrameNo);
end

Body(4).omegaDot=omegaDot_j;

% Calculate the absolute angular acceleration of body5 (Right thigh), i=4, j=5
omega_i=Body(4).omega;
omegaDot_i=Body(4).omegaDot;
A_i=Body(4).A;
omega_i_j=Joint(4).omegaRel;
omegaDot_i_j=Joint(4).omegaRelDot;

for FrameNo=1:nFrames
    omegaDot_j(:,FrameNo) = omegaDot_i(:,FrameNo) + A_i(:,:,FrameNo)*omegaDot_i_j(:,↙
FrameNo)...
                            + skew(omega_i(:,FrameNo))*A_i(:,:,FrameNo)*omega_i_j(:,↙
FrameNo);
end

Body(5).omegaDot=omegaDot_j;

% Calculate the absolute angular acceleration of body6 (Left thigh), i=4, j=6
omega_i=Body(4).omega;
omegaDot_i=Body(4).omegaDot;
A_i=Body(4).A;
omega_i_j=Joint(5).omegaRel;
omegaDot_i_j=Joint(5).omegaRelDot;

for FrameNo=1:nFrames
    omegaDot_j(:,FrameNo) = omegaDot_i(:,FrameNo) + A_i(:,:,FrameNo)*omegaDot_i_j(:,↙
FrameNo)...
                            + skew(omega_i(:,FrameNo))*A_i(:,:,FrameNo)*omega_i_j(:,↙
FrameNo);
end

Body(6).omegaDot=omegaDot_j;

% Calculate the absolute angular acceleration of body7 (Right shin), i=5,j=7
omega_i=Body(5).omega;
omegaDot_i=Body(5).omegaDot;
A_i=Body(5).A;
omega_i_j=Joint(6).omegaRel;
omegaDot_i_j=Joint(6).omegaRelDot;

for FrameNo=1:nFrames
    omegaDot_j(:,FrameNo) = omegaDot_i(:,FrameNo) + A_i(:,:,FrameNo)*omegaDot_i_j(:,↙
FrameNo)...
                            + skew(omega_i(:,FrameNo))*A_i(:,:,FrameNo)*omega_i_j(:,↙
FrameNo);
end
```

```
Body(7).omegaDot=omegaDot_j;

% Calculate the absolute angular acceleration of body8 (Left shin), i=6,j=8
omega_i=Body(6).omega;
omegaDot_i=Body(6).omegaDot;
A_i=Body(6).A;
omega_i_j=Joint(7).omegaRel;
omegaDot_i_j=Joint(7).omegaRelDot;

for FrameNo=1:nFrames
    omegaDot_j(:,FrameNo) = omegaDot_i(:,FrameNo) + A_i(:,:,FrameNo)*omegaDot_i_j(:,↙
FrameNo)...
                            + skew(omega_i(:,FrameNo))*A_i(:,:,FrameNo)*omega_i_j(:,↙
FrameNo);
end

Body(8).omegaDot=omegaDot_j;

% Calculate the absolute angular acceleration of body9 (Right foot), i=7,j=9
omega_i=Body(7).omega;
omegaDot_i=Body(7).omegaDot;
A_i=Body(7).A;
omega_i_j=Joint(8).omegaRel;
omegaDot_i_j=Joint(8).omegaRelDot;

for FrameNo=1:nFrames
    omegaDot_j(:,FrameNo) = omegaDot_i(:,FrameNo) + A_i(:,:,FrameNo)*omegaDot_i_j(:,↙
FrameNo)...
                            + skew(omega_i(:,FrameNo))*A_i(:,:,FrameNo)*omega_i_j(:,↙
FrameNo);
end

Body(9).omegaDot=omegaDot_j;

% Calculate the absolute angular acceleration of body10 (Left foot), i=8,j=10
omega_i=Body(8).omega;
omegaDot_i=Body(8).omegaDot;
A_i=Body(8).A;
omega_i_j=Joint(9).omegaRel;
omegaDot_i_j=Joint(9).omegaRelDot;

for FrameNo=1:nFrames
    omegaDot_j(:,FrameNo) = omegaDot_i(:,FrameNo) + A_i(:,:,FrameNo)*omegaDot_i_j(:,↙
FrameNo)...
                            + skew(omega_i(:,FrameNo))*A_i(:,:,FrameNo)*omega_i_j(:,↙
FrameNo);
end

Body(10).omegaDot=omegaDot_j;
```

```
% Calculates the absolute position, velocity and acceleration of each local CoM

% s-vectors: describes the geometry
% s_i_j: coordinates from body j to body i, in local j-coordinatesystem.


% Time-variable global position vector of the waist joint.
%s_1_0=waist*10^-3;
s_1_0_x=smooth(waist(1,:)*10^-3,0.1,'rloess');
s_1_0_y=smooth(waist(2,:)*10^-3,0.1,'rloess');
s_1_0_z=smooth(waist(3,:)*10^-3,0.1,'rloess')-0.0667;
s_1_0=[s_1_0_x s_1_0_y s_1_0_z]';
[s_1_0_dot s_1_0_dotdot]=FitDif(s_1_0,Tspan,1);

% Constant local vectors between CoM and joints.
s_2_1 = geo.s_2_1;
s_3_1 = geo.s_3_1;
s_4_1 = geo.s_4_1;
s_1_2 = geo.s_1_2;
s_1_3 = geo.s_1_3;
s_1_4 = geo.s_1_4;
s_5_4 = geo.s_5_4;
s_6_4 = geo.s_6_4;
s_4_5 = geo.s_4_5;
s_7_5 = geo.s_7_5;
s_4_6 = geo.s_4_6;
s_8_6 = geo.s_8_6;
s_6_8 = geo.s_6_8;
s_10_8= geo.s_10_8;
s_5_7 = geo.s_5_7;
s_9_7 = geo.s_9_7;
s_7_9 = geo.s_7_9;
s_8_10= geo.s_8_10;
s_t_9 = geo.s_t_9;
s_h_9 = geo.s_h_9;
s_t_10= geo.s_t_10;
s_h_10= geo.s_h_10;
s_w_2 = geo.s_w_2;
s_w_3 = geo.s_w_3;

%r_j=r_i+A_i*s_j_i(i), r_j= Global vector to local center of mass, r_i_j= Global vector↙
to joint
for FrameNo = 1:nFrames

    % Get A-matrices
    A_1=Body(1).A(:,:,FrameNo); A_2=Body(2).A(:,:,FrameNo); A_3=Body(3).A(:,:,FrameNo);
    A_4=Body(4).A(:,:,FrameNo); A_5=Body(5).A(:,:,FrameNo); A_6=Body(6).A(:,:,FrameNo);
    A_7=Body(7).A(:,:,FrameNo); A_8=Body(8).A(:,:,FrameNo); A_9=Body(9).A(:,:,FrameNo);
    A_10=Body(10).A(:,:,FrameNo);

    %%%%%%% Calculate POSITION %%%%%%%
        % Starting point (waist joint)
        r_0_1   = s_1_0(:,FrameNo);
        % Body 1 vector (torso+head)
        r_1     = r_0_1 + A_1*(-s_4_1);
        % Body 2 vector(torso to right arm)
```

```
        r_1_2   = r_1+A_1*s_2_1;
        r_2     = r_1_2+A_2*(-s_1_2);
        % Body 3 vector(torso to left arm)
        r_1_3   = r_1 + A_1*s_3_1;
        r_3     = r_1_3 + A_3*(-s_1_3);
        % Body 4 vector(torso to pelvis)
        r_1_4   = r_1 + A_1*s_4_1;
        r_4     = r_1_4 + A_4*(-s_1_4);
        % Body 5 vector(pelvis to right thigh)
        r_4_5   = r_4 + A_4*s_5_4;
        r_5     = r_4_5 + A_5*(-s_4_5);
        % Body 6 vector(pelvis to left thigh)
        r_4_6   = r_4 + A_4*s_6_4;
        r_6     = r_4_6 + A_6*(-s_4_6);
        % Body 7 vector(right thigh to right shin)
        r_5_7   = r_5 + A_5*s_7_5;
        r_7     = r_5_7 + A_7*(-s_5_7);
        % Body 8 vector(left thigh to left shin)
        r_6_8   = r_6 + A_6*s_8_6;
        r_8     = r_6_8 + A_8*(-s_6_8);
        % Body 9 vector(right shin to right foot)
        r_7_9   = r_7 + A_7*s_9_7;
        r_9     = r_7_9 + A_9*(-s_7_9);
        % Body 10 vector(left shin to left foot)
        r_8_10  = r_8 + A_8*s_10_8;
        r_10    = r_8_10 + A_10*(-s_8_10);

        % Return r vectors to Body-struct
        Body(1).r(:,FrameNo)=r_1; Body(2).r(:,FrameNo)=r_2; Body(3).r(:,FrameNo)=r_3;
        Body(4).r(:,FrameNo)=r_4; Body(5).r(:,FrameNo)=r_5; Body(6).r(:,FrameNo)=r_6;
        Body(7).r(:,FrameNo)=r_7; Body(8).r(:,FrameNo)=r_8; Body(9).r(:,FrameNo)=r_9;
        Body(10).r(:,FrameNo)=r_10;

        % Return joint positions to joint struct
        Joint(1).r(:,FrameNo)=r_0_1; Joint(2).r(:,FrameNo)=r_1_2; Joint(3).r(:,FrameNo)↙
=r_1_3;
        Joint(4).r(:,FrameNo)=r_4_5; Joint(5).r(:,FrameNo)=r_4_6; Joint(6).r(:,FrameNo)↙
=r_5_7;
        Joint(7).r(:,FrameNo)=r_6_8; Joint(8).r(:,FrameNo)=r_7_9; Joint(9).r(:,FrameNo)↙
=r_8_10;


    %%%%%%% Calculate VELOCITY %%%%%%%
        % Velocity of position; r_i_dot=velocity of local mass point, r_i_j_dot= vel of↙
joint point
        % r_j_dot=r_i_dot + skew(omega_i)*A_i*s_j_i+(A_i*s_j_i_dot=0)

        % Get omega_i
        omega_1=Body(1).omega(:,FrameNo); omega_2=Body(2).omega(:,FrameNo);↙
omega_3=Body(3).omega(:,FrameNo);
        omega_4=Body(4).omega(:,FrameNo); omega_5=Body(5).omega(:,FrameNo);↙
omega_6=Body(6).omega(:,FrameNo);
        omega_7=Body(7).omega(:,FrameNo); omega_8=Body(8).omega(:,FrameNo);↙
omega_9=Body(9).omega(:,FrameNo);
        omega_10=Body(10).omega(:,FrameNo);
```

```
        % input velocity
        r_0_dot = s_1_0_dot(:,FrameNo);
        % Vel body 1 (torso+head)
        r_1_dot= r_0_dot+skew(omega_1)*A_1*(-s_4_1);
        % Vel body 2 (torso+head to right arm)

        r_2_dot= r_1_dot + skew(omega_1)*A_1*s_2_1 + skew(omega_2)*A_2*-s_1_2;

        % Vel body 3 (torso+head to left arm)
        r_1_3_dot= r_1_dot+skew(omega_1)*A_1*s_3_1;
        r_3_dot= r_1_3_dot+skew(omega_3)*A_3*-s_1_3;
        % Vel body 4 (torso+head to pelwis)
        r_1_4_dot= r_1_dot+skew(omega_1)*A_1*s_4_1;
        r_4_dot= r_1_4_dot+skew(omega_4)*A_4*-s_1_4;
        % Vel body 5 (pelwis to right thigh)
        r_4_5_dot= r_4_dot+skew(omega_4)*A_4*s_5_4;
        r_5_dot= r_4_5_dot+skew(omega_5)*A_5*-s_4_5;
        % Vel body 6 (pelwis to left thigh)
        r_4_6_dot= r_4_dot+skew(omega_4)*A_4*s_6_4;
        r_6_dot= r_4_6_dot+skew(omega_6)*A_6*-s_4_6;
        % Vel body 7 (right thigh to right shin)
        r_5_7_dot= r_5_dot+skew(omega_5)*A_5*s_7_5;
        r_7_dot= r_5_7_dot+skew(omega_7)*A_7*-s_5_7;
        % Vel body 8 (left thigh to left shin)
        r_6_8_dot= r_6_dot+skew(omega_6)*A_6*s_8_6;
        r_8_dot= r_6_8_dot+skew(omega_8)*A_8*-s_6_8;
        % Vel body 9 (right shin to right foot)
        r_7_9_dot= r_7_dot+skew(omega_7)*A_7*s_9_7;
        r_9_dot= r_7_9_dot+skew(omega_9)*A_9*-s_7_9;
        % Vel body 10 (left shin to left foot)
        r_8_10_dot= r_8_dot+skew(omega_8)*A_8*s_10_8;
        r_10_dot= r_8_10_dot+skew(omega_10)*A_10*-s_8_10;

        % Return r_dot to Body-struct
        Body(1).rDot(:,FrameNo)=r_1_dot; Body(2).rDot(:,FrameNo)=r_2_dot; Body(3).rDot↙
(:,FrameNo)=r_3_dot;
        Body(4).rDot(:,FrameNo)=r_4_dot; Body(5).rDot(:,FrameNo)=r_5_dot; Body(6).rDot↙
(:,FrameNo)=r_6_dot;
        Body(7).rDot(:,FrameNo)=r_7_dot; Body(8).rDot(:,FrameNo)=r_8_dot; Body(9).rDot↙
(:,FrameNo)=r_9_dot;
        Body(10).rDot(:,FrameNo)=r_10_dot;


    %%%%%%% Calculate VELOCITY %%%%%%%
        % Acceleration of CoM; r_i_dotdot=acceleration of local mass point,↙
r_i_j_dotdot=acc of joint point
        % r_j_dot=r_i_dotdot + skew(omega_i_dot)*A_i*s_j_i+skew(omega_i)*skew(omega_i)↙
*A*s_j_i+(2*skew(omega_i)*Ai*s_i_dot+A_i*s_j_i_dotdot=0)

        % Get omegaDot
        omega_1_dot=Body(1).omegaDot(:,FrameNo); omega_2_dot=Body(2).omegaDot(:,↙
FrameNo); omega_3_dot=Body(3).omegaDot(:,FrameNo);
        omega_4_dot=Body(4).omegaDot(:,FrameNo); omega_5_dot=Body(5).omegaDot(:,↙
FrameNo); omega_6_dot=Body(6).omegaDot(:,FrameNo);
        omega_7_dot=Body(7).omegaDot(:,FrameNo); omega_8_dot=Body(8).omegaDot(:,↙
FrameNo); omega_9_dot=Body(9).omegaDot(:,FrameNo);
```

```
        omega_10_dot=Body(10).omegaDot(:,FrameNo);

        % Acc start point
        r_0_dotdot= s_1_0_dotdot(:,FrameNo);
        % Acc body 1
        r_1_dotdot=r_0_dotdot + skew(omega_1_dot)*A_1*(-s_4_1) + skew(omega_1)*skew↙
(omega_1)*A_1*(-s_4_1);
        % Acc body 2
        r_1_2_dotdot=r_1_dotdot + skew(omega_1_dot)*A_1*s_2_1 + skew(omega_1)*skew↙
(omega_1)*A_1*s_2_1;
        r_2_dotdot=r_1_2_dotdot + skew(omega_2_dot)*A_2*-s_1_2 + skew(omega_2)*skew↙
(omega_2)*A_2*-s_1_2;
        % Acc body 3
        r_1_3_dotdot=r_1_dotdot + skew(omega_1_dot)*A_1*s_3_1 + skew(omega_1)*skew↙
(omega_1)*A_1*s_3_1;
        r_3_dotdot=r_1_3_dotdot + skew(omega_3_dot)*A_3*-s_1_3 + skew(omega_3)*skew↙
(omega_3)*A_3*-s_1_3;
        % Acc body 4
        r_1_4_dotdot=r_1_dotdot + skew(omega_1_dot)*A_1*s_4_1 + skew(omega_1)*skew↙
(omega_1)*A_1*s_4_1;
        r_4_dotdot=r_1_4_dotdot + skew(omega_4_dot)*A_4*-s_1_4 + skew(omega_4)*skew↙
(omega_4)*A_4*-s_1_4;
        % Acc body 5
        r_4_5_dotdot=r_4_dotdot + skew(omega_4_dot)*A_4*s_5_4 + skew(omega_4)*skew↙
(omega_4)*A_4*s_5_4;
        r_5_dotdot=r_4_5_dotdot + skew(omega_5_dot)*A_5*-s_4_5 + skew(omega_5)*skew↙
(omega_5)*A_5*-s_4_5;
        % Acc body 6
        r_4_6_dotdot=r_4_dotdot + skew(omega_4_dot)*A_4*s_6_4 + skew(omega_4)*skew↙
(omega_4)*A_4*s_6_4;
        r_6_dotdot=r_4_6_dotdot + skew(omega_6_dot)*A_6*-s_4_6 + skew(omega_6)*skew↙
(omega_6)*A_6*-s_4_6;
        % Acc body 7
        r_5_7_dotdot=r_5_dotdot + skew(omega_5_dot)*A_5*s_7_5 + skew(omega_5)*skew↙
(omega_5)*A_5*s_7_5;
        r_7_dotdot=r_5_7_dotdot + skew(omega_7_dot)*A_7*-s_5_7 + skew(omega_7)*skew↙
(omega_7)*A_7*-s_5_7;
        % Acc body 8
        r_6_8_dotdot=r_6_dotdot + skew(omega_6_dot)*A_6*s_8_6 + skew(omega_6)*skew↙
(omega_6)*A_6*s_8_6;
        r_8_dotdot=r_6_8_dotdot + skew(omega_8_dot)*A_8*-s_6_8 + skew(omega_8)*skew↙
(omega_8)*A_8*-s_6_8;
        % Acc body 9
        r_7_9_dotdot=r_7_dotdot + skew(omega_7_dot)*A_7*s_9_7 + skew(omega_7)*skew↙
(omega_7)*A_7*s_9_7;
        r_9_dotdot=r_7_9_dotdot + skew(omega_9_dot)*A_9*-s_7_9 + skew(omega_9)*skew↙
(omega_9)*A_9*-s_7_9;
        % Acc body 10
        r_8_10_dotdot=r_8_dotdot + skew(omega_8_dot)*A_8*s_10_8 + skew(omega_8)*skew↙
(omega_8)*A_8*s_10_8;
        r_10_dotdot=r_8_10_dotdot + skew(omega_10_dot)*A_10*-s_8_10 + skew(omega_10)↙
*skew(omega_10)*A_10*-s_8_10;

        % Return rDotDot to Body-struct
        Body(1).rDotDot(:,FrameNo)=r_1_dotdot; Body(2).rDotDot(:,FrameNo)=r_2_dotdot;↙
Body(3).rDotDot(:,FrameNo)=r_3_dotdot;
```

```
        Body(4).rDotDot(:,FrameNo)=r_4_dotdot; Body(5).rDotDot(:,FrameNo)=r_5_dotdot;↙
Body(6).rDotDot(:,FrameNo)=r_6_dotdot;
        Body(7).rDotDot(:,FrameNo)=r_7_dotdot; Body(8).rDotDot(:,FrameNo)=r_8_dotdot;↙
Body(9).rDotDot(:,FrameNo)=r_9_dotdot;
        Body(10).rDotDot(:,FrameNo)=r_10_dotdot;
end

clear A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_10
```

```
function [CoM]=CalcCoM(Body,nFrames)
% Calculates the CoM trajectory

nBodies=length(Body);
CoM(1:3,1:nFrames)=0;
TotMass=0;
for BodyNo=1:nBodies
    TotMass=TotMass+Body(BodyNo).m;
end

for FrameNo=1:nFrames
    sum_mr=0;
    for BodyNo=1:nBodies
        m=Body(BodyNo).m;
        r=Body(BodyNo).r(:,FrameNo);
        sum_mr=sum_mr+m*r;
    end
    CoM(:,FrameNo)=sum_mr/TotMass;
end
```

```
% Kinetics of the analysis
fprintf('\nPerforming kinetic analysis')

% Initialize variables
nCoords=nBodies*6;
qDotDot(1:nCoords,1:nFrames)=0;
M_global(1:3,1:3,1:nFrames)=0;
J_global(1:3,1:3,1:nFrames)=0;
b(1:nCoords,1:nFrames)=0;
p(1:nCoords,1:nFrames)=0;
g(1:nCoords,1) = 0;

% Setup vector of accelerations: qDotDot
for BodyNo=1:nBodies
    counter=[1:6] +(6*(BodyNo-1));
    qDotDot(counter,:)=[Body(BodyNo).rDotDot; Body(BodyNo).omegaDot];
end

% Substitutes values in the right places, and multiplies masses.
for i = 1:10;
    g(3+((i-1)*6),1) = -9.82 * Body(i).m;
end
Data.g=g;

% Compute global inertia-matrix for all bodies
for BodyNo=1:nBodies
    for FrameNo=1:nFrames
        A=Body(BodyNo).A(:,:,FrameNo);
        J=Body(BodyNo).J;
        J_global(:,:,FrameNo)=A'*J*A;
    end
    Body(BodyNo).J_global=J_global;
end

% Compute global mass-matrix for all frames
for FrameNo=1:nFrames
    for BodyNo=1:nBodies
        counter=[1:6] +(6*(BodyNo-1));
        M_global(counter,counter,FrameNo)=[eye(3)*Body(BodyNo).m zeros(3); zeros(3)↙
Body(BodyNo).J_global(:,:,FrameNo)];
    end
end

% Setup b-vector containing velocity-terms
for FrameNo=1:nFrames
    for BodyNo=1:nBodies
        counter=[4:6] +(6*(BodyNo-1));
        omega=Body(BodyNo).omega(:,FrameNo);
        J=Body(BodyNo).J_global(:,:,FrameNo);
        b(counter,FrameNo)=skew(omega)*J*omega;
    end
end

% Collect right hand side of equation system
for FrameNo=1:nFrames
   p(:,FrameNo)=-g(:)+b(:,FrameNo)+M_global(:,:,FrameNo)*qDotDot(:,FrameNo);
```

```
end
Data.p = p;

% Calculate ZMP trajectory
Data.ZMP=CalcZMP(Body,nFrames);

% Calculate total external reactions
CalcTotReactions

% Solve equation system
[Reactions DataOut]=SolveEqSys(Body,Data,geo,nFrames);

% Sort assorted outputs
Data.RR=DataOut.RR;
Data.RS=DataOut.RS;
Data.armR=DataOut.armR;
Data.armL=DataOut.armL;
Data.fR=DataOut.fR;
Data.fL=DataOut.fL;
Data.MAP=DataOut.MAP;
Data.RAP=DataOut.RAP;


% Plot current distribution of forces/moments
% figure
% plot(FrameSpan,Data.fR,'r',FrameSpan,Data.fL,'b')

% Add reaction forces and moments to the Joint struct
Joint(1).R  = Reactions(1:3,:);
Joint(1).M  = Reactions(4:6,:);
Joint(2).R  = Reactions(7:9,:);
Joint(2).M  = Reactions(10:12,:);
Joint(3).R  = Reactions(13:15,:);
Joint(3).M  = Reactions(16:18,:);
Joint(4).R  = Reactions(19:21,:);
Joint(4).M  = Reactions(22:24,:);
Joint(5).R  = Reactions(25:27,:);
Joint(5).M  = Reactions(28:30,:);
Joint(6).R  = Reactions(31:33,:);
Joint(6).M  = Reactions(34:36,:);
Joint(7).R  = Reactions(37:39,:);
Joint(7).M  = Reactions(40:42,:);
Joint(8).R  = Reactions(43:45,:);
Joint(8).M  = Reactions(46:48,:);
Joint(9).R  = Reactions(49:51,:);
Joint(9).M  = Reactions(52:54,:);

% Virtual joints. Connects feet to the ground
Joint(10).R = Reactions(55:57,:);
Joint(10).M = Reactions(58:60,:);
Joint(11).R = Reactions(61:63,:);
Joint(11).M = Reactions(64:66,:);
Joint(12).R = Reactions(67:69,:);
Joint(12).M = Reactions(70:72,:);

% Extracts maximum values from Joint(JointNo).R and Joint(JointNo).M
```

```
for i = 1:nJoints+3
    Joint(i).R_xMax = max(abs(Joint(i).R(1,:)));
    Joint(i).R_yMax = max(abs(Joint(i).R(2,:)));
    Joint(i).R_zMax = max(abs(Joint(i).R(3,:)));
    Joint(i).M_xMax = max(abs(Joint(i).M(1,:)));
    Joint(i).M_yMax = max(abs(Joint(i).M(2,:)));
    Joint(i).M_zMax = max(abs(Joint(i).M(3,:)));
end

% Equilibrium check:
if 1==0
figure('name','Equilibrium check')

    subplot(2,1,1)
    hold on
    plot(FrameSpan,Joint(1).R(1,:),'r',FrameSpan,Joint(1).R(2,:),'g',FrameSpan,Joint↙
(1).R(3,:),'b')
    plot(FrameSpan,Data.RAP(1,:),':r',FrameSpan,Data.RAP(2,:),':g',FrameSpan,Data.RAP↙
(3,:),':b')
    title('Equilibrium check: waist joint reactions, calculated in two ways, must↙
coincide.')

    subplot(2,1,2)
    hold on
    plot(FrameSpan,Joint(1).M(1,:),'r',FrameSpan,Joint(1).M(2,:),'g',FrameSpan,Joint↙
(1).M(3,:),'b')
    plot(FrameSpan,Data.MAP(1,:),':r',FrameSpan,Data.MAP(2,:),':g',FrameSpan,Data.MAP↙
(3,:),':b')
    title('Equilibrium check: waist joint moments, calculated in two ways, must↙
coincide.')
end
```

```
function [ZMP]=CalcZMP(Body,nFrames)
% Calculates the Zero Moment Point (ZMP)

ZMP(1:3,1:nFrames)=0;
nBodies=length(Body);
g=9.82;

% Explicit calculation of ZMP [1]
for FrameNo=1:nFrames

    % Calculate commom denominator
    den=0;
    for BodyNo=1:nBodies
        m=Body(BodyNo).m;
        rDotDot_z=Body(BodyNo).rDotDot(3,FrameNo);
        den=den + m*(rDotDot_z+g);
    end

    % Calculate components ZMP for current frame
    num_x=0; num_y=0;
    for BodyNo=1:nBodies

        % Get input
        m=Body(BodyNo).m;

        r_x=Body(BodyNo).r(1,FrameNo);
        r_y=Body(BodyNo).r(2,FrameNo);
        r_z=Body(BodyNo).r(3,FrameNo);

        rDotDot_x=Body(BodyNo).rDotDot(1,FrameNo);
        rDotDot_y=Body(BodyNo).rDotDot(2,FrameNo);
        rDotDot_z=Body(BodyNo).rDotDot(3,FrameNo);

        J=Body(BodyNo).J_global(:,:,FrameNo);
        omega=Body(BodyNo).omega(:,FrameNo);
        omegaDot=Body(BodyNo).omegaDot(:,FrameNo);

        Ldot = J*omegaDot + skew(omega)*J*omega;

        % Calculate numerator
        num_x = num_x + m*(rDotDot_z+g)*r_x - m*rDotDot_x*r_z - Ldot(2);
        num_y = num_y + m*(rDotDot_z+g)*r_y - m*rDotDot_y*r_z + Ldot(1);

    end

    % Compute ZMP
    zmp_x = num_x/den;
    zmp_y = num_y/den;
    zmp_z = 0;

    % Return results
    ZMP(1:3,FrameNo)=[zmp_x zmp_y zmp_z]';
end
% [1] "Planning Walking Patterns for a Biped Robot", Huang Q. et al.,
%     IEEE Transactions on Robotics and Automation, vol.17 no.3, june 2001.
```

```
% Calculates the total reaction force and moment in the virtual point T on
% the ground. T=ZMP.

% Initiate variables
RT(1:3,1:nFrames)=0;
MT(1:3,1:nFrames)=0;

% Set virtual common contact point equal to ZMP
T=Data.ZMP;

clear m a J omega omegaDot w rw Ldot

% Determine reactions by summation
for FrameNo=1:nFrames

    % Do summation over nBodies
    for BodyNo=1:nBodies

        % Get current input
        m        = Body(BodyNo).m;
        a        = Body(BodyNo).rDotDot(:,FrameNo);
        J        = Body(BodyNo).J_global(:,:,FrameNo);
        omega    = Body(BodyNo).omega(:,FrameNo);
        omegaDot = Body(BodyNo).omegaDot(:,FrameNo);
        w        = [0 0 -9.82]';
        rw       = Body(BodyNo).r(:,FrameNo) - T(:,FrameNo);
        Ldot     = J*omegaDot + skew(omega)*J*omega;

        % Summation of forces
        RT(1:3,FrameNo) = RT(1:3,FrameNo) - m*w + m*a;
        % Summation of moments, taken about virtual ground connection point
        MT(1:3,FrameNo) = MT(1:3,FrameNo) - skew(rw)*m*w + Ldot + skew(rw)*(a*m);
    end
end

% Determine extra vectors for reaction computations
s_R_T(1:3,1:nFrames)=0; % right ankle to virtual middle contact point
s_S_T(1:3,1:nFrames)=0; % left ankle to virtual middle contact point
d(1:nFrames)=0; % distance between ankles.

for FrameNo=1:nFrames
   d(FrameNo)=norm(Body(9).r(:,FrameNo)-Body(10).r(:,FrameNo));
   s_R_T(:,FrameNo)=Body(9).r(:,FrameNo)-T(:,FrameNo);
   s_S_T(:,FrameNo)=Body(10).r(:,FrameNo)-T(:,FrameNo);
end

% Return results
Data.RT=RT;
Data.MT=MT;
Data.T=T;

geo.d=d;
geo.s_R_T=s_R_T;
geo.s_S_T=s_S_T;
```

```
function [Reactions DataOut]=SolveEqSys(Body,Data,geo,nFrames)
% Setup system of equilibrium equations and solve
for FrameNo=1:nFrames % Main frame loop

% Get total external reactions and right hand sides.
RT = Data.RT(:,FrameNo); MT = Data.MT(:,FrameNo);
R_seat=[0 0 0]'; M_seat=[0 0 0]'; P=Data.p;
MR=[0 0 0]'; MS=[0 0 0]';

% Get current right hand side of equations; Pi, i= body no
P1=P(1:6,FrameNo); P2=P(7:12,FrameNo); P3=P(13:18,FrameNo); P4=P(19:24,FrameNo);
P5=P(25:30,FrameNo); P6=P(31:36,FrameNo); P7=P(37:42,FrameNo); P8=P(43:48,FrameNo);
P9=P(49:54,FrameNo); P10=P(55:60,FrameNo);

% Get current A-matrices
A_1=Body(1).A(:,:,FrameNo); A_2=Body(2).A(:,:,FrameNo); A_3=Body(3).A(:,:,FrameNo);
A_4=Body(4).A(:,:,FrameNo); A_5=Body(5).A(:,:,FrameNo); A_6=Body(6).A(:,:,FrameNo);
A_7=Body(7).A(:,:,FrameNo); A_8=Body(8).A(:,:,FrameNo); A_9=Body(9).A(:,:,FrameNo);
A_10=Body(10).A(:,:,FrameNo);

% Create current skew matrices for moment-calculation
s2_1 = skew(A_1*geo.s_2_1);
s3_1 = skew(A_1*geo.s_3_1);
s4_1 = skew(A_1*geo.s_4_1);
s1_2 = skew(A_2*geo.s_1_2);
s1_3 = skew(A_3*geo.s_1_3);
s1_4 = skew(A_4*geo.s_1_4);
s5_4 = skew(A_4*geo.s_5_4);
s6_4 = skew(A_4*geo.s_6_4);
s4_5 = skew(A_5*geo.s_4_5);
s7_5 = skew(A_5*geo.s_7_5);
s4_6 = skew(A_6*geo.s_4_6);
s8_6 = skew(A_6*geo.s_8_6);
s6_8 = skew(A_8*geo.s_6_8);
s10_8= skew(A_8*geo.s_10_8);
s5_7 = skew(A_7*geo.s_5_7);
s9_7 = skew(A_7*geo.s_9_7);
s7_9 = skew(A_9*geo.s_7_9);
s8_10= skew(A_10*geo.s_8_10);

% Create current vectors of foot geometry
s_R_T = geo.s_R_T;
s_S_T = geo.s_S_T;
s_A_T = (Body(4).r(:,FrameNo) - Data.ZMP(:,FrameNo));

s_h_9  = A_9*geo.s_h_9;
s_h_10 = A_10*geo.s_h_10;
s_t_9  = A_9*geo.s_t_9;
s_t_10 = A_10*geo.s_t_10;
foot_width=geo.foot_width;

r_9=Body(9).r(:,FrameNo);
r_10=Body(10).r(:,FrameNo);
T=Data.ZMP(:,FrameNo);
```

```
% Get spline distribution factors
[fIncrease fDecrease]=CreateSplineFactors(Data.LengthDSP);

% Get current contact information
cR=Data.ContactR(FrameNo);
cL=Data.ContactL(FrameNo);

% Determine the point of contact: armR and armL from foot CoM to
% contact point, i.e. ZMP.
    % Right foot: Heel-contact / toe-contact?
    if s_R_T(1,FrameNo)>0 % Heel-contact
        if norm(s_R_T(:,FrameNo))>norm(s_h_9(1:2))
            armR=-unit(s_R_T(:,FrameNo))*norm(s_h_9(1:2));
        else
            armR=-s_R_T(:,FrameNo);
        end
    else % Toe-contact
        if norm(s_R_T(:,FrameNo))>norm(s_t_9(1:2))
            armR=-unit(s_R_T(:,FrameNo))*norm(s_t_9(1:2));
        else
            armR=-s_R_T(:,FrameNo);
        end
    end

    % Left foot: Heel-contact / toe-contact?
    if s_S_T(1,FrameNo)>0 % Heel-contact
        if norm(s_S_T(:,FrameNo))>norm(s_h_10(1:2))
            armL=-unit(s_S_T(:,FrameNo))*norm(s_h_10(1:2));
        else
            armL=-s_S_T(:,FrameNo);
        end
    else % Toe-contact
        if norm(s_S_T(:,FrameNo))>norm(s_t_10(1:2))
            armL=-unit(s_S_T(:,FrameNo))*norm(s_t_10(1:2));
        else
            armL=-s_S_T(:,FrameNo);
        end
    end

    % Adjust contact point to not exceed the foot width
    if abs(armR(2))>foot_width/2; armR(2)=(foot_width/2)*sign(armR(2));end;
    if abs(armL(2))>foot_width/2; armL(2)=(foot_width/2)*sign(armL(2));end;


% Determine current load case
if strcmp(Data.LC,'mean_walk') || strcmp(Data.LC,'curb') || strcmp(Data.↙
LC,'start_rightforce')

    % Determine whether in DSP or SSP:
    if cR==1 && cL==1   % Double support phase

        if strcmp(Data.LC,'mean_walk')
            % Define distribution factors fR and fL
            switch Data.DSPno(FrameNo)
                % Factors depending on trailing/leading leg
                case 1 %right foot front, left foot rear
```

```
                    DSPframe=FrameNo;
                    fL=fDecrease(DSPframe);
                    fR=fIncrease(DSPframe);
                case 2 %left foot front, right foot rear
                    DSPframe=Data.PhaseShiftR(1)-FrameNo;
                    fL=fDecrease(DSPframe);
                    fR=fIncrease(DSPframe);
            end

        elseif strcmp(Data.LC,'curb')
            % Define distribution factors fR and fL
            switch Data.DSPno(FrameNo)
                % Factors depending on trailing/leading leg
                case 1
                    if FrameNo<Data.PhaseShiftL(1)-Data.LengthDSP
                        fR=0.5;
                        fL=0.5;
                    else
                        [fIncrease fDecrease]=CreateSplineFactors(Data.LengthDSP);
                        DSPframe=FrameNo-(Data.PhaseShiftL(1)-Data.LengthDSP)+1;
                        fL=0.5*fDecrease(DSPframe);
                        fR=1-fL;%0.5*fIncrease(DSPframe);
                    end
                case 2
                    DSPframe=FrameNo - Data.PhaseShiftL(2)+1;
                    fL=fIncrease(DSPframe);
                    fR=fDecrease(DSPframe);
                case 3
                    % Create spline factors associated with special DSPlength
                    [fIncrease fDecrease]=CreateSplineFactors(Data.nFrames-Data.↙
PhaseShiftR(2)+1);
                    DSPframe=FrameNo - Data.PhaseShiftR(2)+1;
                    fL=fDecrease(DSPframe);
                    fR=fIncrease(DSPframe);
            end
        elseif strcmp(Data.LC,'start_rightforce')
            % Define distribution factors fR and fL
            switch Data.DSPno(FrameNo)
                % Factors depending on trailing/leading leg
                case 1 %right foot front, left foot rear
                    if FrameNo<=Data.PhaseShiftR(1)-Data.LengthDSP
                        fL=0.5;
                        fR=0.5;
                    else
                        DSPframe=FrameNo-(Data.PhaseShiftR(1)-Data.LengthDSP);
                        fR=0.5*fDecrease(DSPframe);
                        fL=1-fR;
                    end
                case 2 %left foot front, right foot rear
                    [fIncrease fDecrease]=CreateSplineFactors(Data.LengthDSP);
                    DSPframe=FrameNo-Data.PhaseShiftR(2)+1;
                    fL=fDecrease(DSPframe);
                    fR=fIncrease(DSPframe);
            end
        end
```

```
        RR=fR*RT;
        RS=fL*RT;

    elseif cR==1 && cL==0   % Single support phase, right leg
        RR=RT;
        RS=[0 0 0]';

    elseif cR==0 && cL==1   % Single support phase, left leg
        RS=RT;
        RR=[0 0 0]';
    end

elseif strcmp(Data.LC,'standref')
    % Standing on both feet - divide reactions equally.
    fR=0.5;
    fL=0.5;
    RR=fR*RT;
    RS=fL*RT;

elseif strcmp(Data.LC,'raise') || strcmp(Data.LC,'sit')

    if strcmp(Data.LC,'raise')
        end_sit_phase = 213;
        end_raise_phase = 242;

        if FrameNo<end_sit_phase
            fS = 0.9;
        elseif FrameNo>=end_sit_phase && FrameNo<end_raise_phase
            [fIncrease fDecrease]=CreateSplineFactors(end_raise_phase-end_sit_phase);
            DSPframe=FrameNo-end_sit_phase+1;
            fS = 0.9*fDecrease(DSPframe);
        else
            fS = 0.0;
        end
    else % LC=sit
        end_stand_phase = 500;
        end_sit_phase = 600;

        if FrameNo<end_stand_phase
            fS = 0.0;
        elseif FrameNo>=end_stand_phase && FrameNo<end_sit_phase
            [fIncrease fDecrease]=CreateSplineFactors(end_sit_phase-end_stand_phase);
            DSPframe=FrameNo-end_stand_phase+1;
            fS = 0.9*fIncrease(DSPframe);
        else
            fS = 0.9;
        end
    end

    R_seat = fS*RT;
    M_seat = skew(-s_A_T)*R_seat;


    fR=0.5;
    fL=0.5;
    RR=fR*(RT-R_seat);
```

```
    RS=fL*(RT-R_seat);
end


%%% SOLVE EQUILIBRIUM EQUATIONS %%%
% Left foot: 10
RK = RS - P10(1:3);
MK = s8_10*(-RK) + skew(armL)*RS - P10(4:6) + [ 0 0 fL*MT(3)]';

% Right foot: 9
RJ = RR - P9(1:3);
MJ = s7_9*(-RJ) + skew(armR)*RR - P9(4:6) + [0 0 fR*MT(3)]';

% Left shin: 8
RI = RK - P8(1:3);
MI = MK + s10_8*RK + s6_8*(-RI) - P8(4:6);

% Right shin: 7
RH = RJ - P7(1:3);
MH = MJ + s9_7*RJ + s5_7*(-RH) - P7(4:6);

% Left thigh: 6
RE = RI - P6(1:3);
ME = MI + s8_6*RI + s4_6*(-RE)-P6(4:6);

% Right thigh: 5
RD = RH - P5(1:3);
MD = MH + s7_5*RH + s4_5*(-RD) - P5(4:6);

% Pelvis: 4
RAP = RD + RE - P4(1:3) + R_seat;
MAP = MD + ME + s5_4*RD + s6_4*RE + s1_4*(-RAP) - P4(4:6) + M_seat;
%R_seat and M_seat are used only in LC's sit and raise.

% Left arm: 3
RC = P3(1:3);
MC = -s1_3*RC + P3(4:6);

% Right arm: 2
RB = P2(1:3);
MB = -s1_2*RB + P2(4:6);

% Torso: 1
RAT = RB + RC + P1(1:3);
MAT = MB + MC -s4_1*RAT - s2_1*(-RB) - s3_1*(-RC) + P1(4:6);

% Return results:
Reactions(:,FrameNo) = ...
[RAT; MAT; RB; MB; RC; MC; RD; MD; RE; ME; RH; MH; RI; MI; RJ; MJ; RK; MK; RR; MR; RS;↙
MS; RT; MT];

% Return further DataOutput for plotting
DataOut.armL(:,FrameNo)=armL;
DataOut.armR(:,FrameNo)=armR;
DataOut.RR(:,FrameNo)=RR;
DataOut.RS(:,FrameNo)=RS;
```

```
DataOut.RT(:,FrameNo)=RT;
DataOut.MT(:,FrameNo)=MT;
DataOut.fL(:,FrameNo)=fL;
DataOut.fR(:,FrameNo)=fR;
DataOut.MAP(:,FrameNo)=MAT;
DataOut.RAP(:,FrameNo)=RAT;
DataOut.R_seat(:,FrameNo)=R_seat;
DataOut.M_seat(:,FrameNo)=M_seat;

end
```

```
% Gear struct for each joint

% Data are used in: (CalcEqLoads and WaveGeneratorBearing)

% Average gear torque [Nm] (New inertia data)
G(1).T_av  = 3.05;
G(3).T_av  = 14.38;
G(5).T_av  = 6.79;
G(7).T_av  = 0.82;
G(19).T_av = 32.52;
G(21).T_av = 14.92;
G(23).T_av = 9.82;
G(25).T_av = 56.72;
G(27).T_av = 49.76;
G(29).T_av = 32.51;

% Rated torque [Nm]
G(1).T_n  = 24;
G(3).T_n  = 24;
G(5).T_n  = 7.8;
G(7).T_n  = 7.8;
G(19).T_n = 40;
G(21).T_n = 7.8;
G(23).T_n = 24;
G(25).T_n = 40;
G(27).T_n = 40;
G(29).T_n = 24;

% Limit for momentary peak torque [Nm]
G(1).T_M  = 110;
G(3).T_M  = 86;
G(5).T_M  = 54;
G(7).T_M  = 54;
G(19).T_M = 147;
G(21).T_M = 54;
G(23).T_M = 110;
G(25).T_M = 147;
G(27).T_M = 147;
G(29).T_M = 110;

% Max torque present in gear [Nm] (New inertia data)
G(1).T_max  = 6;
G(3).T_max  = 23;
G(5).T_max  = 12;
G(7).T_max  = 2;
G(19).T_max = 65;
G(21).T_max = 36;
G(23).T_max = 19;
G(25).T_max = 101;
G(27).T_max = 130;
G(29).T_max = 70;

% Pulley ratio in each joint
G(1).i  = 48/16;
G(3).i  = 60/20;
G(5).i  = 60/20;
```

```
G(7).i  = 20/18;
G(19).i = 36/18;
G(21).i = 44/22;
G(23).i = 40/16;
G(25).i = 72/30;
G(27).i = 60/28;
G(29).i = 32/24;

% Total gearing ratio in each joint
G(1).i_tot  = G(1).i*100;
G(3).i_tot  = G(3).i*120;
G(5).i_tot  = G(5).i*100;
G(7).i_tot  = G(7).i*100;
G(19).i_tot = G(19).i*160;
G(21).i_tot = G(21).i*100;
G(23).i_tot = G(23).i*100;
G(25).i_tot = G(25).i*100;
G(27).i_tot = G(27).i*120;
G(29).i_tot = G(29).i*100;

% Average input speed [rpm]
G(1).n_in_av  = 3408/G(1).i;
G(3).n_in_av  = 819/G(3).i;
G(5).n_in_av  = 1382/G(5).i;
G(7).n_in_av  = 503/G(7).i;
G(19).n_in_av = 1253/G(19).i;
G(21).n_in_av = 698/G(21).i;
G(23).n_in_av = 1457/G(23).i;
G(25).n_in_av = 2342/G(25).i;
G(27).n_in_av = 2125/G(27).i;
G(29).n_in_av = 2250/G(29).i;

% Tilting moment on gears exposed to a tilting moment
G(1).M_t  = sqrt(12^2+23^2);
G(3).M_t  = 0;
G(5).M_t  = 0;
G(7).M_t  = sqrt(1^2+1^2);
G(19).M_t = 0;
G(21).M_t = 0;
G(23).M_t = sqrt(102^2+131^2);
G(25).M_t = 0;
G(27).M_t = 0;
G(29).M_t = 0;

% Pitch diameter output axle gear [m]
G(1).d_p  = 0.046;
G(3).d_p  = 0.057;
G(5).d_p  = 0.057;
G(7).d_p  = 0.019;
G(19).d_p = 0.034;
G(21).d_p = 0.042;
G(23).d_p = 0.038;
G(25).d_p = 0.069;
G(27).d_p = 0.057;
G(29).d_p = 0.051;
```

```
% Number of ocilations per minute
G(1).no  = 60;
G(3).no  = 60;
G(5).no  = 60;
G(7).no  = 60;
G(19).no = 120;
G(21).no = 120;
G(23).no = 60;
G(25).no = 120;
G(27).no = 60;
G(29).no = 60;

% Ocilation angle for the gear output shaft
G(1).Oz  = 40;
G(3).Oz  = 6;
G(5).Oz  = 10;
G(7).Oz  = 16;
G(19).Oz = 21;
G(21).Oz = 8;
G(23).Oz = 21;
G(25).Oz = 14;
G(27).Oz = 37;
G(29).Oz = 64;

% Dynamic load rating [N]
G(1).C  = 11500;
G(3).C  = 11500;
G(5).C  = 8500;
G(7).C  = 8500;
G(19).C = 24200;
G(21).C = 8500;
G(23).C = 11500;
G(25).C = 24200;
G(27).C = 24200;
G(29).C = 11500;

% Pitch diameter on mototpulley [m]
mot(1).dp  = 15.28/1000;
mot(3).dp  = 18.34/1000;
mot(5).dp  = 18.34/1000;
mot(7).dp  = 17.19/1000;
mot(19).dp = 17.19/1000;
mot(21).dp = 21.01/1000;
mot(23).dp = 15.28/1000;
mot(25).dp = 28.65/1000;
mot(27).dp = 25.98/1000;
mot(29).dp = 38.20/1000;

% Pitc belt angle [deg]
mot(19).Psi1 = 12 ;%4;
mot(19).Psi2 = 92 ;%25;
mot(25).Psi1 = 23 ;%26.15;
mot(25).Psi2 = 106 ;%23.9;
mot(29).Psi1 = 92 ;%6;
mot(29).Psi2 = 78 ;%12;
```

```
% Max motor moment [Nm]
mot(1).Mmax = 0.3528;
mot(3).Mmax = 0.736;
mot(5).Mmax = 0.386;
mot(7).Mmax = 0.3528;
mot(19).Mmax = 0.736;
mot(21).Mmax = 0.386;
mot(23).Mmax = 0.3528;
mot(25).Mmax = 0.736;
mot(27).Mmax = 0.736;
mot(29).Mmax = 0.736;

% belt lengths in joint [m]
J(1).L1 = 0.102;
J(3).L1 = 0.74;
J(5).L1 = 0.130;
J(7).L1 = 0.49;
J(19).L1 = 0.125;
J(19).L2 = 0.088;
J(19).L3 = 0.066;
J(21).L1 = 0.60;
J(23).L1 = 0.134;
J(25).L1 = 0.135;
J(25).L2 = 0.142;
J(25).L3 = 0.07;
J(27).L1 = 0.110;
J(29).L1 = 0.140;
J(29).L2 = 0.101;
J(29).L3 = 0.057;
```

```
% Determine pre-tension forces in statica undeterminate joints
clc
GearData

Fi = 30; % Pre-tension force
AE = 100000; % Belt stiffness arbitrary determined, vanishes in the equation

% Joint number to analyse
Nr = 19;

if 1==1 %Calculates joint with 2 motors
    % Stiffness
    k1 = AE/J(Nr).L1;
    k2 = AE/J(Nr).L2;
    k3 = AE/J(Nr).L3;

    % pitch radius
    ra = mot(Nr).dp/2;
    rb = ra;
    rc = G(Nr).d_p/2;

    Mm = mot(Nr).Mmax;

       %[F1      F2      F3   phib    phic   Ma]
    A = [1       0       0    -k1*rb    0     0;...
         0       1       0      0     k2*rc   0;...
         0       0       1     k3*rb -k3*rc   0;...
         ra     -ra      0      0       0    -1;...
         rb      0      -rb     0       0     0;...
         0      -rc      rc     0       0     0];

     b = [Fi Fi Fi 0 Mm Mm]';

     % Solve equation system
     c = A\b;

     fprintf(' F1    = %2.2f  [N]\n',c(1,1))
     fprintf(' F2    = %2.2f  [N]\n',c(2,1))
     fprintf(' F3    = %2.2f  [N]\n',c(3,1))
     fprintf(' psi_b = %2.4f [rad]\n',c(4,1))
     fprintf(' psi_c = %2.4f [rad]\n',c(5,1))
     fprintf(' M_a   = %2.2f  [Nm]\n\n',c(6,1))
end

if 1==0%Calculates joint with 1 motor
   k = AE/J(Nr).L1;

   ra = mot(Nr).dp;
   rb = G(Nr).d_p;

   Mm = mot(Nr).Mmax;
       %F1   F2    Mb   phia]
   A = [1    0     0    -k*ra;...
        0    1     0     k*ra;...
        rb  -rb    1      0  ;...
        ra  -ra    0      0  ];
```

```
   b = [Fi Fi 0 Mm]';

   % Solve equations
   c = A\b;

   fprintf(' F1   = %2.2f  [N]\n',c(1,1))
   fprintf(' F2   = %2.2f  [N]\n',c(2,1))
   fprintf(' Mb   = %2.2f  [Nm]\n',c(3,1))
   fprintf(' psia = %2.4f [rad]\n\n',c(4,1))

end
```

```
%%%%% DCmotorGearUnit_main %%%%%
clc; clear all; close all;
load GearMotorData/MGData
load DCMotorData/DCMotor

% Choose: CPU-M or CPU-S, Joint number, Axis
CPUM = 0; CPUS = 1;
JointNo =6; % JointNo = 1:9
Axis = 2;   % Axis: 1=Xi, 2=Eta, 3=Zeta

% Choice input and output pullery. Input; motor/gear, Output; gear/joint
% 1=18 teeth, 2=21,3=26, 4=48, 5=57, 6=63. 1:6=2MR
% 7=18, 8=22, 9=30, 10=34, 11=38, 12=41, 13=44, 14=56, 15=57,16=62,17=72,18=20,19=26,↙
20=24,21=32,22=16
% no one=0,
PulleyInput = 20;
PulleyOutput= 21;

% Choose if belt drive is before or after the HD gear;
% TimB= before HD gear, TimA= After HD gear, yes=1 no=0
TimA = 0;
TimB = 1;

% Plot motor trajectorie, if yes=1,no=0
PlotMotor=1;

% Choose load case; 1=mean walk, 2=curb, 3=raise, 4=sit, 5=stand, 6=start
LC=1;

%%%%%%% safety constant = SC
SC=1.9;

%%%%%%% Factor of overload; NomTorque*nominal torque for motor. Allow that
%%%%%%% the rms value crosses the nominal torque for the DC motor.
NomTorque=1.5;

% run
CalcJointTor;   % Motor torque
REMaxonData;    % DC motor data
HDGearData;     % CPU gear unit data
TimingBeltPulleyData;
CalcGearEff;    % Calculate the efficiency of the gear unit
CalcServoConstant; % find DC motor suitable for the joint

% Servomotor and gear combination
ServoGear = zeros(noGear/2,noRatio,noServo);
for sNo = 1:noServo
    for Unit = 1:noGear/2
        for Ratio= 1:noRatio
        SG = KnMcBin(Unit,Ratio,sNo)*GCbin(Unit,Ratio);
        if SG==1;
            ServoGear(Unit,Ratio,sNo) = 1;
        end
        end
    end
end
```

```
if max(max(ServoGear))==0
    fprintf(1,'\n********** No match ********** \n');
    else
    % The seleted DC motors are checked if they can deliver the required acceleration
    Pin=PulleyInput; Pout=PulleyOutput;
    [fServo fUnit fRatio]= CalcInertia(Motor,Pulley,JointNo,Axis,Gear,servo,ServoGear,↙
Pin,Pout,CPUM,TimA,TimB,MMax,MB,FrameB,NomTorque);

    %%% Servo motor and gear weight
    if max(max(fServo))==0
            fprintf(1,'\n********** No match ********** \n');
    else
        for i=1:length(fServo)
           if fServo(i)==0
               SGW(:,i)=1000;
           else
            SW=servo(fServo(i)).EC(15);
            GW=Gear(fUnit(i)+5).HD(2); % s-unit
                if TimA+TimB>0
                SGW(:,i)=SW+GW+Pulley(PulleyInput).m+Pulley(PulleyOutput).m;
                else
                SGW(:,i)=SW+GW;
                end
           end
        end
        % Gear and motor combination with respect to min. weight
        [weight place]=min(SGW);
%%%%% Here other DC motors and gear units combinations can be selected manual. Change↙
place!!
        %place=4;
        UnitNo=Gear(fUnit(place)).HD(30);
        RatioNo=Gear(3).HD(2+fRatio(place));
        ServoNo=fServo(place);
        ServoPower=servo(ServoNo).EC(1);
        ServoOrder=servo(ServoNo).EC(16);
        ServoVolt=servo(ServoNo).EC(2);

        % Get the real ratio no,unit no and server motor no.
        if servo(fServo(place)).EC(17)==1
             ServoSign='EC';
        elseif servo(fServo(place)).EC(17)==2
            ServoSign='RE';
        else
            ServoSign='2xRE';
        end
        fprintf('\nRequired Harmonic Drive Gear and Servo Motor:\n\n')
        fprintf('Weight of motor and gear system[kg] = %6.2f\n',weight);
        fprintf('Harmonic Drive unit=                  %2i\n',UnitNo);
        fprintf('Harmonic Drive ratio=                 %2i\n',RatioNo);
        if (PulleyInput+PulleyOutput>0) && (TimB+TimA==1)
        fprintf('Input pulley[Teeth] =                 %2i\n',Pulley(PulleyInput).↙
Te);
        fprintf('Output pulley[Teeth]=                 %2i\n',Pulley(PulleyOutput).↙
Te);
        end
        fprintf('Assigned power rate for motor =       %2i\n',ServoPower);
```

```
        fprintf('Nominal voltage for motor =             %2i\n',ServoVolt);
        fprintf('Order number for motor=     %s motor    %2i\n\n',ServoSign,↙
ServoOrder);
        % Run Plot
        Servo=fServo(place); Unit=fUnit(place); Ratio=fRatio(place);
        [JV,JT]=PlotMotorGear(Servo,Unit,Ratio,Axis,EtaGear,Pulley,OMB,OnB,Motor,servo,↙
Gear,TimA,TimB,JointNo,LC,PlotMotor,Tspan,Pin,Pout,SC);

        %%%%%%%%%%%%%%% Save inetia and mass of motor, pully and gear %%%%%%%%%%%%%
        % If save inetia and mass data JMdata=1 else JMdata=0
        JMdata= 1;
        [MGData,R]=MGPData(JMdata,JointNo,Axis,Pin,Pout,servo,Gear,Pulley,Unit,Ratio,↙
Servo,JT,EtaGear,MGData); % save data

        %%%%%%%%%%%%%%% save Motor Data %%%%%%%%%%%%%%%%%%%%%%%%%%
        if CPUS==1
            Utype='S';
        elseif CPUM==1
            Utype='M';
        end
        DCMotorData(ServoPower,ServoVolt,ServoSign,ServoOrder,R,Pin,Pout,UnitNo,↙
RatioNo,Utype,DCMotor);
        % Calculate electrical power
        [Motor,Overshoot,Trms,Tmax,Smean,Amax,Mrms,Arms]=PowerEl(servo,ServoNo,JT,JV,R,↙
Motor,Data,SC,NomTorque);
        fprintf('Max torque =   %6.2f\n',Tmax);
        fprintf('Max current =   %6.2f\n',Amax);
        fprintf('current rms =   %6.2f\n',Arms);
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%% Battery Properties %%%%%%%%%%%%%%%%%%%%%%%%%%

clc; clear all; close all;
load DCmotorData/Power %             Motor Power
load DCmotorData/DCMotor%            Motordata
load ..\res_mean_walk.mat%           Loadcase
load DCmotorData/Joint
TimingBeltPulleyData;
ServoDataRE48;

Tspan   = Data.Tspan;
Time=Tspan(end); %[s]

Itotal=zeros(1,284);
PlossMean=0;
PmecMean=0;
Pmech=0;


for i=1:33
Joint(i).PlossMean= Power(1,i).PlossMean;
Joint(i).PlossMax=  Power(1,i).PlossMax;
Joint(i).PmecMax=   Power(1,i).PmecMax;
Joint(i).PmecMean=   Power(1,i).PmecMean;
Joint(i).IrmsMotor= Power(1,i).IrmsMotor;
Joint(i).ImaxMotor= Power(1,i).ImaxMotor;
Joint(i).IbatMax=   Power(1,i).IbatMax;
Joint(i).I=         Power(1,i).Imotor;
Joint(i).PmecW=     Power(1,i).PmecW;
Joint(i).PlossW=    Power(1,i).PlossW;
Joint(i).Pjoint=    Power(1,i).Pjoint;
Joint(i).MotorNo=   DCMotor(1,i).No;
Joint(i).Type=      DCMotor(1,i).Type;
Joint(i).Power=     DCMotor(1,i).Power;
Joint(i).Volt=      DCMotor(1,i).Volt;
Joint(i).Pin=       DCMotor(1,i).Pin;
Joint(i).Pout=      DCMotor(1,i).Pout;
Joint(i).MV=        Power(1,i).JV; % Motor speed [rpm]
Joint(i).MT=        Power(1,i).JT; % Motor torque
Joint(i).SC=        Power(1,i).SC;
Joint(i).Ratio=     DCMotor(1,i).Ratio;
Joint(i).UnitNo=     DCMotor(1,i).UnitNo;
Joint(i).UnitType=     DCMotor(1,i).UnitType;
Joint(i).Arms= Power(1,i).Arms;
Joint(i).Trms= Power(1,i).Mrms;

B=isempty(Joint(i).I);
    if B==0
        Itotal=Itotal+Joint(i).I;
        Pmech=Pmech+Joint(i).PmecW;
        PlossMean=PlossMean+Joint(i).PlossMean;
        PmecMean=PmecMean+Joint(i).PmecMean;
    end
end

Ubat=Joint(1).Volt;  %%%%!!!!!! nominal volt for motor at revolute joint 1
```

```
ImaxBat=max(Itotal);        % [A]
Ptotal=PmecMean+PlossMean;  % [W]
IbatMean=Ptotal/Ubat;
P10min=Ptotal/Time*10;      %[W*10min]
PmechMax=max(Pmech);

for i=1:33
    for k=1:284
        if Power(i).SC>0
        PowerMotor(i).Max(1,k)=Joint(i).MV(1,k)*Joint(i).MT(1,k)*pi/30;
        MotorJoint=Joint(i).MV(1,k)*Joint(i).MT(1,k);
        [MaxTorque,Place]=max(abs(Joint(i).MT));
        Joint(i).MotorMax=MaxTorque;
        Joint(i).MotorMaxPlace=Place;
        end
    end
end


BatProp=1;
if BatProp==1

 fprintf('Max current for battery                %6.1f [A]\n',ImaxBat);
 fprintf('Mean current for battery               %6.1f [A]\n',IbatMean);
 fprintf('Mean effect used for one step          %6.1f [W]\n',Ptotal);
 fprintf('Effect used for 10 min walk            %6.1f [W]\n',P10min);
 fprintf('Mean mechanical power(one step)        %6.1f [W]\n',PmecMean);
 fprintf('Max mechanical power                   %6.1f [W]\n\n',PmechMax);
 %fprintf('Assigned power for AAU-BOT             %6.1f [W]\n',Ptotal/Time);
 end
 % JointNo; Joint,Axis: 1,x=5 1,y=3 1,z=1  2/3,y=7/9 4/5,x= 25/13 4/5,y=
 % 5 3 1 7 25 27 23 29 21 19
 JointNo = [5 3 1 7 25 27 23 29 21 19];

% plot power loss: PlotPower=1
PlotPower=0;
 for PlotNo=1:length(JointNo)
    i = JointNo(PlotNo);
         A=isempty(Joint(i).I);
        if A==0
         if PlotPower==1
             figure
             plot (Tspan, Joint(i).PmecW,':r', Tspan, Joint(i).PlossW,'--k',Tspan,↙
Joint(i).I*5,'m' )
             xlabel('Time [s]')
             ylabel('Power [W]')
             title(['Power and current used, JointNo: ', num2str(i)])
             legend('Mechanical Power', 'Power loss motor/control', 'Motor↙
current*scale5')
             grid on
         end
         fprintf('Motor properties,         Joint No; %2i  \n\n',i);
         fprintf('Max Power motor;          [W]            %6.1f\n',max(abs(PowerMotor↙
(i).Max)));%Joint(i).PmecMax);
         fprintf('Mean Power motor;         [W]            %6.1f\n\n',Joint(i).↙
Trms*mean(abs(Joint(i).MV))*pi/30);
```

```
        fprintf('DC motor No.;                              %6.1i\n',DCMotor(i).No);
        fprintf('DC motor type.;                              %s\n',DCMotor(i).Type);
        fprintf('Nominal volt for motor;   [v]              %6.1i\n',DCMotor(i).Volt);
        fprintf('Assigned power for motor; [W]              %6.1i\n',DCMotor(i).Power);
        fprintf('Motor power at max torque;[W]              %6.1f\n',abs(PowerMotor(i).↙
Max(1,Joint(i).MotorMaxPlace)));
        fprintf('Motor speed at max torque;[RPM]            %6.1f\n',abs(Joint(i).MV↙
(Joint(i).MotorMaxPlace)));
        fprintf('Max motor torque;         [Nm]             %6.1f\n', max(abs(Joint(i).↙
MT)));
        fprintf('Mean motor speed;         [RPM]            %6.1f\n',mean(abs(Joint(i).↙
MV)));
        fprintf('Max motor speed;          [RPM]            %6.1f\n',max(abs(Joint(i).↙
MV)));

        fprintf('Pulley (output:input);[Ratio]    %6.1i :%6.1i\n',Pulley(DCMotor(i).↙
Pout).Te,Pulley(DCMotor(i).Pin).Te);
        fprintf('Safety constant;                           %6.1f\n\n',Joint(i).SC);

       end
end


% Max torque and speed for motor and gear
%[Max]=MaxTorqueSpeed(Joint, DCMotor, Power);
 JointNo2 = [];
for PlotNo2=1:length(JointNo2)
    i = JointNo2(PlotNo2);
    fprintf('Motor and Gear properties,         Joint No; %2i  \n\n',i);
    fprintf('Unit No.;                                    %6.1i\n',Joint(i).UnitNo);
    fprintf('Unit Ratio;                [Ratio]           %6.1i\n',Joint(i).Ratio);
    fprintf('Pulley (output:input);     [Ratio]  %6.1i:%6.1i\n',Pulley(DCMotor(i).↙
Pout).Te,Pulley(DCMotor(i).Pin).Te);
    fprintf('DC motor type.;                                %s\n',DCMotor(i).Type);
    fprintf('DC motor No.;                                %6.1i\n',DCMotor(i).No);
    fprintf('Assigned power for motor;  [W]               %6.1i\n',DCMotor(i).Power);
    fprintf('Properties for normal walk       \n\n');
    if strmatch('2xRE',DCMotor(i).Type)==1
    fprintf('Max motor torque;          [Nm](two motors)%6.3f\n',max(abs(Joint(i).↙
MT)));
    fprintf('Rms motor torque;          [Nm](two motors)%6.3f\n',Max(i).RmsTorque);
    else
    fprintf('Max motor torque;          [Nm]            %6.3f\n',max(abs(Joint(i).↙
MT)));
    fprintf('Rms motor torque;          [Nm]            %6.3f\n',Max(i).RmsTorque);
    fprintf('Pulley (output:input);    [Ratio]    %6.1i:%6.1i\n',Pulley(DCMotor(i).↙
Pout).Te,Pulley(DCMotor(i).Pin).Te);
    end

    fprintf('Max motor speed;           [RPM]           %6.0f\n',max(abs(Joint(i).↙
MV)));
    fprintf('Mean motor speed;          [RPM]           %6.1f\n',mean(abs(Joint(i).↙
MV)));
    fprintf('Limet for motor         \n\n');
    fprintf('Max permittet motor speed;  [RPM]          %6.1i\n',Max(i).GearSpeed);
    fprintf('Avg.permittet motor speed;  [RPM]          %6.1i\n',Max(i).GearSpeedAv);
```

```
    if strmatch('2xRE',DCMotor(i).Type)==1
    fprintf('Max permittet motor current;[A] (one motor) %6.1f\n',Max(i).MotorCurrent);
    fprintf('Nominal current for motor;  [A] (one motor) %6.1f\n',Max(i).NomCurrent);
    fprintf('Max permittet motor torque; [Nm](two motors)%6.3f\n',Max(i).MotorTorque);
    fprintf('Nominal torque for motor;   [Nm](two motors)%6.3f\n\n\n',Max(i).↙
NomTorque);
    else
    fprintf('Max permittet motor current;[A]             %6.1f\n',Max(i).MotorCurrent);
    fprintf('Nominal current for motor;  [A]             %6.1f\n',Max(i).NomCurrent);
    fprintf('Max permittet motor torque; [Nm]            %6.3f\n',Max(i).MotorTorque);
    fprintf('Nominal torque for motor;   [Nm]            %6.3f\n\n\n',Max(i).↙
NomTorque);
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%% Efficiency of Gear unit %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% initialize constants
Unit=0; eta50=0; eta80=0; eta100=0; eta120=0; eta160=0;
TimRatioT=0; TimRatioS=0;
Ms = Motor(JointNo).Mean(Axis); % Mean.joint speed [rpm], Axis=local CS

% Belt conection
if TimB==1
    TimRatioT = Pulley(PulleyOutput).Te/Pulley(PulleyInput).Te*TimEff; % timing ratio↙
torque
    TimRatioS = Pulley(PulleyOutput).Te/Pulley(PulleyInput).Te; % timing ratio speed
else
    TimRatioT = 1;
    TimRatioS = 1;
end

% Pulley ratio, choice output/input
if PulleyOutput==0
    TimRatioT = 1;
    TimRatioS = 1;
elseif PulleyInput==0
    TimRatioT = 1;
    TimRatioS = 1;
end

for Unit=1:noGear/2 % Unit: 1=14, 2=17 3=20 4=25 5=32
    if CPUM == 1
        GR = Gear(Unit).HD(3:7);
    elseif CPUS==1
        GR = Gear(5+Unit).HD(3:7);
    end

% mean motor speed with respect to local CS at joint [rpm]
nMotor= TimRatioS*[ Ms*GR(1) Ms*GR(2) Ms*GR(3) Ms*GR(4) Ms*GR(5)]; % nMotor*timing-belt↙
ratio

% Calculation of effiency is based on 'Engimeering Data P.432'
% [Engineering Data For Harmonic Drive Gears].
% The work temperature is approximated 20degC and the eta/speed
% chart is approximated linear.

    if (Unit==1)  && (CPUM==1) % Efficiency of CPU-M 14

       eta50c = -4*10^-5*nMotor(1)+0.7333; % eta from effiency chart
       VG     = Motor(JointNo).Mav(Axis)/TimRatioT*1/Gear(Unit).HD(25); % torque factor↙
for a gear
       kG     = 0.2695*reallog(VG)+1.0656; % K factor is gear dependent
       if (kG<1) && (kG>0.1)
              eta50  = eta50c*kG;   % Effiency at a ratio for a given unit
       elseif kG>=1                 % Upper boundary of graph
              eta50 = eta50c;
       elseif kG<=0.1               % Lower boundary of graph
              eta50 = eta50c*0.1;
       end
```

```
    eta80c = -4*10^-5*nMotor(2)+0.7333;
    VG     = Motor(JointNo).Mav(Axis)/TimRatioT*1/Gear(Unit).HD(26);
    kG     = 0.2695*reallog(VG)+1.0656;
    if (kG<1) && (kG>0.1)
           eta80  = eta80c*kG;   % Effiency at a ratio for a given unit
    elseif kG>=1                 % Upper boundary of graph
           eta80 = eta80c;
    elseif kG<=0.1               % Lower boundary of graph
           eta80 = eta80c*0.1;
    end;

    eta100c  = -5*10^-5*nMotor(3)+0.7583;
    VG       = Motor(JointNo).Mav(Axis)/TimRatioT*1/Gear(Unit).HD(27);
    kG       = 0.2695*reallog(VG)+1.0656;
    if (kG<1) && (kG>0.1)
           eta100  = eta100c*kG; % Effiency at a ratio for a given unit
    elseif kG>=1                 % Upper boundary of graph
           eta100 = eta100c;
    elseif kG<=0.1               % Lower boundary of graph
           eta100 = eta100c*0.1;
    end

    EtaGear(1).HD = [eta50 eta80 eta100 0 0];
end

if (Unit>1) && (CPUM==1) %      Efficiency of CPU-M 17, 20, 25 and 32

    eta50c   = -4*10^-5*nMotor(1)+0.8; % eta from effiency chart
    VG       = Motor(JointNo).Mav(Axis)/TimRatioT*1/Gear(Unit).HD(25);
    kG       = 0.2695*reallog(VG)+1.0656;
    if (kG<1) && (kG>0.1)
           eta50  = eta50c*kG;   % Effiency at a ratio for a given unit
    elseif kG>=1                 % Upper boundary of graph
           eta50 = eta50c;
    elseif kG<=0.1               % Lower boundary of graph
           eta50 = eta50c*0.1;
    end

    eta80c   = -6*10^-5*nMotor(2)+0.8467;
    VG       = Motor(JointNo).Mav(Axis)/TimRatioT*1/Gear(Unit).HD(26);
    kG       = 0.2695*reallog(VG)+1.0656;
    if (kG<1) && (kG>0.1)
           eta80  = eta80c*kG;   % Effiency at a ratio for a given unit
    elseif kG>=1                 % Upper boundary of graph
           eta80 = eta80c;
    elseif kG<=0.1               % Lower boundary of graph
           eta80 = eta80c*0.1;
    end

    eta100c  = -6*10^-5*nMotor(3)+0.8467;
    VG       = Motor(JointNo).Mav(Axis)/TimRatioT*1/Gear(Unit).HD(27);
    kG       = 0.2695*reallog(VG)+1.0656;
    if (kG<1) && (kG>0.1)
           eta100  = eta100c*kG; % Effiency at a ratio for a given unit
    elseif kG>=1                 % Upper boundary of graph
           eta100 = eta100c;
```

```
        elseif kG<=0.1              % Lower boundary of graph
               eta100 = eta100c*0.1;
        end

        eta120c  = -5*10^-5*nMotor(4)+0.8217;
        VG       = Motor(JointNo).Mav(Axis)/TimRatioT*1/Gear(Unit).HD(28);
        kG       = 0.2695*reallog(VG)+1.0656;
        if (kG<1) && (kG>0.1)
               eta120  = eta120c*kG; % Effiency at a ratio for a given unit
        elseif kG>=1                % Upper boundary of graph
               eta120 = eta120c;
        elseif kG<=0.1              % Lower boundary of graph
               eta120 = eta120c*0.1;
        end

        if nMotor(5)==0
            eta160=0;
        else
            eta160c  = -6*10^-5*nMotor(5)+0.8035;
            VG       = Motor(JointNo).Mav(Axis)/TimRatioT*1/Gear(Unit).HD(28);
            kG       = 0.2695*reallog(VG)+1.0656;
            if (kG<1) && (kG>0.1)
                   eta160  = eta160c*kG; % Effiency at a ratio for a given unit
            elseif kG>=1                % Upper boundary of graph
                   eta160 = eta160c;
            elseif kG<=0.1              % Lower boundary of graph
                   eta160 = eta160c*0.1;
            end
        end
        EtaGear(Unit).HD = [eta50 eta80 eta100 eta120 eta160];
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Calculation of the efficiency of the HD S-Unit. The calculation     %
%   procedure at P.152 at Harmonic Drive AG Manual for Units is used.    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if CPUS==1                  % Efficiency of CPU-S 14-32

               [ne]=EtaE(Unit);  % Get correction value ne
                eta50c = -6E-05*(nMotor(1))+0.8183; % eta from effiency chart (P.152)
                VG     = Motor(JointNo).Mav(Axis)/Gear(Unit+5).HD(25); % Torque factor↙
for gear
                kG     = 0.3206*reallog(VG)+1.0096; % K factor is gear dependent
                if (kG<1) && (kG>0.1)
                eta50  = (eta50c-ne(1).e)*kG;  % Effiency at a ratio for a given unit
                elseif kG>=1                   % Upper boundary of graph
                eta50 = eta50c-ne(1).e;
                elseif kG<=0.2                 % Lower boundary of graph
                eta50 = (eta50c-ne(1).e)*0.2;  % Correction value is added... ne().e
                end


                eta80c = -6E-05*(nMotor(2))+0.8183;
                VG     = Motor(JointNo).Mav(Axis)/Gear(Unit+5).HD(26);
                kG     = 0.3206*reallog(VG)+1.0096;
```

```
                if (kG<1) && (kG>0.1)
                eta80  = (eta80c-ne(2).e)*kG;  % Effiency at a ratio for a given unit
                elseif kG>=1                   % Upper boundary of graph
                eta80 = eta80c-ne(2).e;
                elseif kG<=0.2                 % Lower boundary of graph
                eta80 = (eta80c-ne(2).e)*0.2;
                end

                eta100c  = -6E-05*(nMotor(3))+0.8183;
                VG       = Motor(JointNo).Mav(Axis)/Gear(Unit+5).HD(27);
                kG       = 0.3206*reallog(VG)+1.0096;
                if (kG<1) && (kG>0.1)
                eta100  = (eta100c-ne(3).e)*kG;  % effiency at a ratio for a given unit
                elseif kG>=1                     % Upper boundary of graph
                eta100 = eta100c-ne(3).e;
                elseif kG<=0.2                   % Lower boundary of graph
                eta100 = (eta100c-ne(3).e)*0.2;
                end

                if nMotor(4)==0
                    eta120=0;
                else
                eta120c   = -6E-05*(nMotor(4))+0.8183;
                VG        = Motor(JointNo).Mav(Axis)/Gear(Unit+5).HD(28);
                kG        = 0.3206*reallog(VG)+1.0096;
                    if (kG<1) && (kG>0.1)
                    eta120  = (eta120c-ne(4).e)*kG;  % Effiency at a ratio for a given↙
unit
                    elseif kG>=1                     % Upper boundary of graph
                    eta120 = eta120c-ne(4).e;
                    elseif kG<=0.2                   % Lower boundary of graph
                    eta120 = (eta120c-ne(4).e)*0.2;
                    end
                end
                if nMotor(5)==0
                    eta160=0;
                else
                eta160c   = -0.1554*reallog(nMotor(5))+1.805;
                VG        = Motor(JointNo).Mav(Axis)/Gear(Unit+5).HD(28);
                kG        = 0.3206*reallog(VG)+1.0096;
                    if (kG<1) && (kG>0.1)
                    eta160  = (eta160c-ne(5).e)*kG;% Effiency at a ratio for a given↙
unit
                    elseif kG>=1                   % Upper boundary of graph
                    eta160 = eta160c-ne(5).e;
                    elseif kG<=0.2                 % Lower boundary of graph
                    eta160 = (eta160c-ne(5).e)*0.2;
                    end
                end
                EtaGear(Unit).HD = [eta50 eta80 eta100 eta120 eta160];


    end

end
```

```
%%%%%% Check if average torque (Tav) < limit for average torque (TA) %%%%%
if TimA==1          % if timing belt connection is used TimA=1
    KT=TimRatioT;
    KS=TimRatioS;
else
    KT=1;
    KS=1;
end

Tav = Motor(JointNo).Mav(Axis);
TAbin= zeros(noGear/2,noRatio);
for Unit=1:5       % If average torque is under repeated peak torque; Ta=1
    for Ratio=1:noRatio
        TA= Gear(Unit).HD(14+Ratio); % Get limit for repeated peak torque
        if TA >Tav/KT               % Check if avg. torque is under limit
            TAbin(Unit,Ratio)=1;
        end
    end
end

%%%%%%% Check if max. torque (Tmax)< Limit for max torque (TR); TR=1 %%%%%
Tmax = Motor(JointNo).TMax(Axis);
TRbin = zeros(noGear/2,noRatio);
for Unit=1:noGear/2
    for Ratio=1:noRatio
        TR = Gear(Unit).HD(19+Ratio); % Get limit for momentary peak torque
        if TR > Tmax/KT               % Check if max. torque is under limit
            TRbin(Unit,Ratio)=1;
        end
    end
end

% Load limit for gear
LLbin = zeros(noGear/2,noRatio);
for Unit=1:noGear/2
    for Ratio=1:noRatio
        LLbin(Unit,Ratio) = TAbin(Unit,Ratio)*TRbin(Unit,Ratio);
    end
end

% Check if average input speed(Ms) times gear < limit for average speed input
AISLbin = zeros(noGear/2,noRatio); % Average input speed limit
for Unit=1:noGear/2
    for Ratio=1:noRatio
        AIS = Gear(Unit).HD(8);    % Limit for average speed input
        AMs = nMotor(Unit);        % Average speed motor incl. gear ratio
        if AIS>AMs
            AISLbin(Unit,Ratio) = 1;
        end
    end
end

% Gear combination for motor
GCbin = zeros(noGear/2,noRatio);        % Suitable gear for motor
for Unit=1:noGear/2
    for Ratio=1:noRatio
```

```
        GCbin(Unit,Ratio)= LLbin(Unit,Ratio)*AISLbin(Unit,Ratio);
    end
end
```

```
function [fServo fUnit fRatio]= CalcInertia(Motor,Pulley,JointNo,Axis,Gear,servo,↙
ServoGear,Pin,Pout,CPUM,TimA,TimB,MMax,MB,FrameB,NomTorque)

% Check acceleration
alphaMean=mean(abs(Motor(JointNo).acc(Axis,:)));        % [rad/s^2]
[alphaMax FrameA]=max(abs(Motor(JointNo).acc(Axis,:))); %   [rad/s^2]

% Inertia at joint
JR=abs(Motor(JointNo).torque(Axis,FrameB))*1/alphaMax;  % [kgm^2]

%Find unit, ratio and servo number
[nRows nCols nDeep]=size(ServoGear);
[row col val]=find(ServoGear);
RowNo = row';
for i=1:length(col)
    Dep     = ceil(col(i)/nCols);
    Ratio   = col(i) - nCols*(Dep-1)*(Dep>1);
    Servo   = Dep;
    Unit    = RowNo(i);
    HDRatio = Gear(Unit).HD(2+Ratio);
    JS      = servo(Servo).J;               % Inertia of rotor
    if CPUM>0
            JHD = Gear(Unit).J;             % Inertia of HD unit
        else
            JHD = Gear(Unit+5).J;
    end
    if TimA+TimB>0
    JPin = Pulley(Pin).J;                   % Pulley inertia
    JPout= Pulley(Pout).J;
    PRatio = Pulley(Pout).Te/Pulley(Pin).Te; % Ratio of pulley combination
    end
    if (Pin+Pout>0) && (TimB==1)
            J = JS+JPin+(JPout+JHD)/PRatio^2+JR/(PRatio*HDRatio)^2; % The equivalent↙
inertia of pulley before HD
            JT = JS+JPin+(JPout+JHD)/PRatio^2; % Total inetia of motor and gear system
        elseif (Pin+Pout>0) && (TimA==1)
            J = JS+JHD+JPin/HDRatio^2+(JPout+JR)/(HDRatio*PRatio)^2; % Sum of inertia;↙
pullets after HD
            JT = JS+JHD+JPin/HDRatio^2+(JPout)/(HDRatio*PRatio)^2;
        else
            J = JS+JHD+JR/HDRatio^2; % Sum of inertia; motor, HD and robot
            JT = JS+JHD;
    end

    % Max motor acceleration
    Macc=servo(Servo).EC(4)/J;
    % Mean and max loss
    MLmean=J*alphaMean;
    MLmax=J*alphaMax;
    % Nominal motor torque > than MB + MLmean
    if servo(Servo).EC(9)*NomTorque <MB(Unit,Ratio)+MLmean
         Servo=0;
         Ratio=0;
         Unit =0;
    end
    % Stall torque > than Mmax+MLmax
```

```
    if Servo==0
         Servo=0;
    elseif servo(Servo).EC(4)<MMax(Unit,Ratio)+MLmax
         Servo=0;
         Ratio=0;
         Unit =0;
    end
    % max acc for motot> alphamax, assuming thar the voltage is constant
    if Servo==0
       Servo=0;
    elseif Macc<alphaMax
       Servo=0;
       Ratio=0;
       Unit =0;
    end
    % save feasible motors
    fServo(:,i)=Servo;
    fRatio(:,i)=Ratio;
    fUnit(:,i)=Unit;
end
```

```
% Selection of servomotor and driving gear
% it is only the left side of the test person/robot that is consideret,
% the right side is mirrored, with respect to the sagittal plan.

% Load loadcases and Data
if LC==1
    load LoadData\n_res_mean_walk
% if loadData=n_res_mean_walk: based on robot inetia and geometri
%             o_res_mean_walk: Based on test person
elseif LC==2
    load LoadData\n_res_curb.mat
elseif LC==3
    load LoadData\n_res_raise.mat
elseif LC==4
    load LoadData\n_res_sit.mat
elseif LC==5
    load LoadData\n_res_standref.mat
elseif LC==6
    load LoadData\n_res_start.mat
end

nFrames = Data.nFrames;
Tspan   = Data.Tspan;
nJoints = Data.nJoints;

% Mirror joints
if JointNo==5
   JointNo=4;
end
if JointNo==7
   JointNo=6;
end
if JointNo==9
   JointNo=8;
end

% Get joint moment, omega & omega dot
MA  = Joint(1).M; MB  = Joint(2).M; MC  = Joint(3).M; MD  = Joint(4).M;
ME  = Joint(5).M; MH  = Joint(6).M; MI  = Joint(7).M; MJ  = Joint(8).M;

% Rad/s to rpm
RadRpm = 30/pi;
OA  = Joint(1).omegaRel*RadRpm; OB  = Joint(2).omegaRel*RadRpm; % rmp
OC  = Joint(3).omegaRel*RadRpm; OD  = Joint(4).omegaRel*RadRpm; % rmp
OH  = Joint(6).omegaRel*RadRpm; OJ  = Joint(8).omegaRel*RadRpm; % rmp

OADot  = Joint(1).omegaRelDot; OBDot  = Joint(2).omegaRelDot; % rad/s^2
OCDot  = Joint(3).omegaRelDot; ODDot  = Joint(4).omegaRelDot; % rad/s^2
OHDot  = Joint(6).omegaRelDot; OJDot  = Joint(8).omegaRelDot; % rad/s^2

nav=0; t0=0; Ts=0; t=Tspan(end)/nFrames;
clear Motor FrameNo
for i=1:nFrames
FrameNo(i) = i;
% Get current A-matrices
    A_1=Body(1).A(:,:,i); A_2=Body(2).A(:,:,i); A_3=Body(3).A(:,:,i);
```

```
    A_4=Body(4).A(:,:,i); A_5=Body(5).A(:,:,i); A_6=Body(6).A(:,:,i);
    A_7=Body(7).A(:,:,i); A_8=Body(8).A(:,:,i); A_9=Body(9).A(:,:,i);
    A_10=Body(10).A(:,:,i);

% Local vectors ei_j: i= Body no. j=1:3, 1=xi 2=eta 3=zeta.
% A_k: k=1:9, k=segmant no
    e1_1  = A_1(:,1); e1_2  = A_1(:,2); e1_3  = A_1(:,3);
    e4_1  = A_4(:,1); e4_2  = A_4(:,2); e4_3  = A_4(:,3);
    e5_1  = A_5(:,1); e5_2  = A_5(:,2); e5_3  = A_5(:,3);
    e7_1  = A_7(:,1); e7_2  = A_7(:,2); e7_3  = A_7(:,3);
    e9_1  = A_9(:,1); e9_2  = A_9(:,2); e9_3  = A_9(:,3);

% Motor torque at joint. Ci_j, i= joint no, j= with respect to local vector

    CA_1 =  e1_1'*MA(:,i); % waist      %!!!moment with respect to torso     % [Nm]
    CA_2 =  e1_2'*MA(:,i); %            %------------ll---------             % [Nm]
    CA_3 =  e4_3'*MA(:,i); %            %------ ll -----     pelvis          % [Nm]
    CB_1 =  e1_1'*MB(:,i); % Right arm  %------ ll -----     torso           % [Nm]
    CB_2 =  e1_2'*MB(:,i); %            %------------ll---------             % [Nm]
    CB_3 =  e1_3'*MB(:,i); %            %------------ll---------             % [Nm]
    CC_1 =  e1_1'*MC(:,i); % Left arm   %------------ll---------             % [Nm]
    CC_2 =  e1_2'*MC(:,i); %            %------------ll---------             % [Nm]
    CC_3 =  e1_3'*MC(:,i); %            %------------ll---------             % [Nm]
    CD_1 =  e5_1'*MD(:,i); % Right hip  %------ ll -----     pelvis          % [Nm]
    CD_2 =  e5_2'*MD(:,i); %            %------ ll -----      thigh          % [Nm]
    CD_3 =  e4_3'*MD(:,i); %            %------------ll---------             % [Nm]
    CH_1 =  e5_1'*MH(:,i); % Right knee %-------ll------      thigh          % [Nm]
    CH_2 =  e5_2'*MH(:,i); %            %------------ll---------             % [Nm]
    CH_3 =  e5_3'*MH(:,i); %            %------------ll---------             % [Nm]
    CJ_1 =  e7_1'*MJ(:,i); % Right foot %-------ll------      shin           % [Nm]
    CJ_2 =  e9_2'*MJ(:,i); %            %-------ll------      foot           % [Nm]
    CJ_3 =  e9_3'*MJ(:,i); %            %------------ll---------             % [Nm]

% velocitie at each joint with respect to local coordinate system

    OA_1 =  e1_1'*OA(:,i); % waist      %!!!moment with respect to torso[rmp]
    OA_2 =  e1_2'*OA(:,i); %            %------------ll---------        [rmp]
    OA_3 =  e4_3'*OA(:,i); %            %------ ll -----     pelvis     [rmp]
    OB_1 =  e1_1'*OB(:,i); % Right arm  %------ ll -----     torso      [rmp]
    OB_2 =  e1_2'*OB(:,i); %            %------------ll---------        [rmp]
    OB_3 =  e1_3'*OB(:,i); %            %------------ll---------        [rmp]
    OC_1 =  e1_1'*OC(:,i); % Left arm   %------------ll---------        [rmp]
    OC_2 =  e1_2'*OC(:,i); %            %------------ll---------        [rmp]
    OC_3 =  e1_3'*OC(:,i); %            %------------ll---------        [rmp]
    OD_1 =  e5_1'*OD(:,i); % Right hip  %------ ll -----     pelvis     [rmp]
    OD_2 =  e5_2'*OD(:,i); %            %------ ll -----      thigh     [rmp]
    OD_3 =  e4_3'*OD(:,i); %            %------------ll---------        [rmp]
    OH_1 =  e5_1'*OH(:,i); % Right knee %-------ll------      thigh     [rmp]
    OH_2 =  e5_2'*OH(:,i); %            %------------ll---------        [rmp]
    OH_3 =  e5_3'*OH(:,i); %            %------------ll---------        [rmp]
    OJ_1 =  e7_1'*OJ(:,i); % Right foot %-------ll------      shin       [rmp]
    OJ_2 =  e9_2'*OJ(:,i); %            %-------ll------      foot       [rmp]
    OJ_3 =  e9_3'*OJ(:,i); %            %------------ll---------        [rmp]

% Acceleration at each joint with respect to local coordinate system
```

```
    OA_1Dot =  e1_1'*OADot(:,i); % waist      %!!!moment with respect to torso[rad/s^2]
    OA_2Dot =  e1_2'*OADot(:,i); %            %------------ll---------        [rad/s^2]
    OA_3Dot =  e4_3'*OADot(:,i); %            %------ ll -----     pelvis     [rad/s^2]
    OB_1Dot =  e1_1'*OBDot(:,i); % Right arm  %------ ll -----     torso      [rad/s^2]
    OB_2Dot =  e1_2'*OBDot(:,i); %            %------------ll---------        [rad/s^2]
    OB_3Dot =  e1_3'*OBDot(:,i); %            %------------ll---------        [rad/s^2]
    OC_1Dot =  e1_1'*OCDot(:,i); % Left arm   %------------ll---------        [rad/s^2]
    OC_2Dot =  e1_2'*OCDot(:,i); %            %------------ll---------        [rad/s^2]
    OC_3Dot =  e1_3'*OCDot(:,i); %            %------------ll---------        [rad/s^2]
    OD_1Dot =  e5_1'*ODDot(:,i); % Right hip  %------ ll -----     pelvis     [rad/s^2]
    OD_2Dot =  e5_2'*ODDot(:,i); %            %------ ll -----     thigh      [rad/s^2]
    OD_3Dot =  e4_3'*ODDot(:,i); %            %------------ll---------        [rad/s^2]
    OH_1Dot =  e5_1'*OHDot(:,i); % Right knee %-------ll------     thigh      [rad/s^2]
    OH_2Dot =  e5_2'*OHDot(:,i); %            %------------ll---------        [rad/s^2]
    OH_3Dot =  e5_3'*OHDot(:,i); %            %------------ll---------        [rad/s^2]
    OJ_1Dot =  e7_1'*OJDot(:,i); % Right foot %-------ll------      shin      [rad/s^2]
    OJ_2Dot =  e9_2'*OJDot(:,i); %            %-------ll------      foot      [rad/s^2]
    OJ_3Dot =  e9_3'*OJDot(:,i); %            %------------ll---------        [rad/s^2]

%  Max and average values
%  Max. power
    PTorsoXi(i)         = abs((OA_1*1/RadRpm)*CA_1);
    PTorsoEta(i)        = abs((OA_2*1/RadRpm)*CA_2);
    PTorsoZeta(i)       = abs((OA_3*1/RadRpm)*CA_3);
    PRShoulderXi(i)     = abs((OB_1*1/RadRpm)*CB_1);
    PRShoulderEta(i)    = abs((OB_2*1/RadRpm)*CB_2);
    PRShoulderZeta(i)   = abs((OB_3*1/RadRpm)*CB_3);
    PLShoulderXi(i)     = abs((OC_1*1/RadRpm)*CC_1);
    PLShoulderEta(i)    = abs((OC_2*1/RadRpm)*CC_2);
    PLShoulderZeta(i)   = abs((OC_3*1/RadRpm)*CC_3);
    PHipXi(i)           = abs((OD_1*1/RadRpm)*CD_1);
    PHipEta(i)          = abs((OD_2*1/RadRpm)*CD_2);
    PHipZeta(i)         = abs((OD_3*1/RadRpm)*CD_3);
    PKneeXi(i)          = abs((OH_1*1/RadRpm)*CH_1);
    PKneeEta(i)         = abs((OH_2*1/RadRpm)*CH_2);
    PKneeZeta(i)        = abs((OH_3*1/RadRpm)*CH_3);
    PAnkleXi(i)         = abs((OJ_1*1/RadRpm)*CJ_1);
    PAnkleEta(i)        = abs((OJ_2*1/RadRpm)*CJ_2);
    PAnkleZeta(i)       = abs((OJ_3*1/RadRpm)*CJ_3);

% Save torque
    CA_xi(:,i)=CA_1; CA_Eta(:,i)=CA_2;  CA_Zeta(:,i)=CA_3; % Nm
    CB_xi(:,i)=CB_1; CB_Eta(:,i)=CB_2;  CB_Zeta(:,i)=CB_3; % Nm
    CC_xi(:,i)=CC_1; CC_Eta(:,i)=CC_2;  CC_Zeta(:,i)=CC_3; % Nm
    CD_xi(:,i)=CD_1; CD_Eta(:,i)=CD_2;  CD_Zeta(:,i)=CD_3; % Nm
    CH_xi(:,i)=CH_1; CH_Eta(:,i)=CH_2;  CH_Zeta(:,i)=CH_3; % Nm
    CJ_xi(:,i)=CJ_1; CJ_Eta(:,i)=CJ_2;  CJ_Zeta(:,i)=CJ_3; % Nm

 % Velocity absolut
    Vxi(:,i)  = [abs(OA_1) abs(OB_1) abs(OC_1) abs(OD_1) abs(OD_1) abs(OH_1) abs(OH_1)↙
abs(OJ_1) abs(OJ_1)];
    Veta(:,i) = [abs(OA_2) abs(OB_2) abs(OC_2) abs(OD_2) abs(OD_2) abs(OH_2) abs(OH_2)↙
abs(OJ_2) abs(OJ_2)];
    Vzeta(:,i)= [abs(OA_3) abs(OB_3) abs(OC_3) abs(OD_3) abs(OD_3) abs(OH_3) abs(OH_3)↙
abs(OJ_3) abs(OJ_3)];
 % Max acceleration
```

```
    VDotxi(:,i)  = [abs(OA_1Dot) abs(OB_1Dot) abs(OC_1Dot) abs(OD_1Dot) abs(OD_1Dot)↙
abs(OH_1Dot) abs(OH_1Dot) abs(OJ_1Dot) abs(OJ_1Dot)];
    VDoteta(:,i) = [abs(OA_2Dot) abs(OB_2Dot) abs(OC_2Dot) abs(OD_2Dot) abs(OD_2Dot)↙
abs(OH_2Dot) abs(OH_2Dot) abs(OJ_2Dot) abs(OJ_2Dot)];
    VDotzeta(:,i)= [abs(OA_3Dot) abs(OB_3Dot) abs(OC_3Dot) abs(OD_3Dot) abs(OD_3Dot)↙
abs(OH_3Dot) abs(OH_3Dot) abs(OJ_3Dot) abs(OJ_3Dot)];

% Motor torque, velocity and acceleration at each joint with respect to local↙
coordinate system;
% Joint no: 1=A, 2=B, 3=C, 4=D, 5=E, 6==H, 7=I, 8=J, 9=k

    Motor(1).torque(:,i) = [CA_1 CA_2 CA_3]';  Motor(2).torque(:,i) = [CB_1 CB_2↙
CB_3]';
    Motor(3).torque(:,i) = [CC_1 CC_2 CC_3]';  Motor(4).torque(:,i) = [CD_1 CD_2↙
CD_3]';
    Motor(6).torque(:,i) = [CH_1 CH_2 CH_3]';  Motor(8).torque(:,i) = [CJ_1 CJ_2↙
CJ_3]';

    Motor(1).rpm(:,i) = [OA_1 OA_2 OA_3]';     Motor(2).rpm(:,i) = [OB_1 OB_2 OB_3]';
    Motor(3).rpm(:,i) = [OC_1 OC_2 OC_3]';     Motor(4).rpm(:,i) = [OD_1 OD_2 OD_3]';
    Motor(6).rpm(:,i) = [OH_1 OH_2 OH_3]';     Motor(8).rpm(:,i) = [OJ_1 OJ_2 OJ_3]';

    Motor(1).acc(:,i) = [OA_1Dot OA_2Dot OA_3Dot]';     Motor(2).acc(:,i) = [OB_1Dot↙
OB_2Dot OB_3Dot]';
    Motor(3).acc(:,i) = [OC_1Dot OC_2Dot OC_3Dot]';     Motor(4).acc(:,i) = [OD_1Dot↙
OD_2Dot OD_3Dot]';
    Motor(6).acc(:,i) = [OH_1Dot OH_2Dot OH_3Dot]';     Motor(8).acc(:,i) = [OJ_1Dot↙
OJ_2Dot OJ_3Dot]';

end

  MaxVDotXi  = [max(VDotxi(1,:)) max(VDotxi(2,:)) max(VDotxi(3,:)) max(VDotxi(4,:)) max↙
(VDotxi(5,:)) max(VDotxi(6,:)) max(VDotxi(7,:)) max(VDotxi(8,:)) max(VDotxi(9,:))];
  MaxVDotEta = [max(VDoteta(1,:)) max(VDoteta(2,:)) max(VDoteta(3,:)) max(VDoteta(4,:))↙
max(VDoteta(5,:)) max(VDoteta(6,:)) max(VDoteta(7,:)) max(VDoteta(8,:)) max(VDoteta↙
(9,:))];
  MaxVDotZeta= [max(VDotzeta(1,:)) max(VDotzeta(2,:)) max(VDotzeta(3,:)) max(VDotzeta↙
(4,:)) max(VDotzeta(5,:)) max(VDotzeta(6,:)) max(VDotzeta(7,:)) max(VDotzeta(8,:)) max↙
(VDotzeta(9,:))];

  [Joint(5).Acc,Joint(5).place]= max(VDotxi(1,:)); [Joint(3).Acc,Joint(3).place]= max↙
(VDoteta(1,:)); [Joint(1).Acc,Joint(1).place]= max(VDotzeta(1,:));
  [Joint(7).Acc,Joint(7).place]= max(VDoteta(3,:));
  [Joint(25).Acc,Joint(25).place]= max(VDotxi(4,:)); [Joint(27).Acc,Joint(27).place]=↙
max(VDoteta(4,:)); [Joint(23).Acc,Joint(23).place]= max(VDotzeta(4,:));
  [Joint(29).Acc,Joint(29).place]= max(VDoteta(6,:));
  [Joint(21).Acc,Joint(21).place]= max(VDotxi(8,:)); [Joint(19).Acc,Joint(19).place]=↙
max(VDoteta(8,:));

  % Max torque and place
  [Joint(5).Torque,Joint(5).Tplace]= max(Motor(1).torque(1,:));[Joint(3).Torque,Joint↙
(3).Tplace]= max(Motor(1).torque(2,:)); [Joint(1).Torque,Joint(1).Tplace]= max(Motor↙
(1).torque(3,:));
  [Joint(7).Torque,Joint(7).Tplace]= max(Motor(3).torque(2,:));
  [Joint(25).Torque,Joint(25).Tplace]= max(Motor(4).torque(1,:)); [Joint(27).Torque,↙
Joint(27).Tplace]= max(Motor(4).torque(2,:)); [Joint(23).Torque,Joint(23).Tplace]= max↙
```

```
(Motor(4).torque(3,:));
  [Joint(29).Torque,Joint(29).Tplace]= max(Motor(6).torque(2,:));
  [Joint(21).Torque,Joint(21).Tplace]= max(Motor(8).torque(1,:)); [Joint(19).Torque,↙
Joint(19).Tplace]= max(Motor(8).torque(2,:));

  save DCMotorData/Joint.mat Joint

  % Max power at each joint; Max. value,Frame number
  [MaxTXi,TXi]= max(PTorsoXi);[MaxTEta,TEta]=max(PTorsoEta);[MaxTZeta,TZeta]=max↙
(PTorsoZeta);
  [MaxRSXi,RSXi]= max(PRShoulderXi); [MaxRSEta,RSEta]=max(PRShoulderEta); [MaxRSZeta,↙
RSZeta]=max(PRShoulderZeta);
  [MaxLSXi,LSXi]= max(PLShoulderXi); [MaxLSEta,LSEta]=max(PLShoulderEta); [MaxLSZeta,↙
LSZeta]=max(PLShoulderZeta);
  [MaxHXi,HXi]= max(PHipXi); [MaxHEta,HEta]=max(PHipEta); [MaxHZeta,HZeta]=max↙
(PHipZeta);
  [MaxKXi,KXi]= max(PKneeXi); [MaxKEta,KEta]=max(PKneeEta); [MaxKZeta,KZeta]=max↙
(PKneeZeta);
  [MaxAXi,AXi]= max(PAnkleXi); [MaxAEta,AEta]=max(PAnkleEta); [MaxAZeta,AZeta]=max↙
(PAnkleZeta);


  % Save max. power to struct
  Motor(1).PMax = [ MaxTXi MaxTEta MaxTZeta; TXi TEta TZeta]'; Motor(2).PMax = [MaxRSXi↙
MaxRSEta MaxRSZeta; RSXi RSEta RSZeta]';
  Motor(3).PMax = [MaxLSXi MaxLSEta MaxLSZeta; LSXi LSEta LSZeta]'; Motor(4).PMax =↙
[MaxHXi MaxHEta MaxHZeta; HXi HEta HZeta]';
  Motor(6).PMax = [MaxKXi MaxKEta MaxKZeta; KXi KEta KZeta]'; Motor(8).PMax = [MaxAXi↙
MaxAEta MaxAZeta; AXi AEta AZeta]';

  % Save max. poWer to printout
  MaxPowerXi     = [MaxTXi MaxRSXi MaxLSXi MaxHXi MaxHXi MaxKXi MaxKXi MaxAXi MaxAXi];
  MaxPowerEta    = [MaxTEta MaxRSEta MaxLSEta MaxHEta MaxHEta MaxKEta MaxKEta MaxAEta↙
MaxAEta];
  MaxPowerZeta   = [MaxTZeta MaxRSZeta MaxLSZeta MaxHZeta MaxHZeta MaxKZeta MaxKZeta↙
MaxAZeta MaxAZeta];

% Max torque
  Motor(1).TMax = [max(abs(CA_xi)) max(abs(CA_Eta)) max(abs(CA_Zeta))]';  Motor(2).TMax↙
= [max(abs(CB_xi)) max(abs(CB_Eta)) max(abs(CB_Zeta))]';
  Motor(3).TMax = [max(abs(CC_xi)) max(abs(CC_Eta)) max(abs(CC_Zeta))]';  Motor(4).TMax↙
= [max(abs(CD_xi)) max(abs(CD_Eta)) max(abs(CD_Zeta))]';
  Motor(6).TMax = [max(abs(CH_xi)) max(abs(CH_Eta)) max(abs(CH_Zeta))]';  Motor(8).TMax↙
= [max(abs(CJ_xi)) max(abs(CJ_Eta)) max(abs(CJ_Zeta))]';

  MmaxXi  = [Motor(1).TMax(1) Motor(2).TMax(1) Motor(3).TMax(1) Motor(4).TMax(1) Motor↙
(4).TMax(1) Motor(6).TMax(1) Motor(6).TMax(1) Motor(8).TMax(1) Motor(8).TMax(1)];
  MmaxEta = [Motor(1).TMax(2) Motor(2).TMax(2) Motor(3).TMax(2) Motor(4).TMax(2) Motor↙
(4).TMax(2) Motor(6).TMax(2) Motor(6).TMax(2) Motor(8).TMax(2) Motor(8).TMax(2)];
  MmaxZeta= [Motor(1).TMax(3) Motor(2).TMax(3) Motor(3).TMax(3) Motor(4).TMax(3) Motor↙
(4).TMax(3) Motor(6).TMax(3) Motor(6).TMax(3) Motor(8).TMax(3) Motor(8).TMax(3)];

  % Calculate RMS torque
  PA1=0;PA2=0;PA3=0;PB1=0;PB2=0;PB3=0;PC1=0;PC2=0;PC3=0;PD1=0;PD2=0;PD3=0;PH1=0;PH2=0;↙
PH3=0;PJ1=0;PJ2=0;PJ3=0;
  deltaT=Tspan(end)/nFrames;
```

```
for FrameNo = 1:nFrames % numerical integration of f(t,M)=delta(torque)*delta(time)
    PA1= PA1+abs(CA_xi(FrameNo)^2)*deltaT; PA2= PA2+abs(CA_Eta(FrameNo)^2)*deltaT; PA3=↙
PA3+abs(CA_Zeta(FrameNo)^2)*deltaT;
    PB1= PB2+abs(CB_xi(FrameNo)^2)*deltaT; PB2= PB2+abs(CB_Eta(FrameNo)^2)*deltaT; PB3=↙
PB3+abs(CB_Zeta(FrameNo)^2)*deltaT;
    PC1= PC1+abs(CC_xi(FrameNo)^2)*deltaT; PC2= PC2+abs(CC_Eta(FrameNo)^2)*deltaT; PC3=↙
PC3+abs(CC_Zeta(FrameNo)^2)*deltaT;
    PD1= PD1+abs(CD_xi(FrameNo)^2)*deltaT; PD2= PD2+abs(CD_Eta(FrameNo)^2)*deltaT; PD3=↙
PD3+abs(CD_Zeta(FrameNo)^2)*deltaT;
    PH1= PH1+abs(CH_xi(FrameNo)^2)*deltaT; PH2= PH2+abs(CH_Eta(FrameNo)^2)*deltaT; PH3=↙
PH3+abs(CH_Zeta(FrameNo)^2)*deltaT;
    PJ1= PJ1+abs(CJ_xi(FrameNo)^2)*deltaT; PJ2= PJ2+abs(CJ_Eta(FrameNo)^2)*deltaT; PJ3=↙
PJ3+abs(CJ_Zeta(FrameNo)^2)*deltaT;
end

  MRmsA_1=sqrt(1/Tspan(end)*PA1); MRmsA_2=sqrt(1/Tspan(end)*PA2);MRmsA_3=sqrt(1/Tspan↙
(end)*PA3);
  MRmsB_1=sqrt(1/Tspan(end)*PB1); MRmsB_2=sqrt(1/Tspan(end)*PB2);MRmsB_3=sqrt(1/Tspan↙
(end)*PB3);
  MRmsC_1=sqrt(1/Tspan(end)*PC1); MRmsC_2=sqrt(1/Tspan(end)*PC2);MRmsC_3=sqrt(1/Tspan↙
(end)*PC3);
  MRmsD_1=sqrt(1/Tspan(end)*PD1); MRmsD_2=sqrt(1/Tspan(end)*PD2);MRmsD_3=sqrt(1/Tspan↙
(end)*PD3);
  MRmsH_1=sqrt(1/Tspan(end)*PH1); MRmsH_2=sqrt(1/Tspan(end)*PH2);MRmsH_3=sqrt(1/Tspan↙
(end)*PH3);
  MRmsJ_1=sqrt(1/Tspan(end)*PJ1); MRmsJ_2=sqrt(1/Tspan(end)*PJ2);MRmsJ_3=sqrt(1/Tspan↙
(end)*PJ3);

  MrmsXi  = [MRmsA_1 MRmsB_1 MRmsC_1 MRmsD_1 MRmsD_1 MRmsH_1 MRmsH_1 MRmsJ_1 MRmsJ_1];
  MrmsEta = [MRmsA_2 MRmsB_2 MRmsC_2 MRmsD_2 MRmsD_2 MRmsH_2 MRmsH_2 MRmsJ_2 MRmsJ_2];
  MrmsZeta= [MRmsA_3 MRmsB_3 MRmsC_3 MRmsD_3 MRmsD_3 MRmsH_3 MRmsH_3 MRmsJ_3 MRmsJ_3];

  Motor(1).Mrms = [MRmsA_1 MRmsA_2 MRmsA_3]';  Motor(2).Mrms = [MRmsB_1 MRmsB_2↙
MRmsB_3]';
  Motor(3).Mrms = [MRmsC_1 MRmsC_2 MRmsC_3]';  Motor(4).Mrms = [MRmsD_1 MRmsD_2↙
MRmsD_3]';
  Motor(6).Mrms = [MRmsH_1 MRmsH_2 MRmsH_3]';  Motor(8).Mrms = [MRmsJ_1 MRmsJ_2↙
MRmsJ_3]';

  % Avg. torque for HD gear (CMC)
   PA1=0;PA2=0;PA3=0;PB2=0;PC2=0;PD1=0;PD2=0;PD3=0;PH2=0;PJ1=0;PJ2=0;
   PAA1=0;PAA2=0;PAA3=0;PBB2=0;PCC2=0;PDD1=0;PDD2=0;PDD3=0;PHH2=0;PJJ1=0;PJJ2=0;
for FrameNo = 1:nFrames % Numerical integration of f(t,M)=delta(torque)*delta(time)

    PA1= PA1+abs(Vxi(1,FrameNo)*CA_xi(FrameNo)^3)*deltaT; PA2= PA2+abs(Veta(1,FrameNo)↙
*CA_Eta(FrameNo)^3)*deltaT; PA3= PA3+abs(Vzeta(1,FrameNo)*CA_Zeta(FrameNo)^3)*deltaT;
    PB2= PB2+abs(Veta(2,FrameNo)*CB_Eta(FrameNo)^3)*deltaT;
    PC2= PC2+abs(Veta(3,FrameNo)*CC_Eta(FrameNo)^3)*deltaT;
    PD1= PD1+abs(Vxi(4,FrameNo)*CD_xi(FrameNo)^3)*deltaT; PD2= PD2+abs(Veta(4,FrameNo)↙
*CD_Eta(FrameNo)^3)*deltaT; PD3= PD3+abs(Vzeta(4,FrameNo)*CD_Zeta(FrameNo)^3)*deltaT;
    PH2= PH2+abs(Veta(6,FrameNo)*CH_Eta(FrameNo)^3)*deltaT;
    PJ1= PJ1+abs(Vxi(8,FrameNo)*CJ_xi(FrameNo)^3)*deltaT; PJ2= PJ2+abs(Veta(8,FrameNo)↙
*CJ_Eta(FrameNo)^3)*deltaT;

    PAA1= PAA1+abs(Vxi(1,FrameNo))*deltaT; PAA2= PAA2+abs(Veta(1,FrameNo))*deltaT;↙
```

```
PAA3= PAA3+abs(Vzeta(1,FrameNo))*deltaT;
    PBB2= PBB2+abs(Veta(2,FrameNo))*deltaT;
    PCC2= PCC2+abs(Veta(3,FrameNo))*deltaT;
    PDD1= PDD1+abs(Vxi(4,FrameNo))*deltaT; PDD2= PDD2+abs(Veta(4,FrameNo))*deltaT;↙
PDD3= PDD3+abs(Vzeta(4,FrameNo))*deltaT;
    PHH2= PHH2+abs(Veta(6,FrameNo))*deltaT;
    PJJ1= PJJ1+abs(Vxi(8,FrameNo))*deltaT; PJJ2= PJJ2+abs(Veta(8,FrameNo))*deltaT;

end
% The avearge torque at gear unit (cmc)
MavA_1=(PA1/PAA1)^(1/3); MavA_2=(PA2/PAA2)^(1/3);MavA_3=(PA3/PAA3)^(1/3);
MavB_2=(PB2/PBB2)^(1/3);
MavC_2=(PC1/PCC2)^(1/3);
MavD_1=(PD1/PDD1)^(1/3); MavD_2=(PD2/PDD2)^(1/3);MavD_3=(PD3/PDD3)^(1/3);
MavH_2=(PH2/PHH2)^(1/3);
MavJ_1=(PJ1/PJJ1)^(1/3); MavJ_2=(PJ2/PJJ2)^(1/3);

MavXi  = [MavA_1   0      0    MavD_1 MavD_1   0      0    MavJ_1 MavJ_1];
MavEta = [MavA_2 MavB_2 MavC_2 MavD_2 MavD_2 MavH_2 MavH_2 MavJ_2 MavJ_2];
MavZeta= [MavA_3   0      0    MavD_3 MavD_3   0      0      0       0  ];

Motor(1).Mav = [MavA_1 MavA_2 MavA_3]';  Motor(2).Mav = [0 MavB_2 0]';
Motor(3).Mav = [0 MavC_2 0]';  Motor(4).Mav = [MavD_1 MavD_2 MavD_3]';
Motor(6).Mav = [0 MavH_2 0]';  Motor(8).Mav = [MavJ_1 MavJ_2 0]';
SavXi1=0; SavXi4=0; SavXi6=0; SavXi8=0;
SavEta1=0; SavEta2=0; SavEta3=0; SavEta4=0; SavEta6=0; SavEta8=0;
SavZeta1=0; SavZeta4=0;
for i = 1:nFrames
SavXi1=SavXi1+Vxi(1,i)*deltaT; SavXi4=SavXi4+Vxi(4,i)*deltaT; SavXi8=SavXi8+Vxi(8,i)↙
*deltaT;
SavEta1=SavEta1+Veta(1,i)*deltaT; SavEta2=SavEta2+Veta(2,i)*deltaT;↙
SavEta3=SavEta3+Veta(3,i)*deltaT; SavEta4=SavEta4+Veta(4,i)*deltaT;↙
SavEta6=SavEta6+Veta(6,i)*deltaT; SavEta8=SavEta8+Veta(8,i)*deltaT;
SavZeta1=SavZeta1+Vzeta(1,i)*deltaT; SavZeta4=SavZeta4+Vzeta(4,i)*deltaT;
end
SavXi  = [SavXi1/Tspan(end)   0      0    SavXi4/Tspan(end) SavXi4/Tspan(end)   0↙
0    SavXi8/Tspan(end) SavXi8/Tspan(end)];
SavEta = [SavEta1/Tspan(end) SavEta2/Tspan(end) SavEta3/Tspan(end) SavEta4/Tspan(end)↙
SavEta4/Tspan(end) SavEta6/Tspan(end) SavEta6/Tspan(end) SavEta8/Tspan(end)↙
SavEta8/Tspan(end)];
SavZeta= [SavZeta1/Tspan(end)   0      0    SavZeta4/Tspan(end) SavZeta4/Tspan(end)   0↙
0      0       0  ];


% Max velocity
MaxVXi  = [max(Vxi(1,:)) max(Vxi(2,:)) max(Vxi(3,:)) max(Vxi(4,:)) max(Vxi(5,:)) max↙
(Vxi(6,:)) max(Vxi(7,:)) max(Vxi(8,:)) max(Vxi(9,:))];
MaxVEta = [max(Veta(1,:)) max(Veta(2,:)) max(Veta(3,:)) max(Veta(4,:)) max(Veta(5,:))↙
max(Veta(6,:)) max(Veta(7,:)) max(Veta(8,:)) max(Veta(9,:))];
MaxVZeta= [max(Vzeta(1,:)) max(Vzeta(2,:)) max(Vzeta(3,:)) max(Vzeta(4,:)) max(Vzeta↙
(5,:)) max(Vzeta(6,:)) max(Vzeta(7,:)) max(Vzeta(8,:)) max(Vzeta(9,:))];

Motor(1).MaxV = [MaxVXi(1) MaxVEta(1) MaxVZeta(1)]';  Motor(2).MaxV = [0 MaxVEta(2)↙
0]';
Motor(3).MaxV = [0 MaxVEta(3) 0]';  Motor(4).MaxV = [MaxVXi(4) MaxVEta(4) MaxVZeta↙
(4)]';
```

```
Motor(6).MaxV = [0 MaxVEta(6) 0]';  Motor(8).MaxV = [MaxVXi(8) MaxVEta(8) 0]';

% Average velocity
MeanVXi  = [mean(Vxi(1,:)) mean(Vxi(2,:)) mean(Vxi(3,:)) mean(Vxi(4,:)) mean(Vxi(5,:))↙
mean(Vxi(6,:)) mean(Vxi(7,:)) mean(Vxi(8,:)) mean(Vxi(9,:))];
MeanVEta = [mean(Veta(1,:)) mean(Veta(2,:)) mean(Veta(3,:)) mean(Veta(4,:)) mean(Veta↙
(5,:)) mean(Veta(6,:)) mean(Veta(7,:)) mean(Veta(8,:)) mean(Veta(9,:))];
MeanVZeta= [mean(Vzeta(1,:)) mean(Vzeta(2,:)) mean(Vzeta(3,:)) mean(Vzeta(4,:)) mean↙
(Vzeta(5,:)) mean(Vzeta(6,:)) mean(Vzeta(7,:)) mean(Vzeta(8,:)) mean(Vzeta(9,:))];

Motor(1).Mean = [MeanVXi(1) MeanVEta(1) MeanVZeta(1)]'; Motor(2).Mean = [MeanVXi(2)↙
MeanVEta(2) MeanVZeta(2)]';
Motor(3).Mean = [MeanVXi(3) MeanVEta(3) MeanVZeta(3)]'; Motor(4).Mean = [MeanVXi(4)↙
MeanVEta(4) MeanVZeta(4)]';
Motor(6).Mean = [MeanVXi(6) MeanVEta(6) MeanVZeta(6)]'; Motor(8).Mean = [MeanVXi(8)↙
MeanVEta(8) MeanVEta(8)]';

% List values
fprintf('\nMax speed:\n')
fprintf('Joint #:                    1      2      3      4      5      6      7↙
8      9\n');
fprintf('Max{Velociti Xi}    [RPM]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MaxVXi);
fprintf('Max{Velociti Eta}   [RPM]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MaxVEta);
fprintf('Max{Velociti Zeta}  [RPM]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MaxVZeta);
fprintf('\nMean speed:\n')
fprintf('Mean{Velociti Xi}   [RPM]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MeanVXi);
fprintf('Mean{Velociti Eta}  [RPM]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MeanVEta);
fprintf('Mean{Velociti Zeta} [RPM]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MeanVZeta);
fprintf('\nMax torque:\n')
fprintf('M{Max Xi}            [Nm]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MmaxXi);
fprintf('M{Max Eta}           [Nm]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MmaxEta);
fprintf('M{Max Zeta}          [Nm]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MmaxZeta);
fprintf('\nRMS torque:\n')
fprintf('M{RMS Xi}            [Nm]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MrmsXi);
fprintf('M{RMS Eta}           [Nm]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MrmsEta);
fprintf('M{RMS Zeta}          [Nm]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MrmsZeta);
fprintf('\nAvg. torque gear output:\n')
fprintf('M{Avg. Xi}           [Nm]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MavXi);
fprintf('M{Avg. Eta}          [Nm]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MavEta);
fprintf('M{Avg. Zeta}         [Nm]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MavZeta);
fprintf('\nMax power:\n')
```

```
fprintf('Max{Power Xi}          [W]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MaxPowerXi);
fprintf('Max{Power Eta}         [W]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MaxPowerEta);
fprintf('Max{Power Zeta}        [W]: %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %↙
6.2f\n',MaxPowerZeta);
```

```
% CalcServoConstant
% Operating point at max power (MB,nB)
FrameB = Motor(JointNo).PMax(Axis,2); % Get frame number of max power
OMB = abs(Motor(JointNo).torque(Axis,FrameB))*SC; % Get operation point for torque,↙
times safety constant [Nm]
OnB = abs(Motor(JointNo).rpm(Axis,FrameB)); % Get operation point for speed [rpm]

% max velocity for DC motor and operation point
if CPUM+CPUS>=1
    for Unit=1:5        % Unit= HD gear unit: 1=14, 2=17, 3=20, 4=25, 5=32
        for Ratio=1:5   % Gear ratio: 1=50, 2=80, 3=100, 4=120, 5=160
            IG = Gear(Unit).HD(2+Ratio)*TimRatioS; % Get gear ratio * timing-belt↙
conection
             Motor(JointNo).MinV(Unit,Ratio) = IG*OnB; % Min. servomotor speed at nB↙
[rpm]
             NLSmax(Unit,Ratio) = IG*Motor(JointNo).MaxV(Axis); % Min. motor speed↙
[rpm]
        end
    end
end

% Calculate max. continous torque for CPU-M and CPU-S gears
Mrms = Motor(JointNo).Mrms(Axis); % RMS torque at joint about local axis
MB = zeros(noGear/2,noRatio);
for Ratio=1:noGear/2
    for Unit=1:noRatio

        Ieff = Gear(Unit).HD(2+Ratio)*EtaGear(Unit).HD(Ratio)*TimRatioT; % effective↙
gear ratio = gear ratio times gear eff times beltRatio.
          if Ieff == 0 % CPU-14M120= CPU-14M160=CPU-17M160= do not exsist
          MB(Unit,Ratio) = 0;
          else
          MB(Unit,Ratio)= Mrms*SC*1/Ieff; % Operation torque * safety constant dividet↙
with gear eff. [Nm]

         end
    end
end

% Max. torque at motor shaft based on efficiency and ratio from CPU-M and CPU-S gears
MMax = zeros(noGear/2,noRatio);
for Unit=1:noGear/2
   for Ratio=1:noRatio
    TMax = Motor(JointNo).TMax(Axis); % Max torque at joint(jointNo=1:9)with respect to↙
local CS.
               Ieff = Gear(Unit).HD(2+Ratio)*EtaGear(Unit).HD(Ratio)*TimRatioT; %↙
effektiv gear ratio (gear efficiencies * gear ratio*beltratio)(CPUM)
            if Ieff == 0
                MMax(Unit,Ratio)=0;
            else
                MMax(Unit,Ratio)= TMax*SC*1/Ieff; % Max torque produced by joint at↙
input gear * safty constant [Nm]
            end
   end
 end
```

```
%%%%%%%%%%%%%%%%%%%%% Check if servo motor fits the joint%%%%%%%%%%%%%%%%%%
% Check if max.torque < stall torque for motor
MHbin = zeros(noGear/2,noRatio,noServo);
for sNo = 1:noServo
    for Unit = 1:noGear/2
        for Ratio= 1:noRatio
            MHc = MMax(Unit,Ratio);  % Calculated stall torque
            MH  = servo(sNo).EC(4);  % Get motor stall torque
            if (MHc<MH) && (MHc>0)
            MHbin(Unit,Ratio,sNo)=1; % Stall torque binary
            end
        end
    end
end
% Check if max.speed < no load speed for motor
NLSbin = zeros(noGear/2,noRatio,noServo);
for sNo = 1:noServo
    for Unit = 1:noGear/2
        for Ratio= 1:noRatio
            NLS = NLSmax(Unit,Ratio); % Calculated no load speed
            n0  = servo(sNo).EC(3);   % Get no load speed
            if (NLS<n0) && (NLS>0)
            NLSbin(Unit,Ratio,sNo)=1; % No load speed binary
            end
        end
    end
end
% Check if max. continous torque < max. continuros torque for motor
MCTbin = zeros(noGear/2,noRatio,noServo);
for sNo = 1:noServo
    for Unit = 1:noGear/2
        for Ratio= 1:noRatio
            MBc = MB(Unit,Ratio); % MBc = MB check
            MCT  = servo(sNo).EC(9)*NomTorque; % Get max. continuous torque
            if (MBc<=MCT) && (MBc>0)
            MCTbin(Unit,Ratio,sNo)=1; % continuous torque binar
            end
        end
    end
end
% Check if the same moter satisfies MBc, Knc and NLS
KnMcBin = zeros(noGear/2,noRatio,noServo);
for sNo = 1:noServo
    for Unit = 1:noGear/2
        for Ratio= 1:noRatio
            KnMc = MCTbin(Unit,Ratio,sNo)*MHbin(Unit,Ratio,sNo)*NLSbin(Unit,Ratio,sNo);
            if KnMc==1
                KnMcBin(Unit,Ratio,sNo)= 1;
            end
        end
    end
end
```

```
function DCMotor=DCclear(DCMotor,R)
% Clear DCMotor constants

% Right side of robot
DCMotor(R(1)).No=0;
DCMotor(R(1)).Type=0;
DCMotor(R(1)).Power=0;
DCMotor(R(1)).Volt=0;
DCMotor(R(1)).Pin=0;
DCMotor(R(1)).Pout=0;
DCMotor(R(1)).UnitType=0;
DCMotor(R(1)).UnitNo=0;
DCMotor(R(1)).Ratio=0;

% Left side of robot
DCMotor(R(2)).No=0;
DCMotor(R(2)).Type=0;
DCMotor(R(2)).Power=0;
DCMotor(R(2)).Volt=0;
DCMotor(R(2)).Pin=0;
DCMotor(R(2)).Pout=0;
DCMotor(R(2)).UnitType=0;
DCMotor(R(2)).UnitNo=0;
DCMotor(R(2)).Ratio=0;

save DCMotorData/DCMotor.mat DCMotor;
```

```
function DCMotorData(ServoPower,ServoVolt,ServoSign,ServoOrder,R,Pin,Pout,UnitNo,↙
RatioNo,Utype,DCMotor)%
% save DC motor Data: Order number, Assigned power, Voltage, Moter No., ...

% clear DCMotor array
DCMotor=DCclear(DCMotor,R);
% The data is saved under the revolute joint number

% Right side of robot
DCMotor(R(1)).No=ServoOrder;
DCMotor(R(1)).Type=ServoSign;
DCMotor(R(1)).Power=ServoPower;
DCMotor(R(1)).Volt=ServoVolt;
DCMotor(R(1)).Pin=Pin;
DCMotor(R(1)).Pout=Pout;
DCMotor(R(1)).UnitType=Utype;
DCMotor(R(1)).UnitNo=UnitNo;
DCMotor(R(1)).Ratio=RatioNo;

% Left side of robot
DCMotor(R(2)).No=ServoOrder;
DCMotor(R(2)).Type=ServoSign;
DCMotor(R(2)).Power=ServoPower;
DCMotor(R(2)).Volt=ServoVolt;
DCMotor(R(2)).Pin=Pin;
DCMotor(R(2)).Pout=Pout;
DCMotor(R(2)).UnitType=Utype;
DCMotor(R(2)).UnitNo=UnitNo;
DCMotor(R(2)).Ratio=RatioNo;

save DCMotorData/DCMotor.mat DCMotor;
```

```
function [ne]=EtaE(Unit)

% Correction value for s-unit
Eta=[ 0.06  0.04  0.02  0      0;
      0.06  0.04  0.02  0      0;
      0.075 0.06  0.05  0.025  0;
      0     0.08  0.07  0.05   0.02;
      0     0     0.07  0.05   0.02];
  ne(1).e=Eta(1,Unit); ne(2).e=Eta(2,Unit); ne(3).e=Eta(3,Unit);
  ne(4).e=Eta(4,Unit); ne(5).e=Eta(5,Unit);
```

```
% Gear data
% Propperties for Harmonic Drive AG CPU-S and CPU-M gear units
% CPU-M
J_M14 = 0.033*10^-4; % kgm^2
J_M17 = 0.079*10^-4; % kgm^2
J_M20 = 0.193*10^-4; % kgm^2
J_M25 = 0.413*10^-4; % kgm^2
J_M32 = 1.69*10^-4;  % kgm^2
M_M14 = 0.54;        % kg
M_M17 = 0.79;        % kg
M_M20 = 1.30;        % kg
M_M25 = 1.95;        % kg
M_M32 = 3.95;        % kg

% CPU-S
J_S14 = 0.025*10^-4; % kgm^2
J_S17 = 0.059*10^-4; % kgm^2
J_S20 = 0.137*10^-4; % kgm^2
J_S25 = 0.320*10^-4; % kgm^2
J_S32 = 1.200*10^-4; % kgm^2
M_S14 = 0.64;        % kg
M_S17 = 0.95;        % kg
M_S20 = 1.50;        % kg
M_S25 = 2.50;        % kg
M_S32 = 5.40;        % kg

% Common data for CPU units
n_14 = [50 80 100 0 0];     % Ratio 1:n
n_17 = [50 80 100 120 0];   % Ratio 1:n
n_20 = [50 80 100 120 160]; % Ratio 1:n
n_25 = [50 80 100 120 160]; % Ratio 1:n
n_32 = [50 80 100 120 160]; % Ratio 1:n
nAM_14 = [3500 8500];       % Avg and max input speed [rpm]
nAM_17 = [3500 7300];       % Avg and max input speed [rpm]
nAM_20 = [3500 6500];       % Avg and max input speed [rpm]
nAM_25 = [3500 5600];       % Avg and max input speed [rpm]
nAM_32 = [3500 4800];
TA_14 = [6.9 11 11 0 0];    % Avg torque for each ratio [Nm]
TA_17 = [26 27 39 39 0];
TA_20 = [34 47 49 49 49];
TA_25 = [55 87 108 108 108];
TA_32 = [108 167 216 216 216];
TM_14 = [35 47 54 0 0];     % Max peak torque for each ratio [Nm]
TM_17 = [70 87 110 86 0];
TM_20 = [98 127 147 147 147];
TM_25 = [186 255 284 304 314];
TM_32 = [382 568 647 686 686];
TR_14 = [18 23 28 0 0 ];    % Limit for reapeated peak torque [Nm]
TR_17 = [34 43 54 54 0];
TR_20 = [56 74 82 87 92];
TR_25 = [98 137 157 167 176];
TR_32 = [216 304 333 353 372];
TN_14 = [5.4 7.8 7.8 0 0];  % Rated torque at rated speed (2000 rmp) [Nm]
TN_17 = [16 22 24 24 0];
TN_20 = [25 34 40 40 40];
TN_25 = [39 63 67 67 67];
```

```
TN_32 = [76 118 137 137 137];

Gear(1).HD = [J_M14 M_M14 n_14 nAM_14 TA_14 TR_14 TM_14 TN_14 14]; Gear(2).HD = [J_M17↙
M_M17 n_17 nAM_17 TA_17 TR_17 TM_17 TN_17 17];
Gear(3).HD = [J_M20 M_M20 n_20 nAM_20 TA_20 TR_20 TM_20 TN_20 20]; Gear(4).HD = [J_M25↙
M_M25 n_25 nAM_25 TA_25 TR_25 TM_25 TN_25 25];
Gear(5).HD = [J_M32 M_M32 n_32 nAM_32 TA_32 TR_32 TM_32 TN_32 32]; Gear(6).HD = [J_S14↙
M_S14 n_14 nAM_14 TA_14 TR_14 TM_14 TN_14 14];
Gear(7).HD = [J_S17 M_S17 n_17 nAM_17 TA_17 TR_17 TM_17 TN_17 17]; Gear(8).HD = [J_S20↙
M_S20 n_20 nAM_20 TA_20 TR_20 TM_20 TN_20 20];
Gear(9).HD = [J_S25 M_S25 n_25 nAM_25 TA_25 TR_25 TM_25 TM_25 25]; Gear(10).HD= [J_S32↙
M_S32 n_32 nAM_32 TA_32 TR_32 TM_32 TN_32 32];

noRatio= length(n_20); noGear = length(Gear); noGearRatio= noGear*noRatio;
Gear(1).J=J_M14; Gear(2).J=J_M17; Gear(3).J=J_M20; Gear(4).J=J_M25; Gear(5).J=J_M32;
Gear(6).J=J_S14; Gear(7).J=J_S17; Gear(8).J=J_S20; Gear(9).J=J_S25; Gear(10).J=J_S32;
```

```
% Gear efficiency
function []=HDGearEff(Unit,Ratio,Axis)

    if (Unit==1)  && (CPUM==1) % Efficiency of CPU-M 14

       eta50c = -4*10^-5*nMotor(1)+0.7333; % eta from efficiency chart
       VG     = Motor(JointNo).Mrms(Axis)/TimRatioT*1/Gear(Unit).HD(25); % torque↙
factor for a gear
       kG     = 0.2695*reallog(VG)+1.0656; % K factor is gear dependent
       if (kG<1) && (kG>0.1)
              eta50  = eta50c*kG;  % efficiency at a given ratio for a given unit
       elseif kG>=1  % upper boundary of graph
              eta50 = eta50c;
       elseif kG<=0.1 % lower boundary of graph
              eta50 = eta50c*0.1;
       end

       eta80c = -4*10^-5*nMotor(2)+0.7333;
       VG     = Motor(JointNo).Mrms(Axis)/TimRatioT*1/Gear(Unit).HD(26);
       kG     = 0.2695*reallog(VG)+1.0656;
       if (kG<1) && (kG>0.1)
              eta80  = eta80c*kG;  % efficiency at a given ratio for a given unit
       elseif kG>=1  % upper boundary of graph
              eta80 = eta80c;
       elseif kG<=0.1 % lower boundary of graph
              eta80 = eta80c*0.1;
       end;

       eta100c  = -5*10^-5*nMotor(3)+0.7583;
       VG       = Motor(JointNo).Mrms(Axis)/TimRatioT*1/Gear(Unit).HD(27);
       kG       = 0.2695*reallog(VG)+1.0656;
       if (kG<1) && (kG>0.1)
              eta100  = eta100c*kG;  % efficiency at a given ratio for a given unit
       elseif kG>=1  % upper boundary of graph
              eta100 = eta100c;
       elseif kG<=0.1 % lower boundary of graph
              eta100 = eta100c*0.1;
       end

       EtaGear(1).HD = [eta50 eta80 eta100 0 0];
    end

    if (Unit>1) && (CPUM==1) % Efficiency of CPU-M 17, 20, 25 and 32

       eta50c   = -4*10^-5*nMotor(1)+0.8; % eta from efficiency  chart
       VG       = Motor(JointNo).Mrms(Axis)/TimRatioT*1/Gear(Unit).HD(25);
       kG       = 0.2695*reallog(VG)+1.0656;
       if (kG<1) && (kG>0.1)
              eta50  = eta50c*kG;  % efficiency  at a given ratio for a given unit
       elseif kG>=1  % upper boundary of graph
              eta50 = eta50c;
       elseif kG<=0.1 % lower boundary of graph
              eta50 = eta50c*0.1;
       end

       eta80c   = -6*10^-5*nMotor(2)+0.8467;
```

```
        VG       = Motor(JointNo).Mrms(Axis)/TimRatioT*1/Gear(Unit).HD(26);
        kG       = 0.2695*reallog(VG)+1.0656;
        if (kG<1) && (kG>0.1)
               eta80  = eta80c*kG;  % efficiency  at a given ratio for a given unit
        elseif kG>=1  % upper boundary of graph
               eta80 = eta80c;
        elseif kG<=0.1 % lower boundary of graph
               eta80 = eta80c*0.1;
        end

        eta100c  = -6*10^-5*nMotor(3)+0.8467;
        VG       = Motor(JointNo).Mrms(Axis)/TimRatioT*1/Gear(Unit).HD(27);
        kG       = 0.2695*reallog(VG)+1.0656;
        if (kG<1) && (kG>0.1)
               eta100  = eta100c*kG;  % efficiency  at a given ratio for a given unit
        elseif kG>=1  % upper boundary of graph
               eta100 = eta100c;
        elseif kG<=0.1 % lower boundary of graph
               eta100 = eta100c*0.1;
        end

        eta120c  = -5*10^-5*nMotor(4)+0.8217;
        VG       = Motor(JointNo).Mrms(Axis)/TimRatioT*1/Gear(Unit).HD(28);
        kG       = 0.2695*reallog(VG)+1.0656;
        if (kG<1) && (kG>0.1)
               eta120  = eta120c*kG;  % efficiency  at a given ratio for a given unit
        elseif kG>=1  % upper boundary of graph
               eta120 = eta120c;
        elseif kG<=0.1 % lower boundary of graph
               eta120 = eta120c*0.1;
        end

        if nMotor(5)==0
            eta160=0;
        else
            eta160c  = -6*10^-5*nMotor(5)+0.8035;
            VG       = Motor(JointNo).Mrms(Axis)/TimRatioT*1/Gear(Unit).HD(28);
            kG       = 0.2695*reallog(VG)+1.0656;
            if (kG<1) && (kG>0.1)
                   eta160  = eta160c*kG;  % efficiency  at a given ratio for a given↙
unit
            elseif kG>=1  % upper boundary of graph
                   eta160 = eta160c;
            elseif kG<=0.1 % lower boundary of graph
                   eta160 = eta160c*0.1;
            end
        end
        EtaGear(Unit).HD = [eta50 eta80 eta100 eta120 eta160];
    end

%%%%%% Calculation of the efficiency of the HD S-Unit. The calculation
%%%%%% procedure at P.152 at Harmonic Drive AG Manual for Units is used.

    if CPUS==1 % Efficiency of CPU-S 14-32 (Unit<5)  &&

               [ne]=EtaE(Unit); % get correction value ne
```

```
                eta50c = -0.0896*reallog(nMotor(1))+1.3641; % eta from efficiency  chart
                VG     = Motor(JointNo).Mrms(Axis)/Gear(Unit+5).HD(25); % torque factor↙
for a gear
                kG     = 0.3206*reallog(VG)+1.0096; % K factor is gear dependent
                if (kG<1) && (kG>0.1)
                eta50  = (eta50c-ne(1).e)*kG;  % efficiency  at a given ratio for a↙
given unit
                elseif kG>=1  % upper boundary of graph
                eta50 = eta50c-ne(1).e;
                elseif kG<=0.2 % lower boundary of graph
                eta50 = (eta50c-ne(1).e)*0.2; % correction value is added... ne().e
                end


                eta80c = -0.0896*reallog(nMotor(2))+1.3641;
                VG     = Motor(JointNo).Mrms(Axis)/Gear(Unit+5).HD(26);
                kG     = 0.3206*reallog(VG)+1.0096;
                if (kG<1) && (kG>0.1)
                eta80  = (eta80c-ne(2).e)*kG;  % efficiency  at a given ratio for a↙
given unit
                elseif kG>=1  % upper boundary of graph
                eta80 = eta80c-ne(2).e;
                elseif kG<=0.2 % lower boundary of graph
                eta80 = (eta80c-ne(2).e)*0.2;
                end

                eta100c  = -0.0896*reallog(nMotor(3))+1.3641;
                VG       = Motor(JointNo).Mrms(Axis)/Gear(Unit+5).HD(27);
                kG       = 0.3206*reallog(VG)+1.0096;
                if (kG<1) && (kG>0.1)
                eta100  = (eta100c-ne(3).e)*kG;  % efficiency  at a given ratio for a↙
given unit
                elseif kG>=1  % upper boundary of graph
                eta100 = eta100c-ne(3).e;
                elseif kG<=0.2 % lower boundary of graph
                eta100 = (eta100c-ne(3).e)*0.2;
                end

                if nMotor(4)==0
                    eta120=0;
                else
                eta120c   = -0.0896*reallog(nMotor(4))+1.3641;
                VG        = Motor(JointNo).Mrms(Axis)/Gear(Unit+5).HD(28);
                kG        = 0.3206*reallog(VG)+1.0096;
                    if (kG<1) && (kG>0.1)
                    eta120  = (eta120c-ne(4).e)*kG;  % efficiency  at a given ratio for↙
a given unit
                    elseif kG>=1  % upper boundary of graph
                    eta120 = eta120c-ne(4).e;
                    elseif kG<=0.2 % lower boundary of graph
                    eta120 = (eta120c-ne(4).e)*0.2;
                    end
                end
                if nMotor(5)==0
                    eta160=0;
                else
```

```
                eta160c   = -0.1554*reallog(nMotor(5))+1.805;
                VG        = Motor(JointNo).Mrms(Axis)/Gear(Unit+5).HD(28);
                kG        = 0.3206*reallog(VG)+1.0096;
                    if (kG<1) && (kG>0.1)
                    eta160  = (eta160c-ne(5).e)*kG;  % efficiency  at a given ratio for↙
 a given unit
                    elseif kG>=1  % upper boundary of graph
                    eta160 = eta160c-ne(5).e;
                    elseif kG<=0.2 % lower boundary of graph
                    eta160 = (eta160c-ne(5).e)*0.2;
                    end
                end
                EtaGear(Unit).HD = [eta50 eta80 eta100 eta120 eta160];
```

```
function[Max]=MaxTorqueSpeed(Joint, DCMotor, Power)
% Max permitted torque and speed for motor and gear.

ServoDataRE48;
HDGearData;
TimingBeltPulleyData;

% Efficiency of  HD Unit appox=50%
Eff=2;

for i=1:33

        B=isempty(Joint(i).I);
        if B==0

        switch Joint(i).UnitNo
            case 14
                Unit=1;
            case 17
                Unit=2;
            case 20
                Unit=3;
        end

        switch Joint(i).Ratio
            case 50
                Ratio=1;
            case 80
                Ratio=2;
            case 100
                Ratio=3;
            case 120
                Ratio=4;
            case 160
                Ratio=5;

        end

        switch DCMotor(i).No
            case 148877
                A=strmatch('2xRE',DCMotor(i).Type);
                if A==1
                    motor=1;
                else
                    motor=8;
                end
            case 218008
                 B=strmatch('2xRE',DCMotor(i).Type);
                if B==1
                    motor=2;
                else
                    motor=4;
                end
            case 273755
                    motor=5;
            case 273762
```

```
                    motor=6;
            case 310009
                    motor=7;
        end

        Max(i).MotorTorque=(Gear(Unit+5).HD(19+Ratio)*Eff/( Gear(3).HD(2+Ratio)*(Pulley↙
(DCMotor(i).Pout).Te)/(Pulley(DCMotor(i).Pin).Te)));
        %Max(i).MotorCurrent=(Gear(Unit+5).HD(19+Ratio)*Eff/( Gear(3).HD(2+Ratio)↙
*Pulley(DCMotor(i).Pout).Te/Pulley(DCMotor(i).Pin).Te))*1/servo(motor).EC(11);
        Max(i).GearTorque=Gear(Unit+5).HD(19+Ratio);
        Max(i).GearSpeed=Gear(Unit+5).HD(9);
        Max(i).GearSpeedAv=Gear(Unit+5).HD(8);
        Max(i).NomCurrent=servo(motor).EC(8);
        Max(i).NomTorque=servo(motor).EC(9);
        Max(i).RmsTorque=Power(1,i).IrmsMotor*servo(motor).EC(11);
    end
end
```

```
function [MGData]=MGclear(MGData,R)

for i=1:2
MGData(R(i)).Jm=0;
MGData(R(i)).Jg=0;
MGData(R(i)).Mm=0;
MGData(R(i)).Mg=0;
MGData(R(i)).Ug=0;
MGData(R(i)).Ub=0;
MGData(R(i)).Jpi=0;
MGData(R(i)).Jpo=0;
MGData(R(i)).Mpi=0;
MGData(R(i)).Mpo=0;
MGData(R(i)).JT=0;
MGData(R(i)).Eff=0;
MGData(R(i)).Mh=0;
MGData(R(i)).N0=0;
end

save GearMotorData/MGData.mat MGData;
```

```
function [MGData,R]=MGPData(JMdata,JointNo,Axis,Pin,Pout,servo,Gear,Pulley,Unit,Ratio,↙
Servo,JT,EtaGear,MGData)%
% Save motor and gear data
switch JointNo
    case 1
        if Axis==1
            R(1)=5;
            R(2)=5;
        elseif Axis==2
            R(1)=3;
            R(2)=3;
        elseif Axis==3
            R(1)=1;
            R(2)=1;
        end
    case {2,3}
        if Axis==2
            R(1)=7;
            R(2)=9;
        end
    case 4
        if Axis==1
            R(1)=25;
            R(2)=13;
        elseif Axis==2
            R(1)=27;
            R(2)=15;
        elseif Axis==3
            R(1)=23;
            R(2)=11;
        end

    case 6
        if Axis==2
            R(1)=29;
            R(2)=17;
        end

    case 8
        if Axis==1
            R(1)=21;
            R(2)=33;
        elseif Axis==2
            R(1)=19;
            R(2)=31;
        end
end
% Clear MGData array
MGData=MGclear(MGData,R);
% Efficiency of gear system
Eff= EtaGear(Unit).HD(Ratio)*0.98; % HD-Unit*belt drive
for i=1:2
    if (JMdata==1) && (Pout+Pin>0)
        % Inetia of mass [kgm^2]
        MGData(R(i)).Jm  = servo(Servo).EC(14); % Rotor
        MGData(R(i)).Jg  = Gear(Unit).HD(1); % Wave generator
```

```
        % Mass [kg]
        MGData(R(i)).Mm  = servo(Servo).EC(15); % Rotor
        MGData(R(i)).Mg  = Gear(Unit).HD(2); % wave generator
        % Gear ratio
        MGData(R(i)).Ug = Gear(3).HD(2+Ratio); % Gear ratio of HD
        MGData(R(i)).Ub = Pulley(Pout).Te/Pulley(Pin).Te; % Gear ratio of belt↙
connection

        MGData(R(i)).Jpi = Pulley(Pin).J;   % Input pulley; Inetia of mass [kgm^2]
        MGData(R(i)).Jpo = Pulley(Pout).J;  % Output pulley [kgm^2]
        MGData(R(i)).Mpi = Pulley(Pin).m;   % Input pulley [kg]
        MGData(R(i)).Mpo = Pulley(Pout).m;  % Output pulley [kg]
        MGData(R(i)).JT = JT;               % Total inertia of gear motor system[kgm^2]
        MGData(R(i)).Eff = Eff;             % Efficiency of the gear system
        MGData(R(i)).Mh = servo(Servo).EC(4); % Motor: stall torque[Nm]
        MGData(R(i)).N0 = servo(Servo).EC(3); % Motor: No load speed[rpm]
    else
        MGData(R(i)).Jm  = servo(Servo).EC(14); % Rotor; inetia of mass [kgm^2]
        MGData(R(i)).Jg  = Gear(Unit).HD(1);    % Wave generator
        MGData(R(i)).Mm = servo(Servo).EC(15);  % Rotor [kg]
        MGData(R(i)).Mg  = Gear(Unit).HD(2);    % wave generator [kg]
        MGData(R(i)).Ug= Gear(3).HD(2+Ratio);   % Gear ratio of HD
        MGData(R(i)).Ub = 0;
        MGData(R(i)).Jpi= 0;
        MGData(R(i)).Jpo= 0;
        MGData(R(i)).Mpi= 0;
        MGData(R(i)).Mpo= 0;
        MGData(R(i)).JT= JT;
        MGData(R(i)).Eff = Eff;
        MGData(R(i)).Mh = servo(Servo).EC(4); % [rpm]
        MGData(R(i)).N0 = servo(Servo).EC(3); % [rpm/Nm]
    end
end
save GearMotorData/MGData.mat MGData;
```

```
function [JV,JT]=PlotMotorGear(Servo,Unit,Ratio,Axis,EtaGear,Pulley,OMB,OnB,Motor,↙
servo,Gear,TimA,TimB,JointNo,LC,PlotMotor,Tspan,Pin,Pout,SC)
% Plot motor and gear data
if TimA+TimB>0 % if belt-drive conection
TimRatioT = Pulley(Pout).Te/Pulley(Pin).Te*0.98; % timing ratio torque
TimRatioS = Pulley(Pout).Te/Pulley(Pin).Te; % timing ratio speed
else
TimRatioT=1;
TimRatioS=1;
end
RatioNo=Gear(3).HD(2+Ratio);
EtaEff=EtaGear(Unit).HD(Ratio); % Efficiency of gear
% Motor or generator: if DC motor is motor; 1/eff_gear. if DC motor generator; eff_gear
for i=1:length(Tspan)
JointSign=Motor(JointNo).rpm(Axis,i)*Motor(JointNo).torque(Axis,i);
    if JointSign>0
     MotorTorque=Motor(JointNo).torque(Axis,i)*SC/(EtaEff*RatioNo*TimRatioT);
    elseif JointSign<=0
        MotorTorque=Motor(JointNo).torque(Axis,i)*SC*EtaEff/(RatioNo*TimRatioT);
    end
   MotorT(:,i)=MotorTorque;
end
JT=MotorT; % Torque at motor times safty constant
Mb=abs(Motor(JointNo).TMax(Axis))*SC/(RatioNo*TimRatioT*EtaEff); % Max absolute torque
nb=0;  % Max absolute torqu @ zero speed
JV=Motor(JointNo).rpm(Axis,:)*RatioNo*TimRatioS; %Motor speed
MHm=servo(Servo).EC(4);% Stall torque for motor
nom=servo(Servo).EC(3); % No load speed for motor
Ms=servo(Servo).EC(7); % Max. permissible speed
McT=servo(Servo).EC(9); % Max. continous torque
NTL=1.4*McT; % 1.4xNominal torque
NTL4=4*McT;
%MS=[Ms Ms 0]; % Line for continous operation
%MT=[0 McT McT];
MH=[MHm 0]; %Torque-speed line
no=[0 nom];
% if torque-speed diamond
 MH=[MHm  0   -MHm   0   MHm ];
 no=[0   nom   0   -nom   0];
% Workspace for continous operation
MT=[0 McT McT McT -McT -McT 0];
MS=[Ms Ms 0  -Ms -Ms Ms Ms];

% limit for 1,4xnominal torque
NT=[NTL NTL ];
NTv=[-NTL -NTL];
NS=[Ms -Ms];
% limit for 4xnominal torque
NT4=[NTL4 NTL4 ];
NT4L=[-NTL4 -NTL4];

%%%%%%%%%%%%%%%%%%%%%%%%%% Gear plot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if TimB==1
JTG=Motor(JointNo).torque(Axis,:);  % Torque at gear output
JVG=Motor(JointNo).rpm(Axis,:);     %Gear output speed
else
```

```
JTG=Motor(JointNo).torque(Axis,:)/(TimRatioT);
JVG=Motor(JointNo).rpm(Axis,:)*TimRatioS; %Gear speed
end
if TimB==1
MSOut=Gear(Unit).HD(9)/RatioNo; % Max. speed in/ratio for gear
ASOut=Gear(Unit).HD(8)/RatioNo;
else
MSOut=Gear(Unit).HD(9)*TimRatioS/RatioNo; % max. speed in/ratio for gear
ASOut=Gear(Unit).HD(8)*TimRatioS/RatioNo; % Limit for average speed in/Ratio for gear
end
TROut=Gear(Unit).HD(Ratio+19);% Limit for momentary peak. torque
TAOut=Gear(Unit).HD(Ratio+9); % Limit for av. torque
XLIM=McT*5;

TRO=[ TROut TROut -TROut -TROut TROut];  % Limit for max. output torque
MSO=[-MSOut MSOut  MSOut -MSOut -MSOut]; % Limit max. output speed
TAO=[ TAOut TAOut -TAOut -TAOut  TAOut]; % Limit for continuous output peak. torque
ASO=[-ASOut ASOut  ASOut -ASOut -ASOut]; % Limit for continuous output speed
% Plot Joint and Motor
if Axis==1
    AxisNo='Roll';
elseif Axis==2
    AxisNo='Pitch';
elseif Axis==3
    AxisNo='Yaw';
end
if PlotMotor==1
    figure
    subplot(3,1,1)
    plot (JT,JV,Mb,nb,'xr',JT(1),JV(1),'ok',MH,no,'--',MT,MS,'-.r',NT,NS,':m',NTv,NS,':↙
m',NT4,NS,':r',NT4L,NS,':r')
    xlim([-XLIM,XLIM])
    xlabel('Torque [Nm]')
    ylabel('Speed [RPM]')
    title(['Motor workspace.  Joint No: ' num2str(JointNo),'    Axis:  ',num2str↙
(AxisNo),',   Safety factor: ',num2str(SC) ])
    if Axis==1
    legend( 'Xi, Motor trajectory','Max abs Torque','starting point','Torque-speed↙
line', 'Continuous operation workspace','1.4xNominal Torque','4xNominal Torque')%↙
'Operation point''worst-case;Torque','worst-case;Speed',
    end
    if Axis==2
        legend( 'Eta, Motor trajectory','Max abs Torque','starting point','Torque-speed↙
line','Continuous operation workspace','1.4xNominal Torque','4xNominal Torque')%'worst-↙
case;Torque','worst-case;Speed',
    end
    if Axis==3
        legend('Zeta, Motor trajectory','Max abs Torque','starting point','Torque-speed↙
line','Continuous operation workspace','1.4xNominal Torque','4xNominal Torque')%'worst-↙
case;Torque','worst-case;Speed',
    end
    subplot(3,1,2)
    plot (Tspan,JT)
    xlabel('Time [s]')
    ylabel('Torque [Nm]')
    title('Motor Torque')
```

```
    if Axis==1
    legend( 'Roll Torque')
    end
    if Axis==2
        legend( 'Pitch Torque')
    end
    if Axis==3
        legend('Yaw Torque')
    end
    grid on

    subplot(3,1,3)
    plot (JTG,JVG,JTG(1),JVG(1),'ok',TRO,MSO,TAO,ASO,'r')
    xlabel('Torque [Nm]')
    ylabel('Speed [RPM]')
    title('Gear workspace')
    if Axis==1
        legend( 'Xi, Gear trajectory','starting point','Limet for momentary output↙
speed/torque', 'Limet for continuous output torque')
    end
    if Axis==2
        legend( 'Eta, Gear trajectory','starting point','Limet for momentary output↙
speed/torque', 'Limet for continuous output torque')
    end
    if Axis==3
        legend('Zeta, Gear trajectory','starting point','Limet for momentary output↙
torque', 'Limet for continuous output torque')
    end
    grid on

 end
```

```
function [Power]=PowerClear(Power,R)

for l=1:2
Power(R(l)).PlossMean=0;
Power(R(l)).PlossMax=0;
Power(R(l)).PmecMean=0;
Power(R(l)).PmecMax=0;
Power(R(l)).IrmsMotor=0;
Power(R(l)).ImaxMotor=0;
Power(R(l)).IbatMax=0;
Power(R(l)).Imotor=0;
Power(R(l)).PmecW=0;
Power(R(l)).PlossW=0;
Power(R(l)).JV=0;
Power(R(l)).JT=0;
Power(R(l)).SC=0;
Power(R(l)).NomTorque=0;
Power(R(l)).Mrms=0;
Power(R(l)).Arms=0;
end
save DCMotorData/Power.mat Power
```

```
function [Motor,Overshoot,Trms,Tmax,Smean,Amax,Mrms,Arms]=PowerEl(servo,ServoNo,JT,JV,↙
R,Motor,Data,SC,NomTorque)
% calculating current values for DC motor
load DCMotorData/Power.mat
% Clear Power
Power=PowerClear(Power,R);
% Motor constants
Km=servo(ServoNo).EC(11);                                % Torque constant [Nm/A]
Rmotor=servo(ServoNo).EC(2)/servo(ServoNo).EC(18);       % [Ohm]
Ubat= servo(ServoNo).EC(2);                              % Nominal voltage for motor
NT= servo(ServoNo).EC(9);                                % Nominal torque
% Data
Tspan=Data.Tspan;       % The time for one step
nFrames= Data.nFrames;  % Number of data samples
vel=JV*pi/30;           % Motor velocity [rad/s]
Smax=max(abs(JV));      % rpm
Smean=mean(abs(JV));    % rpm
MT=JT;                  % Motor torque [Nm]


I=MT*1/Km;      % Motor current [A]

Pmec  =0;
Ploss =0;
% motor effect used
for k=1:nFrames

% if no power is used when returning
        Pjoint(k)=MT(k)*vel(k);
        Pmech=MT(k)*vel(k);
        if Pmech<0
           Pmec(k)=0;
        else
           Pmec(k)=Pmech;
        end
        if Pmech>0
           iM=I(k);                     % Current [A]
           MT(k)=JT(k);                 % Torque [Nm]
           Omega(k)=vel(k);             % Velocity [m/s]
        else
           iM=0;                        % Generated power = 0
           MT(k)=0;
           Omega(k)=0;
        end
           Ploss(k)=iM^2*2*Rmotor;      % Save power loss
           i(k)=iM;                     % save current
end
PlossMean=mean(Ploss);  % Mean power loss for motor and control
Tmax=max(abs(JT));      % max motor torque
Amax=max(abs(I));
[PlMax,frame]=max(Ploss);   % [W] check if power is greatest at Ploss or Pmec
Pm= Pmec(frame);            % [W] mechanical power
PbatLoss=PlMax+Pm;          % [W] Max Power with respect to max power loss from motor
[PmMax,frame1]=max(Pmec);   % [W] max mechancal power
Pl= Ploss(frame1);          % [W] power loss motor
PbatMec=PmMax+Pl;           % [W] Max power with respect to max motor power
```

```
if PbatLoss<PbatMec
    PbatMax=PbatMec;
else
    PbatMax=PbatLoss;
end
IbatMax=PbatMax/Ubat; % [A]
% total  effect used for one step
idt=0; Tdt=0; Mdt=0;Idt=0; deltaT=Tspan(end)/nFrames;
for FrameNo = 1:nFrames % numerical integration of f(t,M)=delta(torque)*delta(time)
     idt= idt+abs(i(FrameNo)^2)*deltaT;
     Tdt= Tdt+abs(MT(FrameNo)^2)*deltaT;
     Mdt= Mdt+JT(FrameNo)^2*deltaT;
     Idt=Idt+I(FrameNo)^2*deltaT;
end
Irms=sqrt(1/Tspan(end)*idt);
Trms=sqrt(1/Tspan(end)*Tdt);
Mrms=sqrt(1/Tspan(end)*Mdt);
Arms=sqrt(1/Tspan(end)*Idt);
% save data to file
for l=1:2
Power(R(l)).PlossMean=PlossMean;            % [W] total effect loss from motor
Power(R(l)).PlossMax=PlMax;                 % Max loss from motor
Power(R(l)).PmecMean=mean(abs(MT))*mean(abs(Omega));  % [W] total mechanic effect used
Power(R(l)).PmecMax=PmMax;                  % Max mechanical power
Power(R(l)).IrmsMotor=Irms;                 % Nominal current
Power(R(l)).ImaxMotor=max(abs(i));          % Max motor current
Power(R(l)).IbatMax=IbatMax;
Power(R(l)).Imotor=i; % [A]
Power(R(l)).PmecW=Pmec;
Power(R(l)).Pjoint=Pjoint;
Power(R(l)).PlossW=Ploss;
Power(R(l)).JV=JV;                          % motor speed
Power(R(l)).MT=MT;                          % Motor torque
Power(R(l)).JT=JT;
Power(R(l)).SC=SC;                          % Safety constant
Power(R(l)).NomTorque=NomTorque;
Power(R(l)).Mrms=Mrms;                      % Motor torque
Power(R(l)).Arms=Arms;
end
save DCMotorData/Power.mat Power
Overshoot= ceil((Tmax-NT)*100/NT);
```

```
function [Psi1, Psi2]=Psi
Psi1=zeros(1,33);
Psi2=zeros(1,33);

Psi1(1,19)=4;
Psi2(1,19)=25;

Psi1(1,25)=27;
Psi2(1,25)=24;

Psi1(1,29)=6;
Psi2(1,29)=12;


Psi1(1,17)=Psi1(1,29);
Psi2(1,17)=Psi2(1,29);
Psi1(1,13)=Psi1(1,25);
Psi2(1,13)=Psi2(1,25);
Psi1(1,31)=Psi1(1,19);
Psi2(1,31)=Psi2(1,19);
```

```
% Servomotor; Maxon EC-serie

% 2x RE 40 - 150W (order number 148877)
Pm = 300;           % W
Vn = 48;            % volt
n0 = 7580;          % no load speed [rpm]
Mh = 5;             % stall torque [Nm]
ST = 1502;          % speed/torque gradient [rpm/Nm]
I0 = 0.0537;        % no load current [A]
MS = 12000;         % Max. speed
MCC = 3.12;         % Max. continuous current [A]
MCT = 368*10^-3;    % Max. continuous torque [Nm]
Me = 0.92;          % Max.efficiency
Km = 120.6*10^-3;   % torque constant [Nm/A]
Kn = 134;           % speed constant [rpm/V]
tc = 4.36;          % Mechanical time constant [ms]
Jr = 129*10^-7;     % rotor inertia [kg*m^2]
m  = 0.960;         % weight of motor [kg]
ON = 148877;        % Order number
RE = 3;             % If EC motor =1, RE motor =2, 2xRE motor=3
As = 82.8;          % Starting current

servo(1).EC = [Pm Vn n0 Mh ST I0 MS MCC MCT Me Km Kn tc Jr m ON RE As];

% RE 35 - 90W (order number 273755)
Pm = 90;            % W
Vn = 48;            % volt
n0 = 7270;          % no load speed [rpm]
Mh = 0.967;         % stall torque [Nm]
ST = 7620;          % speed/torque gradient [rpm/Nm]
I0 = 0.077;         % no load current [A]
MS = 12000;         % Max. speed
MCC = 1.63;         % Max. continuous current [A]
MCT = 96.5*10^-3;   % Max. continuous torque [Nm]
Me = 0.85;          % Max.efficiency
Km = 62.2*10^-3;    % torque constant [Nm/A]
Kn = 154;           % speed constant [rpm/V]
tc = 5.38;          % Mechanical time constant [ms]
Jr = 67.4*10^-7;    % rotor inertia [kg*m^2]
m  = 0.340;         % weight of motor [kg]
ON = 273755;        % Order number
RE = 2;             % If EC motor =1, RE motor =2
As = 15.5 ;         % starting current
servo(2).EC = [Pm Vn n0 Mh ST I0 MS MCC MCT Me Km Kn tc Jr m ON RE As];


% RE 30 - 60W (order number 310009)
Pm = 60;            % W
Vn = 48;            % volt
n0 = 8490;          % no load speed [rpm]
Mh = 1.020;         % stall torque [Nm]
ST = 8330;          % speed/torque gradient [rpm/Nm]
I0 = 78.5*10^-3;    % no load current [A]
MS = 12000;         % Max. speed
MCC = 1.72;         % Max. continuous current [A]
MCT = 88.2*10^-3;   % Max. continuous torque [Nm]
```

```
Me = 0.88;          % Max.efficiency
Km = 53.8*10^-3;    % torque constant [Nm/A]
Kn = 178;           % speed constant [rpm/V]
tc = 3.01;          % Mechanical time constant [ms]
Jr = 33.3*10^-7;    % rotor inertia [kg*m^2]
m  = 0.238;         % weight of motor [kg]
ON = 310009;        % Order number
RE = 2;             % If EC motor =1, RE motor =2
As = 39.3 ;         % starting current
servo(3).EC = [Pm Vn n0 Mh ST I0 MS MCC MCT Me Km Kn tc Jr m ON RE As];

% RE 40 - 150W (order number 148877)
Pm = 150;           % W
Vn = 48;            % volt
n0 = 7580;          % no load speed [rpm]
Mh = 2.5;           % stall torque [Nm]
ST = 3040;          % speed/torque gradient [rpm/Nm]
I0 = 0.0312;        % no load current [A]
MS = 12000;         % Max. speed
MCC = 3.12;         % Max. continuous current [A]
MCT = 184*10^-3;    % Max. continuous torque [Nm]
Me = 0.92;          % Max.efficiency
Km = 60.3*10^-3;    % torque constant [Nm/A]
Kn = 158;           % speed constant [rpm/V]
tc = 4.39;          % Mechanical time constant [ms]
Jr = 138*10^-7;     % rotor inertia [kg*m^2]
m  = 0.480;         % weight of motor [kg]
ON = 148877;        % Order number
RE = 2;             % If EC motor =1, RE motor =2
As = 41.4 ;         % starting current
servo(4).EC = [Pm Vn n0 Mh ST I0 MS MCC MCT Me Km Kn tc Jr m ON RE As];

noServo = length(servo);

for i=1:noServo
    servo(i).J= servo(i).EC(14);
end
```

```
function [RMST]=RMStorque(Joint,JointNo)

MT=Joint(JointNo).MT;
T=length(MT);
Tdt=0; deltaT=1/T;
for FrameNo = 1:T % numerical integration of f(t,M)=delta(torque)*delta(time)
     Tdt= Tdt+MT(FrameNo)^2*deltaT;
end
RMST=sqrt(1/T*Tdt);
```

```
% Timing-belt and pulley data

%%%%%%% Timing-belt %%%%%%
TimEff = 0.98;

%%%%%%% Pulley %%%%%%

% Number of teeth
Pulley(1).Te = 18; % [teeth]
Pulley(2).Te = 21;
Pulley(3).Te = 26;
Pulley(4).Te = 48;
Pulley(5).Te = 57;
Pulley(6).Te = 63;
Pulley(7).Te = 18;
Pulley(8).Te = 22;
Pulley(9).Te = 30;
Pulley(10).Te = 34;
Pulley(11).Te = 38;
Pulley(12).Te = 41;
Pulley(13).Te = 44;
Pulley(14).Te = 56;
Pulley(15).Te = 57;
Pulley(16).Te = 62;
Pulley(17).Te = 72;
Pulley(18).Te = 20;
Pulley(19).Te = 26;
Pulley(20).Te = 24;
Pulley(21).Te = 32;
Pulley(22).Te = 16;
% Pulley width
Belt= 15*10^-3; % [m]
% Density of alu
Rho= 2.7*10^3; % [kg/m^3]

% Diameter of pulley
Pulley(1).D = 17.19*10^-3; % [m]
Pulley(2).D = 13.36*10^-3;
Pulley(3).D = 16.55*10^-3;
Pulley(4).D = 30.55*10^-3;
Pulley(5).D = 36.28*10^-3;
Pulley(6).D = 40.1*10^-3;
Pulley(7).D = 17.19*10^-3;
Pulley(8).D = 21.01*10^-3;
Pulley(9).D = 28.65*10^-3;
Pulley(10).D= 32.47*10^-3;
Pulley(11).D = 36.29*10^-3;
Pulley(12).D = 39.15*10^-3;
Pulley(13).D = 42.2*10^-3;
Pulley(14).D = 54.43*10^-3;
Pulley(15).D = 54.48*10^-3;
Pulley(16).D = 59.21*10^-3;
Pulley(17).D = 68.75*10^-3;
Pulley(18).D = 19.10*10^-3;
Pulley(19).D = 24.83*10^-3;
Pulley(20).D = 38.2*10^-3;
```

```
Pulley(21).D = 50.93*10^-3;
Pulley(22).D = 15.28*10^-3;
PulleyNo=length(Pulley);
for i=1:PulleyNo
% Weight of pulley
Pulley(i).m = Belt*pi/4*(Pulley(i).D)^2*Rho; % [kg]

% inertia of pulley
Pulley(i).J = 1/8*Pulley(1).m*Pulley(i).D^2; % [kg*m^2]

end
```

```
% Forward Dynamic Analysis by DMS10/gr.43A
% Simulates robot in time domain.
clear all; close all; clc;

% Set global status for structs
global B RJ F E S cst Results

% Constants struct: cst
cst.nSteps            = 4400;             % Total number of time steps.
cst.dt                = 0.25e-3;          % Time increment.
cst.nBodies           = 37;               % Number of bodies. 1..37
cst.nRJs              = cst.nBodies-1;    % Number of revolute joints.
cst.MotorsOn          = 1;                % Turn motors on or off, 0/1.
cst.SaveSteps         = 1;                % Save results inteval.
cst.Angles            = 'BryantAngles';   % Basebody orientation.
cst.g                 = 9.82;             % Acceleration due to gravity.
cst.LockRJs           = 0;                % Lock all RJs, 1/0.
cst.FootForces        = 1;                % Apply foot forces 1/0.
cst.BendKnees         = 15;               % Half knee angle [deg].
S.nSteps              = 1;                % Number of robot steps.

% Initialize inertia, geometry, connections and so forth
DefineRobot2;

% Set initial basebody values
B(1).Theta = [0 0 0]';
B(1).r     = [-0.0097 0 cst.PelvisHeight-0.00055]';
B(1).omega = [0 0 0]';
B(1).rD    = [0 0 0]';
B(1).rDD   = [0 0 0]';
B(1).omegaD= [0 0 0]';

% Vectors for integration information
qD1=0; qDD1=0;

% Generate trajectories for control
MotionPlanning();
InverseKinematics();

% Time-loop
StartTime=cputime;
for tStep=1:cst.nSteps

    % Update time and write tStep to screen
    t=cst.dt*(tStep-1);
    if rem(tStep,10)==0
        clc
        fprintf('Time step: %i/%i', tStep,cst.nSteps)
        fprintf('\nEnergy deviation: %3.4fJ\n',E.Etot)
    end

    % Update kinematics
    CalcA(); % Calculate A-matrix for all bodies
    CalcPos(); % Calculate r-vector for all bodies
    CalcOmega(); % Calculate angular velocity of all bodies
    CalcVel(); % Calculate velocity of all bodies
```

```
    % Regulation, determines motor torques from control-input
    Regulator2(t,tStep);

    % Calculate foot forces
    ConstrainFeet(tStep);

    % Calculates energy balance
    EnergyBalance2(tStep);

    % Calculate gamma-terms for RHS of EOMs
    CalcGamma();

    % Setup and solve EOMs
    y = Setupy(tStep);
    C = SetupC();
    x = C\y;

    % Extract accelerations and velocities for integration
    qDD = ExtractAccelerations(x);
    qD  = ExtractVelocities();

    % Perform time integration
    TrapezoidIntegration(tStep,qD,qD1,qDD,qDD1);
    qD1=qD; qDD1=qDD; % Store old values for next integration

    % Sort results
    SortResults(x,qDD);

    % Calculate cartesian accelerations of bodies
    CalcAccelerations();

    % Calculate balance information
    cst.r_com=CalcCoM();
    cst.r_zmp=CalcZMP();

    % Save selected results
    SaveResults2(tStep,cst.SaveSteps);
end

% Save time invariant results
CstResults=SaveConstants();

% Analysis done
fprintf('Elapsed time: %3.0fs\n',cputime-StartTime)

% PLOT RESULTS:
%PlotEnergyLevels2;
%PlotMotorData;
%Animator2;
%PlotPsis;
%PlotConstrainingForces
```

```
function []=DefineRobot2()
%DefineRobot2.m

global cst RJ B F

% Body names
B(1).name = 'Pelvis';
B(2).name = 'Waist yaw motor';
B(3).name = 'Waist pitch joint';
B(4).name = 'Waist pitch motor';
B(5).name = 'Waist cross';
B(6).name = 'Waist roll motor';
B(7).name = 'Torso';
B(8).name = 'Left shoulder motor';
B(9).name = 'Left arm';
B(10).name = 'Right shoulder motor';
B(11).name = 'Right arm';
B(12).name = 'Left hip yaw motor';
B(13).name = 'Left hip roll joint';
B(14).name = 'Left hip roll motor';
B(15).name = 'Left hip cross joint';
B(16).name = 'Left hip pitch motor';
B(17).name = 'Left thigh';
B(18).name = 'Left knee motor';
B(19).name = 'Left shin';
B(20).name = 'Left ankle pitch motor';
B(21).name = 'Left ankle cross joint';
B(22).name = 'Left ankle roll motor';
B(23).name = 'Left foot';
B(24).name = 'Right hip yaw motor';
B(25).name = 'Right roll hip joint';
B(26).name = 'Right hip roll motor';
B(27).name = 'Right hip cross joint';
B(28).name = 'Right hip pitch motor';
B(29).name = 'Right thigh';
B(30).name = 'Right knee motor';
B(31).name = 'Right shin';
B(32).name = 'Right ankle pitch motor';
B(33).name = 'Right ankle cross joint';
B(34).name = 'Right ankle roll motor';
B(35).name = 'Right foot';
B(36).name = 'Left toe';
B(37).name = 'Right toe';

% Determines veather the body is a motor or not 1 = motor, 0 = not motor.
B(1).motor = 0;
B(2).motor = 1;
B(3).motor = 0;
B(4).motor = 1;
B(5).motor = 0;
B(6).motor = 1;
B(7).motor = 0;
B(8).motor = 1;
B(9).motor = 0;
B(10).motor = 1;
B(11).motor = 0;
```

```
B(12).motor = 1;
B(13).motor = 0;
B(14).motor = 1;
B(15).motor = 0;
B(16).motor = 1;
B(17).motor = 0;
B(18).motor = 1;
B(19).motor = 0;
B(20).motor = 1;
B(21).motor = 0;
B(22).motor = 1;
B(23).motor = 0;
B(24).motor = 1;
B(25).motor = 0;
B(26).motor = 1;
B(27).motor = 0;
B(28).motor = 1;
B(29).motor = 0;
B(30).motor = 1;
B(31).motor = 0;
B(32).motor = 1;
B(33).motor = 0;
B(34).motor = 1;
B(35).motor = 0;
B(36).motor = 0;
B(37).motor = 0;

% Vectors with numbers of the RJ's, attached to the body
B(1).RJs = [1 2 11 12 23 24];
B(2).RJs = [1];
B(3).RJs = [2 3 4];
B(4).RJs = [3];
B(5).RJs = [4 6];
B(6).RJs = [5];
B(7).RJs = [5 6 7 8 9 10];
B(8).RJs = [7];
B(9).RJs = [8];
B(10).RJs = [9];
B(11).RJs = [10];
B(12).RJs = [11];
B(13).RJs = [12 13 14];
B(14).RJs = [13];
B(15).RJs = [14 16];
B(16).RJs = [15];
B(17).RJs = [15 16 17 18];
B(18).RJs = [17];
B(19).RJs = [18 19 20];
B(20).RJs = [19];
B(21).RJs = [20 22];
B(22).RJs = [21];
B(23).RJs = [21 22];
B(24).RJs = [23];
B(25).RJs = [24 25 26];
B(26).RJs = [25];
B(27).RJs = [26 28];
B(28).RJs = [27];
```

```
B(29).RJs = [27 28 29 30];
B(30).RJs = [29];
B(31).RJs = [30 31 32];
B(32).RJs = [31];
B(33).RJs = [32 34];
B(34).RJs = [33];
B(35).RJs = [33 34];
B(36).RJs = [35];
B(37).RJs = [36];

% Updated inertia- and geometric properties.
% Mass: [kg], Inertia: [kg*m2], Geometry: [m]
% 1: Pelvis
Ixx = 0.0547; Iyy = 0.0129; Izz = 0.0657;
B(1).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(1).m = 4.9;
s{2,1} = [-80 0 4]'*10^-3;
s{3,1} = [22 0 14]'*10^-3;
s{24,1} = [-96 -77 4]'*10^-3;
s{25,1} = [22 -140 -16]'*10^-3;
s{12,1} = [-96 77 4]'*10^-3;
s{13,1} = [22 140 -16]'*10^-3;

% 3: Waist pitch joint
Ixx = 0.00988; Iyy = 0.00678; Izz = 0.00587;
B(3).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(3).m = 2.27;
s{1,3} = [4 38 -80]'*10^-3;
s{4,3} = [-16 -22 -32]'*10^-3;
s{5,3} = [4 38 40]'*10^-3;

% 5: Waist cross joint
Ixx = 0.000349; Iyy = 0.000956; Izz = 0.00121;
B(5).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(5).m = 0.75;
s{3,5} = [0 0 0]'*10^-3;
s{7,5} = [0 0 0]'*10^-3;

% 7: Torso
Ixx = 0.826; Iyy = 0.586; Izz = 0.502;
B(7).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(7).m = 21.6;
s{5,7} = [19 0 -341]'*10^-3;
s{6,7} = [-54 0 -211]'*10^-3;
s{10,7} = [75 -222 84]'*10^-3;
s{11,7} = [25 -280 84]'*10^-3;
s{8,7} = [75 222 84]'*10^-3;
s{9,7} = [25 280 84]'*10^-3;

% 9: Left arm
Ixx = 0.0307; Iyy = 0.0419; Izz = 0.0125;
B(9).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(9).m = 0.9;
s{7,9} = [-4 -48 347]'*10^-3;

% 11: Right arm
```

```
Ixx = 0.0307; Iyy = 0.0419; Izz = 0.0125;
B(11).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(11).m = 0.9;
s{7,11} = [-4 48 347]'*10^-3;

% 25: Right hip pitch joint
Ixx = 0.0161; Iyy = 0.0127; Izz = 0.00932;
B(25).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(25).m = 3.07;
s{1,25} = [38 24 28]'*10^-3;
s{26,25} = [-24 -54 77]'*10^-3;
s{27,25} = [41 24 -36]'*10^-3;

% 13: Left hip pitch joint
Ixx = 0.0161; Iyy = 0.0127; Izz = 0.00932;
B(13).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(13).m = 3.07;
s{1,13} = [38 -24 28]'*10^-3;
s{14,13} = [-24 54 77]'*10^-3;
s{15,13} = [41 -24 -36]'*10^-3;

% 27: Right hip cross joint
Ixx = 0.00047; Iyy = 0.000324; Izz = 0.000644;
B(27).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(27).m = 0.64;
s{25,27} = [0 0 0]'*10^-3;
s{29,27} = [0 0 0]'*10^-3;

% 15: Left hip cross joint
Ixx = 0.00047; Iyy = 0.000324; Izz = 0.000644;
B(15).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(15).m = 0.64;
s{13,15} = [0 0 0]'*10^-3;
s{17,15} = [0 0 0]'*10^-3;

% 29: Right thigh
Ixx = 0.099; Iyy = 0.0894; Izz = 0.024;
B(29).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(29).m = 6.66;
s{27,29} = [2 41 146]'*10^-3;
s{28,29} = [-19 -27 37]'*10^-3;
s{30,29} = [13 -27 -34]'*10^-3;
s{31,29} = [2 27+14 -165]'*10^-3;

% 17: Left thigh
Ixx = 0.099; Iyy = 0.0894; Izz = 0.024;
B(17).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(17).m = 6.66;
s{15,17} = [2 -41 146]'*10^-3;
s{16,17} = [-19 27 37]'*10^-3;
s{18,17} = [13 27 -34]'*10^-3;
s{19,17} = [2 -27-14 -165]'*10^-3;

% 31: Right shin
Ixx = 0.0726; Iyy = 0.0686; Izz = 0.013;
B(31).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
```

```
B(31).m = 4.97;
s{29,31} = [-4 -30 252]'*10^-3;
s{32,31} = [21 25 -13]'*10^-3;
s{33,31} = [-4 -30 -118]'*10^-3;

% 19: Left shin
Ixx = 0.0726; Iyy = 0.0686; Izz = 0.013;
B(19).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(19).m = 4.97;
s{17,19} = [-4 30 252]'*10^-3;
s{20,19} = [21 -25 -13]'*10^-3;
s{21,19} = [-4 30 -118]'*10^-3;

% 33: Right ankle cross joint
Ixx = 0.000181; Iyy = 0.000358; Izz = 0.000434;
B(33).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(33).m = 0.47;
s{31,33} = [0 0 0]'*10^-3;
s{35,33} = [0 0 0]'*10^-3;

% 21: Left ankle cross joint
Ixx = 0.000181; Iyy = 0.000358; Izz = 0.000434;
B(21).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(21).m = 0.47;
s{19,21} = [0 0 0]'*10^-3;
s{23,21} = [0 0 0]'*10^-3;

% 35: Right foot
Ixx = 0.00789; Iyy = 0.0167; Izz = 0.0145;
B(35).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(35).m = 2.75;
s{33,35} = [19 10 46]'*10^-3;
s{34,35} = [0 -30 46]'*10^-3;
s{37,35} = [132 10 -76]'*10^-3;

% 23: Left foot
Ixx = 0.00789; Iyy = 0.0167; Izz = 0.0145;
B(23).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(23).m = 2.75;
s{21,23} = [19 -10 46]'*10^-3;
s{22,23} = [0 30 46]'*10^-3;
s{36,23} = [132 10 -76]'*10^-3;

% 37: Right toe
Ixx = 0.000139; Iyy = 0.000426; Izz = 0.000312;
B(37).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(37).m=0.015;
s{35,37} = [0 0 0]'*10^-3;

% 36: Left toe
Ixx = 0.000139; Iyy = 0.000426; Izz = 0.000312;
B(36).J = [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
B(36).m=0.015;
s{23,36} = [0 0 0]'*10^-3;

% Motors
```

```
s{1,2}=[0 0 0]'*10^-3;
s{3,4}=[0 0 0]'*10^-3;
s{7,6}=[0 0 0]'*10^-3;
s{7,8}=[0 0 0]'*10^-3;
s{7,10}=[0 0 0]'*10^-3;
s{1,12}=[0 0 0]'*10^-3;
s{13,14}=[0 0 0]'*10^-3;
s{17,16}=[0 0 0]'*10^-3;
s{17,18}=[0 0 0]'*10^-3;
s{19,20}=[0 0 0]'*10^-3;
s{23,22}=[0 0 0]'*10^-3;
s{1,24}=[0 0 0]'*10^-3;
s{25,26}=[0 0 0]'*10^-3;
s{29,28}=[0 0 0]'*10^-3;
s{29,30}=[0 0 0]'*10^-3;
s{31,32}=[0 0 0]'*10^-3;
s{35,34}=[0 0 0]'*10^-3;

% Power transmission specification
m_60=0.283; m_90=0.340; m_150=0.480; m_2x150=0.960; % rotor mass
J_60=33.5e-7; J_90=67.6e-7; J_150=138e-7; J_2x150=2*138e-7; % rotor inertia
J_14=0.091e-4; J_17=0.193e-4; J_20=0.404e-4; % gear inertia
t_60=3.01*cst.dt; t_90=5.38*cst.dt; t_150=4.39*cst.dt; % time constants
I=eye(3);
% Mmax= 4x nominal motor torque [Nm], N0= motor nominal speed [rpm]

% Waist yaw
B(2).Mmax= 4*88.2/1000;
B(2).N0  = 7750;
B(2).m   = m_60;
B(2).t   = t_60;
B(2).J   = (J_17+J_60+0.108e-4)*I;
RJ(2).u  = 100*48/16;
RJ(2).mu = 1.0;

% Waist pitch
B(4).m   = m_150;
B(4).t   = t_150;
B(4).N0  = 7000;
B(4).Mmax= 4*184/1000;
B(4).J   = (J_17+J_150+0.3171e-4)*I;
RJ(4).u  = 120*56/18;
RJ(4).mu = 1.0;

% Waist roll
B(6).m   = m_90;
B(6).t   = t_90;
B(6).Mmax= 4*96.5/1000;
B(6).N0  = 6490;
B(6).J   = (J_14+J_90+0.2911e-4)*I;
RJ(6).u  = 100*62/18;
RJ(6).mu = 1.0;

% Left shoulder pitch
B(8).m   = m_60;
B(8).t   = t_60;
```

```
B(8).N0  = 7750;
B(8).Mmax= 4*88.2/1000;
B(8).J   = (J_14+J_60+0.0051e-4)*I;
RJ(8).u  = 100*20/18;
RJ(8).mu = 1.0;

% Right shoulder pitch
B(10).m   = m_60;
B(10).t   = t_60;
B(10).N0  = 7750;
B(10).Mmax= 4*88.2/1000;
B(10).J   = (J_14+J_60+0.0051e-4)*I;
RJ(10).u  = 100*20/18;
RJ(10).mu = 1.0;

% Right hip yaw
B(24).m   = m_60;
B(24).t   = t_60;
B(24).N0  = 7750;
B(24).Mmax= 4*88.2/1000;
B(24).J   = (J_17+J_60+0.0510e-4)*I;
RJ(24).u  = 100*40/16;
RJ(24).mu = 1.0;

% Left hip yaw
B(12).m   = m_60;
B(12).t   = t_60;
B(12).N0  = 7750;
B(12).Mmax= 4*88.2/1000;
B(12).J   = (J_17+J_60+0.0510e-4)*I;
RJ(12).u  = 100*40/16;
RJ(12).mu = 1.0;

% Right hip roll
B(26).m   = m_2x150;
B(26).t   = t_150;
B(26).Mmax= 4*2*184/1000;
B(26).N0  = 7000;
B(26).J   = (J_20+J_2x150+0.5576e-4)*I;
RJ(26).u  = 120*72/30;
RJ(26).mu = 1.0;

% Left hip roll
B(14).m   = m_2x150;
B(14).t   = t_150;
B(14).Mmax= 4*2*184/1000;
B(14).N0  = 7000;
B(14).J   = (J_20+J_2x150+0.5576e-4)*I;
RJ(14).u  = 120*72/30;
RJ(14).mu = 1.0;

% Right hip pitch
B(28).m   = m_150;
B(28).t   = t_150;
B(28).Mmax= 4*2*184/1000;
B(28).N0  = 7000;
```

```
B(28).J   = (J_20+J_150+0.2937e-4)*I;
RJ(28).u  = 160*60/28;
RJ(28).mu = 1.0;

% Left hip pitch
B(16).m   = m_150;
B(16).t   = t_150;
B(16).Mmax= 4*2*184/1000;
B(16).N0  = 7000;
B(16).J   = (J_20+J_150+0.2937e-4)*I;
RJ(16).u  = 160*60/28;
RJ(16).mu = 1.0;

% Right knee pitch
B(30).m   = m_2x150;
B(30).t   = t_150;
B(30).Mmax= 4*2*184/1000;
B(30).N0  = 7000;
B(30).J   = (J_17+J_2x150+0.2111e-4)*I;
RJ(30).u  = 100*32/24;
RJ(30).mu = 1.0;

% Left knee pitch
B(18).m   = m_2x150;
B(18).t   = t_150;
B(18).Mmax= 4*2*184/1000;
B(18).N0  = 7000;
B(18).J   = (J_17+J_2x150+0.2111e-4)*I;
RJ(18).u  = 100*32/24;
RJ(18).mu = 1.0;

% Right ankle pitch
B(32).m   = m_2x150;
B(32).t   = t_150;
B(32).Mmax= 4*2*184/1000;
B(32).N0  = 7000;
B(32).J   = (J_20+J_2x150+0.0421e-4)*I;
RJ(32).u  = 120*38/18;
RJ(32).mu = 1.0;

% Left ankle pitch
B(20).m   = m_2x150;
B(20).t   = t_150;
B(20).Mmax= 4*2*184/1000;
B(20).N0  = 7000;
B(20).J   = (J_20+J_2x150+0.0421e-4)*I;
RJ(20).u  = 120*38/18;
RJ(20).mu = 1.0;

% Right ankle roll
B(34).m   = m_90;
B(34).t   = t_90;
B(34).Mmax= 4*96.5/1000;
B(34).N0  = 6490;
B(34).J   = (J_14+J_90+0.0789e-4)*I;
RJ(34).u  = 100*44/22;
```

```
RJ(34).mu = 1.0;

% Left ankle roll
B(22).m   = m_90;
B(22).t   = t_90;
B(22).Mmax= 4*96.5/1000;
B(22).N0  = 6490;
B(22).J   = (J_14+J_90+0.0789e-4)*I;
RJ(22).u  = 100*44/22;
RJ(22).mu = 1.0;

% No gearing or actuation of toes:
RJ(35).u=1;
RJ(36).u=1;

%Gear efficiency
motors=[2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34];
for i=1:length(motors)
    j=motors(i);
    RJ(j).mu=0.65; % gear efficiency
    B(j).Mm=0; %set motor initial torques zero
end


% save:
cst.heel = [-62 10 -76]'*10^-3;
cst.toes = [0 60 0]'*10^-3;
cst.hand = [223 44 -184]'*10^-3;
cst.s=s;

% Feet data
F(1).name = 'Left foot';
F(2).name = 'Right foot';
F(1).bodyNo = 23;
F(2).bodyNo = 35;
F(1).contact=1; %initial contact state
F(2).contact=1;
cst.ContactShift = 0; % vector containing time steps of SSP/DSP shifts

% Revolute Joint Names
RJ(1).name = 'Waist yaw motor';
RJ(2).name = 'Waist yaw joint';
RJ(3).name = 'Waist pitch motor';
RJ(4).name = 'Waist pitch joint';
RJ(5).name = 'Waist roll motor';
RJ(6).name = 'Waist roll joint';
RJ(7).name = 'Left shoulder motor';
RJ(8).name = 'Left shoulder joint';
RJ(9).name = 'Right shoulder motor';
RJ(10).name = 'Right shoulder joint';
RJ(11).name = 'Left hip yaw motor';
RJ(12).name = 'Left hip yaw joint';
RJ(13).name = 'Left hip roll motor';
RJ(14).name = 'Left hip roll joint';
RJ(15).name = 'Left hip pitch motor';
RJ(16).name = 'Left hip pitch joint';
```

```
RJ(17).name = 'Left knee motor';
RJ(18).name = 'Left knee joint';
RJ(19).name = 'Left ankle pitch motor';
RJ(20).name = 'Left ankle pitch joint';
RJ(21).name = 'Left ankle roll motor';
RJ(22).name = 'Left ankle roll joint';
RJ(23).name = 'Right hip yaw motor';
RJ(24).name = 'Right hip yaw joint';
RJ(25).name = 'Right hip roll motor';
RJ(26).name = 'Right hip roll joint';
RJ(27).name = 'Right hip pitch motor';
RJ(28).name = 'Right hip pitch joint';
RJ(29).name = 'Right knee motor';
RJ(30).name = 'Right knee joint';
RJ(31).name = 'Right ankle pitch motor';
RJ(32).name = 'Right ankle pitch joint';
RJ(33).name = 'Right ankle roll motor';
RJ(34).name = 'Right ankle roll joint';
RJ(35).name = 'Left toe';
RJ(36).name = 'Right toe';

% Body closest to basebody in RJ
RJ(1).a = 1;
RJ(2).a = 1;
RJ(3).a = 3;
RJ(4).a = 3;
RJ(5).a = 7;
RJ(6).a = 5;
RJ(7).a = 7;
RJ(8).a = 7;
RJ(9).a = 7;
RJ(10).a = 7;
RJ(11).a = 1;
RJ(12).a = 1;
RJ(13).a = 13;
RJ(14).a = 13;
RJ(15).a = 17;
RJ(16).a = 15;
RJ(17).a = 17;
RJ(18).a = 17;
RJ(19).a = 19;
RJ(20).a = 19;
RJ(21).a = 23;
RJ(22).a = 21;
RJ(23).a = 1;
RJ(24).a = 1;
RJ(25).a = 25;
RJ(26).a = 25;
RJ(27).a = 29;
RJ(28).a = 27;
RJ(29).a = 29;
RJ(30).a = 29;
RJ(31).a = 31;
RJ(32).a = 31;
RJ(33).a = 35;
RJ(34).a = 33;
```

```
RJ(35).a = 23;
RJ(36).a = 35;

% Body farthest from basebody in RJ
RJ(1).b = 2;
RJ(2).b = 3;
RJ(3).b = 4;
RJ(4).b = 5;
RJ(5).b = 6;
RJ(6).b = 7;
RJ(7).b = 8;
RJ(8).b = 9;
RJ(9).b = 10;
RJ(10).b = 11;
RJ(11).b = 12;
RJ(12).b = 13;
RJ(13).b = 14;
RJ(14).b = 15;
RJ(15).b = 16;
RJ(16).b = 17;
RJ(17).b = 18;
RJ(18).b = 19;
RJ(19).b = 20;
RJ(20).b = 21;
RJ(21).b = 22;
RJ(22).b = 23;
RJ(23).b = 24;
RJ(24).b = 25;
RJ(25).b = 26;
RJ(26).b = 27;
RJ(27).b = 28;
RJ(28).b = 29;
RJ(29).b = 30;
RJ(30).b = 31;
RJ(31).b = 32;
RJ(32).b = 33;
RJ(33).b = 34;
RJ(34).b = 35;
RJ(35).b = 36;
RJ(36).b = 37;


% Determine whether the joint is: not driven 0, driven 1 or driving 2.
% RJ type: 0= unactuated, 1= next body on gear output shaft,
%          2= motor RJ,   3= next body on gear stator
RJ(1).drive = 2;
RJ(2).drive = 1;
RJ(3).drive = 2;
RJ(4).drive = 1;
RJ(5).drive = 2;
RJ(6).drive = 3;
RJ(7).drive = 2;
RJ(8).drive = 1;
RJ(9).drive = 2;
RJ(10).drive = 1;
RJ(11).drive = 2;
```

```
RJ(12).drive = 1;
RJ(13).drive = 2;
RJ(14).drive = 1;
RJ(15).drive = 2;
RJ(16).drive = 3;
RJ(17).drive = 2;
RJ(18).drive = 1;
RJ(19).drive = 2;
RJ(20).drive = 1;
RJ(21).drive = 2;
RJ(22).drive = 3;
RJ(23).drive = 2;
RJ(24).drive = 1;
RJ(25).drive = 2;
RJ(26).drive = 1;
RJ(27).drive = 2;
RJ(28).drive = 3;
RJ(29).drive = 2;
RJ(30).drive = 1;
RJ(31).drive = 2;
RJ(32).drive = 1;
RJ(33).drive = 2;
RJ(34).drive = 3;
RJ(35).drive = 0;
RJ(36).drive = 0;

% Unit vector describing axis of rotation, in preveious body CS.
RJ(1).e = [0 0 1]';
RJ(2).e = [0 0 1]';
RJ(3).e = [0 1 0]';
RJ(4).e = [0 1 0]';
RJ(5).e = [1 0 0]';
RJ(6).e = [1 0 0]';
RJ(7).e = [0 1 0]';
RJ(8).e = [0 1 0]';
RJ(9).e = [0 1 0]';
RJ(10).e = [0 1 0]';
RJ(11).e = [0 0 1]';
RJ(12).e = [0 0 1]';
RJ(13).e = [1 0 0]';
RJ(14).e = [1 0 0]';
RJ(15).e = [0 1 0]';
RJ(16).e = [0 1 0]';
RJ(17).e = [0 1 0]';
RJ(18).e = [0 1 0]';
RJ(19).e = [0 1 0]';
RJ(20).e = [0 1 0]';
RJ(21).e = [1 0 0]';
RJ(22).e = [1 0 0]';
RJ(23).e = [0 0 1]';
RJ(24).e = [0 0 1]';
RJ(25).e = [1 0 0]';
RJ(26).e = [1 0 0]';
RJ(27).e = [0 1 0]';
RJ(28).e = [0 1 0]';
RJ(29).e = [0 1 0]';
```

```
RJ(30).e = [0 1 0]';
RJ(31).e = [0 1 0]';
RJ(32).e = [0 1 0]';
RJ(33).e = [1 0 0]';
RJ(34).e = [1 0 0]';
RJ(35).e = [0 1 0]';
RJ(36).e = [0 1 0]';

% set initial gear output torque zero
for k=1:cst.nRJs
   if RJ(k).drive==1 || RJ(k).drive==3
      RJ(k).Mg = 0;
   end
end

% RJ number in chain: m, counted from pelvis.
RJ(1).nJ = 1;
RJ(2).nJ = 1;
RJ(3).nJ = 3;
RJ(4).nJ = 2;
RJ(5).nJ = 4;
RJ(6).nJ = 3;
RJ(7).nJ = 4;
RJ(8).nJ = 4;
RJ(9).nJ = 4;
RJ(10).nJ = 4;
RJ(11).nJ = 1;
RJ(12).nJ = 1;
RJ(13).nJ = 2;
RJ(14).nJ = 2;
RJ(15).nJ = 4;
RJ(16).nJ = 3;
RJ(17).nJ = 4;
RJ(18).nJ = 4;
RJ(19).nJ = 5;
RJ(20).nJ = 5;
RJ(21).nJ = 7;
RJ(22).nJ = 6;
RJ(23).nJ = 1;
RJ(24).nJ = 1;
RJ(25).nJ = 2;
RJ(26).nJ = 2;
RJ(27).nJ = 4;
RJ(28).nJ = 3;
RJ(29).nJ = 4;
RJ(30).nJ = 4;
RJ(31).nJ = 5;
RJ(32).nJ = 5;
RJ(33).nJ = 7;
RJ(34).nJ = 6;
RJ(35).nJ = 7;
RJ(36).nJ = 7;

% Vectors to previous (sa) and next (sb) body's COM from  k'th RJ
for k=1:cst.nRJs
   a=RJ(k).a;
```

```
   b=RJ(k).b;
   RJ(k).sa=-cst.s{b,a};
   RJ(k).sb=-cst.s{a,b};
end

% Set drivenby and drives status for RJs
% RJ(1).name = 'Waist yaw motor';
RJ(1).drives = 2;
RJ(1).drivenBody = 3;
% RJ(2).name = 'Waist yaw joint';
RJ(2).drivenby =1;
% RJ(3).name = 'Waist pitch motor';
RJ(3).drives = 4;
RJ(3).drivenBody = 5;
% RJ(4).name = 'Waist pitch joint';
RJ(4).drivenby =3;
% RJ(5).name = 'Waist roll motor';
RJ(5).drives = 6;
RJ(5).drivenBody = 5;
% RJ(6).name = 'Waist roll joint';
RJ(6).drivenby =5;
% RJ(7).name = 'Left shoulder motor';
RJ(7).drives = 8;
RJ(7).drivenBody = 9;
% RJ(8).name = 'Left shoulder joint';
RJ(8).drivenby =7;
% RJ(9).name = 'Right shoulder motor';
RJ(9).drives = 10;
RJ(9).drivenBody = 11;
% RJ(10).name = 'Left shoulder joint';
RJ(10).drivenby =9;
% RJ(11).name = 'Left hip yaw motor';
RJ(11).drives = 12;
RJ(11).drivenBody = 13;
% RJ(12).name = 'Left hip yaw joint';
RJ(12).drivenby =11;
% RJ(13).name = 'Left hip roll motor';
RJ(13).drives = 14;
RJ(13).drivenBody = 15;
% RJ(14).name = 'Left hip roll joint';
RJ(14).drivenby =13;
% RJ(15).name = 'Left hip pitch motor';
RJ(15).drives = 16;
RJ(15).drivenBody = 15;
% RJ(16).name = 'Left hip pitch joint';
RJ(16).drivenby =15;
% RJ(17).name = 'Left knee motor';
RJ(17).drives = 18;
RJ(17).drivenBody = 19;
% RJ(18).name = 'Left knee joint';
RJ(18).drivenby =17;
% RJ(19).name = 'Left ankle pitch motor';
RJ(19).drives = 20;
RJ(19).drivenBody = 21;
% RJ(20).name = 'Left ankle pitch joint';
RJ(20).drivenby =19;
```

```
% RJ(21).name = 'Left ankle roll motor';
RJ(21).drives = 22;
RJ(21).drivenBody = 21;
% RJ(22).name = 'Left ankle roll joint';
RJ(22).drivenby =21;
% RJ(23).name = 'Right hip yaw motor';
RJ(23).drives = 24;
RJ(23).drivenBody = 25;
% RJ(24).name = 'Right hip yaw joint';
RJ(24).drivenby =23;
% RJ(25).name = 'Right hip roll motor';
RJ(25).drives = 26;
RJ(25).drivenBody = 27;
% RJ(26).name = 'Right hip roll joint';
RJ(26).drivenby =25;
% RJ(27).name = 'Right hip pitch motor';
RJ(27).drives = 28;
RJ(27).drivenBody = 27;
% RJ(28).name = 'Right hip pitch joint';
RJ(28).drivenby =27;
% RJ(29).name = 'Right knee motor';
RJ(29).drives = 30;
RJ(29).drivenBody = 31;
% RJ(30).name = 'Right knee joint';
RJ(30).drivenby =29;
% RJ(31).name = 'Right ankle pitch motor';
RJ(31).drives = 32;
RJ(31).drivenBody = 33;
% RJ(32).name = 'Right ankle pitch joint';
RJ(32).drivenby =31;
% RJ(33).name = 'Right ankle roll motor';
RJ(33).drives = 34;
RJ(33).drivenBody = 33;
% RJ(34).name = 'Right ankle roll joint';
RJ(34).drivenby =33;
% RJ(35).name = 'Left toe';
% RJ(36).name = 'Right toe';

% Path of RJs from basebody to RJ
RJ(1).JointPath = [1];
RJ(2).JointPath = [2];
RJ(3).JointPath = [2 4];
RJ(4).JointPath = [2];
RJ(5).JointPath = [2 4 6];
RJ(6).JointPath = [2 4];
RJ(7).JointPath = [2 4 6];
RJ(8).JointPath = [2 4 6];
RJ(9).JointPath = [2 4 6];
RJ(10).JointPath = [2 4 6];
RJ(11).JointPath = [11];
RJ(12).JointPath = [12];
RJ(13).JointPath = [12];
RJ(14).JointPath = [12];
RJ(15).JointPath = [12 14 16];
RJ(16).JointPath = [12 14];
RJ(17).JointPath = [12 14 16];
```

```
RJ(18).JointPath = [12 14 16];
RJ(19).JointPath = [12 14 16 18];
RJ(20).JointPath = [12 14 16 18];
RJ(21).JointPath = [12 14 16 18 20 22];
RJ(22).JointPath = [12 14 16 18 20];
RJ(23).JointPath = [23];
RJ(24).JointPath = [24];
RJ(25).JointPath = [24];
RJ(26).JointPath = [24];
RJ(27).JointPath = [24 26 28];
RJ(28).JointPath = [24 26];
RJ(29).JointPath = [24 26 28];
RJ(30).JointPath = [24 26 28];
RJ(31).JointPath = [24 26 28 30];
RJ(32).JointPath = [24 26 28 30];
RJ(33).JointPath = [24 26 28 30 32 34];
RJ(34).JointPath = [24 26 28 30 32];
RJ(35).JointPath = [12 14 16 18 20 22];
RJ(36).JointPath = [24 26 28 30 32 34];

% Path of bodies from basebody to RJ
RJ(1).BodyPath=[1];
RJ(2).BodyPath=[1];
RJ(3).BodyPath=[1 3 5];
RJ(4).BodyPath=[1 3];
RJ(5).BodyPath=[1 3 5 7];
RJ(6).BodyPath=[1 3 5];
RJ(7).BodyPath=[1 3 5 7];
RJ(8).BodyPath=[1 3 5 7];
RJ(9).BodyPath=[1 3 5 7];
RJ(10).BodyPath=[1 3 5 7];
RJ(11).BodyPath=[1];
RJ(12).BodyPath=[1];
RJ(13).BodyPath=[1 13];
RJ(14).BodyPath=[1 13];
RJ(15).BodyPath=[1 13 15 17];
RJ(16).BodyPath=[1 13 15];
RJ(17).BodyPath=[1 13 15 17];
RJ(18).BodyPath=[1 13 15 17];
RJ(19).BodyPath=[1 13 15 17 19];
RJ(20).BodyPath=[1 13 15 17 19];
RJ(21).BodyPath=[1 13 15 17 19 21 23];
RJ(22).BodyPath=[1 13 15 17 19 21];
RJ(23).BodyPath=[1];
RJ(24).BodyPath=[1];
RJ(25).BodyPath=[1 25];
RJ(26).BodyPath=[1 25];
RJ(27).BodyPath=[1 25 27 29];
RJ(28).BodyPath=[1 25 27];
RJ(29).BodyPath=[1 25 27 29];
RJ(30).BodyPath=[1 25 27 29];
RJ(31).BodyPath=[1 25 27 29 31];
RJ(32).BodyPath=[1 25 27 29 31];
RJ(33).BodyPath=[1 25 27 29 31 33 35];
RJ(34).BodyPath=[1 25 27 29 31 33];
RJ(35).BodyPath=[1 13 15 17 19 21 23];
```

```
RJ(36).BodyPath=[1 25 27 29 31 33 35];

% Set joint axis of rotation
for k=1:cst.nRJs
    [junk ax] = max(RJ(k).e);
    RJ(k).axis  = ax; % 1= xi, 2=eta, 3= zeta
    RJ(k).IntError = 0; % also set initial integration error zero
end

% Previous RJ for all bodies
for k=1:cst.nRJs
    B(RJ(k).b).PrevJoint = k;
end

% Body weight
for i=1:cst.nBodies
   B(i).w=B(i).m*[0 0 -cst.g]';
end

% Previous body number
for i=1:cst.nBodies
    clear body_candidates
    for k=1:length(B(i).RJs)
        RJno=B(i).RJs(k);
        body_candidates(k)=RJ(RJno).a;
    end
    B(i).PrevBody = min(body_candidates);
end


% Set path of bodies and joints from pelvis to current body.
for i=2:cst.nBodies
    B(i).BodyPath  = RJ(B(i).PrevJoint).BodyPath;
    B(i).JointPath = RJ(B(i).PrevJoint).JointPath;
    if B(i).JointPath(end)~=B(i).PrevJoint
        B(i).JointPath = [B(i).JointPath B(i).PrevJoint];
    end
end

clear body_candidates i k

% calc. total mass
cst.mass=0;
for i=1:cst.nBodies
    cst.mass = cst.mass + B(i).m;
end

% set furhter initial values
cst.R_ext=[0 0 cst.mass*cst.g]'; %initial external reaction
cst.r_zmp=[0 0 0]'; % initial position of ZMP

% set initial RJ angles (relative coordinates)
for k=1:cst.nRJs
    RJ(k).psi  = 0;
    RJ(k).psiD = 0;
end
```

```
% bend knees in initial position
bend=cst.BendKnees;
RJ(28).psi = deg2rad(-bend);
RJ(30).psi = deg2rad(2*bend);
RJ(32).psi = deg2rad(-bend);
RJ(16).psi = deg2rad(-bend);
RJ(18).psi = deg2rad(2*bend);
RJ(20).psi = deg2rad(-bend);

% update motor angles for above RJs
RJs=[16 18 20 28 30 32];
for k=1:length(RJs)
    j=RJs(k);
    if RJ(j).drive==3
        RJ(RJ(j).drivenby).psi = -RJ(j).psi*RJ(j).u;
    else %drive=1
        RJ(RJ(j).drivenby).psi = RJ(j).psi*RJ(j).u;
    end
end

% set initial reference trajectories
for k=1:cst.nRJs
    RJ(k).psiRef=RJ(k).psi;
end

% determine appropriate pelvis height (because of bend knees)
L_shin  = 0.370;
L_thigh = 0.311;
H_ankle = 0.0760+0.0460;
H_pelvis= 0.0160+0.036+0.0280;
cst.PelvisHeight = H_ankle + cos(deg2rad(bend))*L_shin + ...
                   cos(deg2rad(bend))*L_thigh + H_pelvis;

% save interval related
cst.SaveStep=0;
```

```
function []=MotionPlanning()
% Generates trajectories for feet and pelvis COM.

global cst RJ B S

% constants defining step and gait
S.l       = 0.30;               % step lenght [m]
S.h       = 0.020;              % step height [m]
S.w       = 2*0.14;             % step width [m]
S.f       = 1.500;              % step frequency [1/s]
S.tStep   = 1/S.f;              % step time [s]
S.tStand  = 0.1;                % initial stand still time [s]
S.tSwing  = 0.7*S.tStep;        % swing phase time [s]
S.tStance = S.tStep - S.tSwing; % stance phase time [s]
S.b       = 0.10;               % y position of CoM, when crossing into SSP
Step1     = 0.5;                % shorten 1st step by this factor
S.g       = 9.82;

% check if sufficient time is allocated:
total_time=cst.nSteps*cst.dt;
needed_time=S.tStand+S.tStep*S.nSteps;
tStand_end =total_time-needed_time;
if needed_time>total_time;
    fprintf('Not enough time for all steps...\n');
end

% Get foot projection geometry:
S.heel=cst.heel(1:2);
S.Rankle=cst.s{33,35};
S.Lankle=cst.s{21,23};
S.Rtoe=cst.toes(1:2)+cst.s{37,35}(1:2);
S.Ltoe=-cst.toes(1:2)+cst.s{37,35}(1:2);

CalcA();
CalcPos();

S.z = norm(B(1).r-(B(35).r+S.Rankle));

% Initialize result storage arrays:
S.ZMP_ref(1:3,1:cst.nSteps)=0; S.ZMP_ref(1:3,1)=[B(1).r(1:2); 0];
S.COM_ref(1:3,1:cst.nSteps)=0; S.COM_ref(1:3,1)=B(1).r;
S.COM_ref(3,:)=S.COM_ref(3,1);
S.Rfoot(1:3,1:cst.nSteps)=0;
S.Rfoot(1:3,1)=[0 -S.w/2 B(35).r(3)+S.Rankle(3)];
S.Lfoot(1:3,1:cst.nSteps)=0;
S.Lfoot(1:3,1)=[0  S.w/2 B(23).r(3)+S.Lankle(3)];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Gait phase book keeping information: %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
S.tDSPini=floor(S.tStand/cst.dt)+1; S.tDSPend=[];
S.tSSPini=[]; S.tSSPend=[];
StepNo=1; S.Phase(1)=0; S.Step(1)=0;
time(1)=cst.dt; timeSteps(1)=1;
```

```
for tStep=2:cst.nSteps
    t=tStep*cst.dt; % get time
    time(tStep)=t; % record time
    timeSteps(tStep)=tStep;
    tt= t-S.tStand; % time from beginning walking
    StepNo=floor(tt/S.tStep)+1; % update StepNo
    if t<=S.tStand
        S.Phase(tStep)=0;
        S.Step(tStep)=0;
    elseif StepNo<=S.nSteps
        if rem(tt,S.tStep)>S.tStance % if in SSP
            if rem(StepNo,2)==1
                S.Phase(tStep)=1;
            else
                S.Phase(tStep)=-1;
            end
        else % else in DSP
            S.Phase(tStep)=0;
        end
        S.Step(tStep)=StepNo;
    else % done walking: Phase=DSP, StepNo=-1
        S.Phase(tStep)=0;
        S.Step(tStep)=-1;
    end

    % record  first and last tStep numbers of DSPs:
    if abs(S.Phase(tStep-1))==1 && S.Phase(tStep)==0
        S.tDSPini = [S.tDSPini tStep+1];
    elseif S.Phase(tStep-1)==0 && abs(S.Phase(tStep))==1
        S.tDSPend = [S.tDSPend tStep-1];
    end

    % record  first and last tStep numbers of SSPs:
    if abs(S.Phase(tStep))==1 && S.Phase(tStep-1)==0
        S.tSSPini = [S.tSSPini tStep];
    elseif S.Phase(tStep)==0 && abs(S.Phase(tStep-1))==1
        S.tSSPend = [S.tSSPend tStep];
    end
end
S.tDSPini(end)=[];

% Generate vector of ZMP movement for each DSP
Phases=S.Phase(S.tSSPini);
for StepNo=1:S.nSteps
    if StepNo==1
        zmp_x = -S.COM_ref(1,1);
        zmp_y = 0.5*S.w*Phases(StepNo);
        FootHolds(1:2,StepNo)=[0 S.w/2]';
    elseif StepNo==2
        zmp_x = Step1*S.l;
        zmp_y = S.w*Phases(StepNo);
        FootHolds(1:2,StepNo)=[S.l/2 -S.w/2]';
    else
        zmp_x = S.l/2;
        zmp_y = S.w*Phases(StepNo);
        FootHolds(1:2,StepNo)=[(StepNo-1)*S.l/2 zmp_y/2]';
```

```
    end
    ZMP_move(1:2,StepNo)=[zmp_x zmp_y]';
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Gait trajectory generation   %%%
%%% i.e. ankle, ZMP trajectories %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Switcher=0;
for tStep=2:cst.nSteps

    t=tStep*cst.dt; % get time
    StepNo=S.Step(tStep); % get StepNo

    if StepNo==0 %before walking
        % maintain position of both feet at initial position
        S.Rfoot(1:3,tStep)=[0 -S.w/2 B(35).r(3)+S.Rankle(3)];
        S.Lfoot(1:3,tStep)=[0  S.w/2 B(23).r(3)+S.Lankle(3)];
        S.ZMP_ref(1:3,tStep) = S.ZMP_ref(1:3,tStep-1);
        S.COM_ref(1:3,tStep) = S.COM_ref(1:3,tStep-1);
    elseif S.Step(tStep)>0 %during walking

        % Deal with 1st step
        if StepNo==1
            L = Step1*S.l;  h = Step1*S.h;
            LastSwing=1;
        else
            L = S.l;  h = S.h;
        end

        if S.Phase(tStep)==0 %DSP, determine ZMP reference
            LastStance=tStep;
            % time from beginning of current step (stance only)
            tt  = rem((t-S.tStand),S.tStep);
            x_zmp(tStep) = ZMP_move(1,StepNo)/2-(ZMP_move(1,StepNo)/2)*cos(pi/S.↙
tStance*tt);
            y_zmp(tStep) = ZMP_move(2,StepNo)/2-(ZMP_move(2,StepNo)/2)*cos(pi/S.↙
tStance*tt);
            S.Lfoot(1:3,tStep) = [S.Lfoot(1:2,tStep-1); B(35).r(3)+S.Rankle(3)];
            S.Rfoot(1:3,tStep) = [S.Rfoot(1:2,tStep-1); B(23).r(3)+S.Lankle(3)];
            S.ZMP_ref(1:3,tStep) = S.ZMP_ref(1:3,LastSwing) + [x_zmp(tStep) y_zmp↙
(tStep) 0]';
        else %SSP, determine swing-foot reference
            tt  = rem((t-S.tStand),S.tStep)-S.tStance; % time from beginning of current↙
swing-phase
            x = L/2-(L/2)*cos(pi/S.tSwing*tt);
            z = (h/2)*(1-cos(2*pi/S.tSwing*tt));
            switch S.Phase(tStep) %left/right SSP?
                case 1 % left SSP
                    S.Rfoot(1:3,tStep) = S.Rfoot(1:3,LastStance) + [x 0 z]';
                    S.Lfoot(1:3,tStep) = S.Lfoot(1:3,LastStance);
                    S.ZMP_ref(1:3,tStep) = S.ZMP_ref(1:3,tStep-1);
                case -1 % right SSP
                    S.Lfoot(1:3,tStep) = S.Lfoot(1:3,LastStance) + [x 0 z]';
                    S.Rfoot(1:3,tStep) = S.Rfoot(1:3,LastStance);
```

```
                    S.ZMP_ref(1:3,tStep) = S.ZMP_ref(1:3,tStep-1);
            end
            LastSwing=tStep;
        end

    elseif StepNo<0 %after walking
        % maintain position of both feet at final position
        S.Lfoot(1:3,tStep) = S.Lfoot(1:3,tStep-1);
        S.Rfoot(1:3,tStep) = S.Rfoot(1:3,tStep-1);
        S.ZMP_ref(1:3,tStep) = S.ZMP_ref(1:3,tStep-1);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Determine pelvis COM trajectory       %%%
%%% using inverted pendulum model in SSPs %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Determine final foothold
ZMP_move(1:2,end+1)=[S.l/2 S.w*-Phases(end)]';
for StepNo=1:S.nSteps

    xDyD_i = ZMP_move(1,StepNo)/ZMP_move(2,StepNo);
    xDyD_f = ZMP_move(1,StepNo+1)/ZMP_move(2,StepNo+1);

    y_i(StepNo) = -Phases(StepNo)*S.b;
    y_f(StepNo) = y_i(StepNo);
    x_i(StepNo) = y_i(StepNo)*xDyD_i;
    x_f(StepNo) = y_f(StepNo)*xDyD_f;

    Tc = sqrt(S.z/S.g);
    Ts = S.tSwing-cst.dt;

    yD_i(StepNo) = (y_f(StepNo)-cosh(Ts/Tc)*y_i(StepNo))/(Tc*sinh(Ts/Tc));
    xD_i(StepNo) = (x_f(StepNo)-cosh(Ts/Tc)*x_i(StepNo))/(Tc*sinh(Ts/Tc));
    xD_f(StepNo) = sinh(Ts/Tc)/Tc*x_i(StepNo)+cosh(Ts/Tc)*xD_i(StepNo);
    yD_f(StepNo) = sinh(Ts/Tc)/Tc*y_i(StepNo)+cosh(Ts/Tc)*yD_i(StepNo);

    w=Tc^-1;
    C1=0.5*(x_i(StepNo)+1/w*xD_i(StepNo));
    C2=0.5*(x_i(StepNo)-1/w*xD_i(StepNo));
    C3=0.5*(y_i(StepNo)+1/w*yD_i(StepNo));
    C4=0.5*(y_i(StepNo)-1/w*yD_i(StepNo));

    % SSPs
    for tStep=S.tSSPini(StepNo):S.tSSPend(StepNo)
        t=(tStep-S.tSSPini(StepNo))*cst.dt;
        x_t = C1*exp(w*t) + C2*exp(-w*t);
        y_t = C3*exp(w*t) + C4*exp(-w*t);

        S.COM_ref(1:2,tStep) = FootHolds(1:2,StepNo) + [x_t y_t]';
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Determine pelvis COM trajectory       %%%
```

```
%%% using cubic splines in DSPs           %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for StepNo=1:S.nSteps
    if StepNo==1 %first DSP
        px0 = S.COM_ref(1,1);
        px1 = FootHolds(1,StepNo) + x_i(StepNo);
        mx0 = 0;
        mx1 = xD_i(StepNo)*S.tStance;
        py0 = 0;
        py1 = FootHolds(2,StepNo) + y_i(StepNo);
        my0 = 0;
        my1 = yD_i(StepNo)*S.tStance;
    else
        px0 = FootHolds(1,StepNo-1) + x_f(StepNo-1);
        px1 = FootHolds(1,StepNo) + x_i(StepNo);
        mx0 = xD_f(StepNo-1)*S.tStance;
        mx1 = xD_i(StepNo)*S.tStance;
        py0 = FootHolds(2,StepNo-1) + y_f(StepNo-1);
        py1 = FootHolds(2,StepNo) + y_i(StepNo);
        my0 = yD_f(StepNo-1)*S.tStance;
        my1 = yD_i(StepNo)*S.tStance;
    end

    % DSPs
    n_tSteps=S.tDSPend(StepNo)-S.tDSPini(StepNo)+2;
    clear ttt; ttt = 0:1/(n_tSteps):1;
    step=0;
    for tStep=S.tDSPini(StepNo)-1:S.tDSPend(StepNo)+1
        step=step+1;
        t=ttt(step);
        x_t = (2*t^3-3*t^2+1)*px0 + (t^3-2*t^2+t)*mx0...
              + (-2*t^3+3*t^2)*px1 + (t^3-t^2)*mx1;
        y_t = (2*t^3-3*t^2+1)*py0 + (t^3-2*t^2+t)*my0...
              + (-2*t^3+3*t^2)*py1 + (t^3-t^2)*my1;

        S.COM_ref(1:2,tStep) = [x_t y_t]';
    end
end

% Last DSP
px0 = S.COM_ref(1,S.tSSPend(end));
px1 = S.COM_ref(1,S.tSSPend(end))+S.l/4;
mx0 = xD_f(end)*tStand_end;
mx1 = 0;
py0 = S.COM_ref(2,S.tSSPend(end));
py1 = 0;
my0 = yD_f(end)*tStand_end;
my1 = 0;
n_tSteps=cst.nSteps-(S.tSSPend(end)+1);
for tStep=S.tSSPend(end)+1:cst.nSteps
        t=(tStep-(S.tSSPend(end)+1))/(n_tSteps+1);
        x_t = (2*t^3-3*t^2+1)*px0 + (t^3-2*t^2+t)*mx0...
              + (-2*t^3+3*t^2)*px1 + (t^3-t^2)*mx1;
        y_t = (2*t^3-3*t^2+1)*py0 + (t^3-2*t^2+t)*my0...
              + (-2*t^3+3*t^2)*py1 + (t^3-t^2)*my1;
```

```
        S.COM_ref(1:2,tStep) = [x_t y_t]';
end
```

```
function []=InverseKinematics()
% Determines the angular trajectories for all active joints
% from the COM and ankle trajectories, using inverse kinematics.

global S RJ B cst
s=cst.s;

% Initialize arrays
time = 1:cst.nSteps;
ul(time)=0; vl(time)=0; wl(time)=0; ur(time)=0; vr(time)=0; wr(time)=0;
S.ThetaL_1(time)=0; S.ThetaL_2(time)=0; S.ThetaL_3(time)=0;
S.ThetaL_4(time)=0; S.ThetaL_5(time)=0; S.ThetaL_6(time)=0;
S.ThetaR_1(time)=0; S.ThetaR_2(time)=0; S.ThetaR_3(time)=0;
S.ThetaR_4(time)=0; S.ThetaR_5(time)=0; S.ThetaR_6(time)=0;
ss{1,1}(1:3,time) = 0; ss{2,1}(1:3,time) = 0; ss{3,2}(1:3,time) = 0;
ss{4,4}(1:3,time) = 0; ss{5,4}(1:3,time) = 0; ss{6,5}(1:3,time) = 0;

%%%%%% Define local vectors from joint to joint ss(a,b) to a from b %%%%%%
% 1:RHip, 2:RShin, 3:RAnkle, 4:LHip, 5:LShin, 6:LAnkle, 10:Pelvis.

ssl{1,1} = s{25,1}-s{1,25} + s{27,25};                  % CoM Pelvis to RHip
ssl{2,1} = (s{31,29}-s{27,29});                         % RHip to Rknee
ssl{3,2} = s{33,31}-s{29,31};                           % RKnee to RAnkle
ssl{4,3} = s{37,35};                                    % RAnkle to CoM RFoot

% Left Leg
ssl{4,4} =  s{13,1}-s{1,13}+s{15,13};                   % CoM Pelvis to LHip
ssl{5,4} = (s{19,17}-s{15,17});                         % LHip to Lknee
ssl{6,5} = s{21,19}-s{17,19};                           % LKnee to LAnkle
ssl{7,6} = s{36,23};                                    % LAnkle to CoM LFoot

%Determine constant leg lengths
d1 = norm(ssl{2,1}); %Length of thigh
d2 = norm(ssl{3,2}); %Length of shin

% Calc. joint trajectories by inverse kinematics
for tStep=1:cst.nSteps
    % Determine pelvis position relative to feet
    S.LfootRel(1:3,tStep) = S.COM_ref(1:3,tStep)+ssl{4,4}(1:3) - S.Lfoot(1:3,tStep);
    S.RfootRel(1:3,tStep) = S.COM_ref(1:3,tStep)+ssl{1,1}(1:3) - S.Rfoot(1:3,tStep);

    % Extract u, v and w
    %Left
    ul(tStep) = S.LfootRel(1,tStep);
    vl(tStep) = S.LfootRel(2,tStep);
    wl(tStep) = S.LfootRel(3,tStep);
    %Right
    ur(tStep) = S.RfootRel(1,tStep);
    vr(tStep) = S.RfootRel(2,tStep);
    wr(tStep) = S.RfootRel(3,tStep);

    % Calculates Angles for legs
    % Theta_1: Hip Roll, Theta_2: Hip Pitch, Theta_3: Hip Yaw,
    % Theta_4: Knee pitch, Theta_5: Ankle pitch, Theta_6: Ankle Roll

    %Left
```

```
    S.ThetaL_4(tStep) = acos((ul(tStep)^2+wl(tStep)^2+vl(tStep)^2-d1^2-d2^2)/↙
(2*d1*d2));
    S.ThetaL_5(tStep) = asin((-d1*sin(S.ThetaL_4(tStep)))/(sqrt(ul(tStep)^2+vl(tStep)↙
^2+wl(tStep)^2))-...
                        atan(ul(tStep)/(sqrt(wl(tStep)^2+vl(tStep)^2))));
    S.ThetaL_6(tStep) = atan(vl(tStep)/wl(tStep));
    S.ThetaL_1(tStep) = -S.ThetaL_6(tStep);
    S.ThetaL_2(tStep) = -S.ThetaL_4(tStep)-S.ThetaL_5(tStep);
    S.ThetaL_3(tStep) = 0;

    %Right
    S.ThetaR_4(tStep) = acos((ur(tStep)^2+wr(tStep)^2+vr(tStep)^2-d1^2-d2^2)/↙
(2*d1*d2));
    S.ThetaR_5(tStep) = asin((-d1*sin(S.ThetaR_4(tStep)))/(sqrt(ur(tStep)^2+vr(tStep)↙
^2+wr(tStep)^2))-...
                        atan(ur(tStep)/(sqrt(wr(tStep)^2+vr(tStep)^2))));
    S.ThetaR_6(tStep) = atan(vr(tStep)/wr(tStep));
    S.ThetaR_1(tStep) = -S.ThetaR_6(tStep);
    S.ThetaR_2(tStep) = -S.ThetaR_4(tStep)-S.ThetaR_5(tStep);
    S.ThetaR_3(tStep) = 0;
end
```

```
function []=CalcA()
% Update transformation matrices for all bodies.

global B RJ cst

% Basebody (pelvis): Bryant / Euler angles / Euler parameters
if strcmp(cst.Angles,'BryantAngles')
    B(1).A = BryantTrMatrix(B(1).Theta);
elseif strcmp(cst.Angles,'EulerAngles')
    B(1).A = EulerTrMatrix(B(1).Theta);
elseif strcmp(cst.Angles,'EulerParameters')
    B(1).A = EulerParamTrMatrix(B(1).p);
end

% Remaining bodies: Euler-parameters
for ismotor=0:1 %First treat non-motor bodies
    for i=2:cst.nBodies
        if B(i).motor==ismotor

            % Get previous body number and A matrix
            PrevBody = B(i).PrevBody;
            Ai = B(PrevBody).A;

            % RJ number connecting current body to previuos
            PrevJoint = B(i).PrevJoint;

            % Euler-parameter vector and relative rotation
            e     = RJ(PrevJoint).e;
            psi   = RJ(PrevJoint).psi;
            e0    = cos(psi/2);
            e1    = e(1)*sin(psi/2);
            e2    = e(2)*sin(psi/2);
            e3    = e(3)*sin(psi/2);

            % Euler-parameter transformation matrix (Nikravesh,1988,p.160)
            Aji = 2*[e0^2+e1^2-0.5     e1*e2-e0*e3     e1*e3+e0*e2;
                       e1*e2+e0*e3   e0^2+e2^2-0.5     e2*e3-e0*e1;
                       e1*e3-e0*e2     e2*e3+e0*e1   e0^2+e3^2-0.5];

            % Update A matrix of current body
            B(i).A = Ai*Aji;
       end
    end
end
```

```
function [] = CalcPos()
% Calculates cartesian position for each body

global B cst

for ismotor=0:1 % treat non-motors first.
    for j=2:cst.nBodies
        if B(j).motor==ismotor
            i=B(j).PrevBody;
            r_i = B(i).r;
            A_i = B(i).A;
            s_i = cst.s{j,i};
            A_j = B(j).A;
            s_j = cst.s{i,j};

            B(j).r = r_i + A_i*s_i - A_j*s_j;
        end
    end
end
```

```
function []=CalcOmega()
% Calculates the angular velocity (omega) of all bodies.

global B RJ cst

% Remaining bodies
for ismotor=0:1 % treat non-motor bodies first
    for i=2:cst.nBodies
       if B(i).motor==ismotor

          % Get number of previous body and joint connecting to this
          PrevBody  = B(i).PrevBody;
          PrevJoint = B(i).PrevJoint;

          % Get input data
          omega_i = B(PrevBody).omega;
          A       = B(PrevBody).A;
          psiD    = RJ(PrevJoint).psiD;
          e_loc   = RJ(PrevJoint).e;
          e       = A*e_loc;

          % Compute
          B(i).omega = omega_i + psiD*e;
       end
    end
end
```

```
function [] = CalcVel
% Calculates velocity of each body part

global B cst

for ismotor=0:1 % first non-motor bodies
    for j=2:cst.nBodies
        if B(j).motor==ismotor

            i       = B(j).PrevBody;
            rD_i    = B(i).rD;
            omega_i = skew(B(i).omega);
            s_i     = B(i).A*cst.s{j,i};
            omega_j = skew(B(j).omega);
            s_j     = B(j).A*cst.s{i,j};

            B(j).rD = rD_i + omega_i*s_i - omega_j*s_j;
        end
    end
end
```

```
function [] = Regulator2(t,tStep)
% Determines motor torques by PID control

global B RJ cst F S

% Set reference trajectories
F(1).Rot(1:3) = TrMatrixToBryant(B(F(1).bodyNo).A);
F(2).Rot(1:3) = TrMatrixToBryant(B(F(2).bodyNo).A);
%Right leg
RJ(24).psiRef = S.ThetaR_3(tStep);
RJ(26).psiRef = S.ThetaR_1(tStep);
RJ(28).psiRef = S.ThetaR_2(tStep);
RJ(30).psiRef = S.ThetaR_4(tStep);
RJ(32).psiRef = S.ThetaR_5(tStep);
RJ(34).psiRef = S.ThetaR_6(tStep) - F(2).Rot(1);
%Left leg
RJ(12).psiRef = S.ThetaL_3(tStep);
RJ(14).psiRef = S.ThetaL_1(tStep);
RJ(16).psiRef = S.ThetaL_2(tStep);
RJ(18).psiRef = S.ThetaL_4(tStep);
RJ(20).psiRef = S.ThetaL_5(tStep);
RJ(22).psiRef = S.ThetaL_6(tStep) - F(1).Rot(1);

% Set gains
RJ(2).Kp  = 100;  RJ(2).Ki = 10;  RJ(2).Kd = 1;   %Waist Yaw
RJ(4).Kp  = 100;  RJ(4).Ki = 10;  RJ(4).Kd = 1;   %Waist Pitch
RJ(6).Kp  = 100;  RJ(6).Ki = 70;  RJ(6).Kd = 1;   %Waist Roll
RJ(8).Kp  =  50;  RJ(8).Ki = 10;  RJ(8).Kd = 0.5; %Shoulder Pitch
RJ(12).Kp = 100; RJ(12).Ki = 10; RJ(12).Kd = 10;  %Hip Yaw
RJ(14).Kp = 100; RJ(14).Ki = 10; RJ(14).Kd = 10;  %Hip Roll
RJ(16).Kp = 100; RJ(16).Ki = 15; RJ(16).Kd = 20;  %Hip Pitch
RJ(18).Kp = 110; RJ(18).Ki = 15; RJ(18).Kd = 10;  %Knee Pitch
RJ(20).Kp = 100; RJ(20).Ki = 10; RJ(20).Kd = 10;  %Ankle Pitch
RJ(22).Kp = 100; RJ(22).Ki = 10; RJ(22).Kd = 2;   %Ankle Roll
% Symmetry
RJ(10).Kp = RJ(8).Kp;  RJ(10).Ki = RJ(8).Ki;  RJ(10).Kd = RJ(8).Kd;
RJ(24).Kp = RJ(12).Kp; RJ(24).Ki = RJ(12).Ki; RJ(24).Kd = RJ(12).Kd;
RJ(26).Kp = RJ(14).Kp; RJ(26).Ki = RJ(14).Ki; RJ(26).Kd = RJ(14).Kd;
RJ(28).Kp = RJ(16).Kp; RJ(28).Ki = RJ(16).Ki; RJ(28).Kd = RJ(16).Kd;
RJ(30).Kp = RJ(18).Kp; RJ(30).Ki = RJ(18).Ki; RJ(30).Kd = RJ(18).Kd;
RJ(32).Kp = RJ(20).Kp; RJ(32).Ki = RJ(20).Ki; RJ(32).Kd = RJ(20).Kd;
RJ(34).Kp = RJ(22).Kp; RJ(34).Ki = RJ(22).Ki; RJ(34).Kd = RJ(22).Kd;

% Loop motors and determine torque
if cst.MotorsOn == 1
    for j = 1:cst.nRJs
        if RJ(j).drive == 1 || RJ(j).drive == 3
            % driving motor's body-no
            m = RJ(RJ(j).drivenby).b;

            if RJ(j).drive == 1
                Error = RJ(j).psiRef-RJ(j).psi;
            else
                Error = RJ(j).psi-RJ(j).psiRef;
            end
```

```
            % store error information
            RJ(j).IntError  = RJ(j).IntError + Error*cst.dt;
            if tStep == 1
                RJ(j).DiffError = 0;
            else
                RJ(j).DiffError = (Error-RJ(j).Error)/cst.dt;
            end
            RJ(j).Error = Error;

            % Determine PID response
            P = RJ(j).Kp*Error;
            I = RJ(j).Ki*(RJ(j).IntError);
            D = RJ(j).Kd*(RJ(j).DiffError);
            % Motor torque
            Mm = P + I + D;

            if abs(Mm)>B(m).Mmax % apply limit
                Mm=B(m).Mmax*sign(Mm);%limit motor torque to 4xM_nom
                RJ(j).IntError = 0;% reset integrator, avoiding windup
            end

            % Return motor torque, through low-pass filter
            alpha = cst.dt/(B(m).t+cst.dt);
            B(m).Mm = alpha*Mm + (1-alpha)*B(m).Mm;
        end
    end
end

% Turns off all motors, will collapse robot
if cst.MotorsOn==0 && cst.LockRJs==0
    for j = 1:cst.nRJs
        if RJ(j).drive == 1 || RJ(j).drive == 3
            B(RJ(RJ(j).drivenby).b).Mm = 0;
        end
    end
end
```

```
function []=ConstrainFeet(tStep)
% Applies  spring-damper system connected to feet
% thereby constraining them to the ground.

global B F cst
if cst.FootForces==1 % apply foot forces?
    dh=-cst.s{37,35}(3); % get ankle height

    if tStep==1 % get initial position and orientation of feet
        F(1).r_i=[B(F(1).bodyNo).r(1:2); dh];
        F(2).r_i=[B(F(2).bodyNo).r(1:2); dh];
        F(1).theta_i=TrMatrixToBryant(B(F(1).bodyNo).A);
        F(2).theta_i=TrMatrixToBryant(B(F(2).bodyNo).A);
    end

    % record new position of feet, if contact
    if F(1).contact==0 && B(F(1).bodyNo).r(3)<=dh
        F(1).r_i=B(F(1).bodyNo).r(1:3);
        cst.ContactShift = [cst.ContactShift tStep];
    end
    if F(2).contact==0 && B(F(2).bodyNo).r(3)<=dh
        F(2).r_i=B(F(2).bodyNo).r(1:3);
        cst.ContactShift = [cst.ContactShift tStep];
    end

    % Set spring and damper coefficients
    cst.k_l = 70*cst.g/0.001*[0.5 0.2 1]';
    cst.c_l = [0.5 0.2 1]'*1.0*sqrt(cst.k_l(3)*70);
    cst.k_r = 5000*[1 1 0.5]';
    cst.c_r = [1 1 0.5]'*0.2*sqrt(cst.k_r(1)*70);
    cst.friction=0.5;

    % Calculate forces and moments needed for constraining feet
    for f=1:2

        % get kinematics
        j       = F(f).bodyNo;
        A       = B(j).A;
        theta   = transpose(TrMatrixToBryant(A));
        theta_i = [0 0 0]';
        r_i     = F(f).r_i;
        r       = B(j).r;

        if B(j).rD(3)>0 % ignore damping and traction if foot moves upwards
            F(f).rD     = [0 0 0]';
            F(f).dL(1:2)= [0 0]';
            F(f).dTheta = [0 0 0]';
            F(f).omega  = [0 0 0]';
        end

        if r(3)<=dh % if contact: determine spring compression
            F(f).dL      = r_i-r;
            F(f).dTheta  = theta_i-theta;
            F(f).omega   = B(j).omega;
            F(f).rD      = B(j).rD;
            F(f).contact = 1;
```

```
        else % no contact
            F(f).rD      = [0 0 0]';
            F(f).dL      = [0 0 0]';
            F(f).dTheta  = [0 0 0]';
            F(f).omega   = [0 0 0]';
            F(f).contact = 0;
        end

        % calculate spring and damper forces and moments
        c_r = cst.c_r*(F(f).dL(3)/0.001);
        c_l = cst.c_l*(F(f).dL(3)/0.001);

        F(f).Fs = cst.k_l.*F(f).dL;
        F(f).Fd = -c_l.*F(f).rD;
        F(f).Ms = cst.k_r.*F(f).dTheta;
        F(f).Md = -c_r.*F(f).omega;

        % sum up forces and moments
        F(f).Fsum = F(f).Fs + F(f).Fd;
        F(f).Msum = F(f).Ms + F(f).Md;

        % store velocities for energy calculations
        F(f).rD    = B(j).rD;
        F(f).omega = B(j).omega;

        % limit reactions between feet and ground
        if F(f).Fsum(3)<0
           F(f).Fsum (3) = 0;
        end
        if abs(F(f).Msum(2))>abs(F(f).Fsum(3)*cst.s{37,35}(1)) % max pitch torque
            F(f).Msum(2) = sign(F(f).Msum(2))*abs(F(f).Fsum(3)*cst.s{37,35}(1));
        end
        if abs(F(f).Msum(2))>abs(F(f).Fsum(3)*cst.s{37,35}(1)) % max pitch torque
            F(f).Msum(2) = sign(F(f).Msum(2))*abs(F(f).Fsum(3)*cst.s{37,35}(1));
        end
        if abs(F(f).Msum(1))>abs(F(f).Fsum(3)*cst.toes(2)) % max roll torque
            F(f).Msum(1) = sign(F(f).Msum(1))*abs(F(f).Fsum(3)*cst.toes(2));
        end
        if norm(F(f).Fsum(1:2))>abs(cst.friction*F(f).Fsum(3)) % max traction force
            F(f).Fsum(1:2) = abs(cst.friction*F(f).Fsum(3))*unit(F(f).Fsum(1:2));
        end

    end

else % disregard calculated foot forces
    for f=1:2
       F(f).Fsum=[0 0 0]'; F(f).Msum=[0 0 0]';
    end
end
```

```
function [] = EnergyBalance2(tStep)
% Calculates Energy balance.

global B cst E RJ F Results

% Initialize variables
E.Epot=0; E.EkinTrans=0; E.EkinRot=0;
if tStep==1
    cst.Epot_initial=0; E.Emotor=0; E.Egear=0; E.Efoot=0;
    for f=1:2
        F_old{f}=[0 0 0]'; M_old{f}=[0 0 0]'; rD_old{f}=[0 0 0]'; omega_old{f}=[0 0↙
0]';
    end
    Mg_old(1:cst.nRJs)=0; psiDg_old(1:cst.nRJs)=0;
    Mm_old(1:cst.nRJs)=0; psiD_old(1:cst.nRJs)=0;
end

% Get values from previous time step for trapezoid integration
if tStep>1
    if cst.FootForces==1 % foot forces
        for f=1:2
            F_old{f}=Results(tStep-1).F(f).Fsum;
            M_old{f}=Results(tStep-1).F(f).Msum;
            rD_old{f}=Results(tStep-1).F(f).rD;
            omega_old{f}=Results(tStep-1).F(f).omega;
        end
    end
    for k=1:cst.nRJs % motor torques
        if RJ(k).drive==2
           Mm_old(k)=Results(tStep-1).Mm{k+1};
           psiD_old(k)=Results(tStep-1).psiD{k};

           g = RJ(k).drives;
           Mg_old(g)=Results(tStep-1).Mg{g};
           psiDg_old(g)=Results(tStep-1).psiD{g};
        end
    end
end

% Summing up mechanical energy for all bodies
for j = 1:cst.nBodies
    % Potential energy
    m = B(j).m;
    g = cst.g;
    h = B(j).r(3);
    E.Epot = E.Epot + m*g*h;

    % Translatoric kinetic energy
    v = B(j).rD;
    E.EkinTrans = E.EkinTrans + 0.5*m*v'*v;

    % Rotational kinetic energy
    A       = B(j).A;
    J       = B(j).J;
    omega_m = A'*B(j).omega;
    E.EkinRot = E.EkinRot + 0.5*omega_m'*J*omega_m;
```

```
end

% Store initial level of potential energy...
if tStep==1; cst.Epot_initial = E.Epot; end
% ...and subtract, to reduce potential energy level
E.Epot = E.Epot - cst.Epot_initial;

% Energy related to foot constraining
if cst.FootForces==1
    for f=1:2
        E.Efoot = E.Efoot + (F(f).Fsum'*F(f).rD + F_old{f}'*rD_old{f})*cst.dt/2;
        E.Efoot = E.Efoot + (F(f).Msum'*F(f).omega + M_old{f}'*omega_old{f})*cst.dt/2;
    end
end

% Calculate applied energy (due to motor torques)
% and energy lost in gears due to efficiency
for k=1:cst.nRJs
   if RJ(k).drive==2
        b    = RJ(k).b;
        g    = RJ(k).drives;
        mu   = RJ(g).mu;
        Mm   = B(b).Mm;
        Mg   = (RJ(g).Mg+Mg_old(g))/2;
        psiDg= (RJ(g).psiD+psiDg_old(g))/2;
        psiD = RJ(k).psiD;

        Emot = (Mm*psiD + Mm_old(k)*psiD_old(k))*cst.dt/2;
        E.Emotor = E.Emotor + Emot;

        if RJ(g).drive==1
            if Mg*psiDg>0 % motor drives gear
                E.Egear  = E.Egear - (1/mu-1)*Mg*psiDg*cst.dt;
            else % gear drives motor
                E.Egear  = E.Egear - (mu-1)*Mg*psiDg*cst.dt;
            end
        else
            if Mg*psiDg>0 % motor drives gear
                E.Egear  = E.Egear - (1/mu-1)*Mg*psiDg*cst.dt;
            else % gear drives motor
                E.Egear  = E.Egear - (mu-1)*Mg*psiDg*cst.dt;
            end
        end
    end
end

%Calculate totals
E.Epot    = E.Epot;
E.Ekin    = E.EkinTrans + E.EkinRot;
E.Emech   = E.Epot + E.Ekin;
E.Eapp    = E.Emotor + E.Efoot + E.Egear;
E.Etot    = E.Emech - E.Eapp;
```

```
function []=CalcGamma()
% Calculates gamma-contribution to the RHS of the EOMs.

global B RJ F cst

%%% CALCULATE GAMMA_w %%%
for i=2:cst.nBodies

   % Reset gamma_w
   gamma_w=0;

   % Get numbers of RJ's in path to current body.
   Joints = B(i).JointPath;
   m = length(Joints);

   % Loop over RJ's from pelvis to current body
   for j=1:m
       % Get input
       k       = Joints(j);
       a       = RJ(k).a;
       omega_a = B(a).omega;
       A       = B(a).A;
       psiD    = RJ(k).psiD;
       e_loc   = RJ(k).e;
       e       = A*e_loc;

       % Compute
       gamma_w = gamma_w + psiD*skew(omega_a)*e;
   end

   % Return results
   B(i).gamma_w = gamma_w;
end

%%% CALCULATE GAMMA_r %%%
for i=2:cst.nBodies

   % Get number of bodies and their numbers from pelvis to current RJ
   Joints = B(i).JointPath;
   m = length(Joints);

   % Reset gamma_r
   omega_1 = B(1).omega;
   l_0     = B(RJ(B(i).JointPath(1)).a).A*-RJ(B(i).JointPath(1)).sa; %vector from↙
pelvis CoM to first RJ
   gamma_r = skew(omega_1)*skew(omega_1)*l_0;

   % Loop over the chain of RJ's from pelvis to current body
   for j=1:m %summation-loop

       % Get book-keeping input
       k       = Joints(j);
       a       = RJ(k).a;
       b       = RJ(k).b;
       Aa      = B(a).A;
       Ab      = B(b).A;
```

```
        r_i     = B(i).r;
        r_k     = B(a).r-Aa*RJ(k).sa;

        % Get geometry
        if j<m
            l_k     = Ab*RJ(k).sb - B(RJ(Joints(j+1)).a).A*RJ(Joints(j+1)).sa; % From↙
current RJ to next
            d_k     = r_i - r_k; % From current RJ to i'th body's CoM
        else
            l_k     = Ab*RJ(k).sb;
            d_k     = l_k;
        end

        % Get data
        omega_a = B(a).omega;
        omega_b = B(b).omega;
        A       = B(a).A;
        psiD    = RJ(k).psiD;
        e_loc   = RJ(k).e;
        e       = A*e_loc;

        % Compute
        gamma_r = gamma_r - psiD*skew(d_k)*skew(omega_a)*e + skew(omega_b)*skew(omega_b)↙
*l_k;
    end

    % Return results
    B(i).gamma_r = gamma_r;
end
```

```
function [ y ]=Setupy(tStep)
% Sets up right hand side (y) of EOMs
% y contains the following terms:
% w_g, wJw, m*gamma_r, J*gamma_w, Mm, F

global RJ B F cst

% Initialize y-vector
y(1:cst.nBodies*6,1)=0;

% Fill in body-weights:
for i=1:cst.nBodies
   y([1:3]+6*(i-1)) = -B(i).w;
end

% Fill in squared velocity terms:
for i=1:cst.nBodies
   omega = B(i).omega;
   A     = B(i).A;
   J     = A*B(i).J*A';

   y([4:6]+6*(i-1)) = skew(omega)*J*omega;
end

% Fill in m*gamma_r:
for i=2:cst.nBodies
   m       = B(i).m;
   gamma_r = B(i).gamma_r;

   y([1:3]+6*(i-1)) = y([1:3]+6*(i-1)) + m*gamma_r;
end

% Fill in J*gamma_w:
for i=2:cst.nBodies
   J       = B(i).A*B(i).J*(B(i).A)';
   gamma_w = B(i).gamma_w;

   y([4:6]+6*(i-1)) = y([4:6]+6*(i-1)) + J*gamma_w;
end

% Fill in Mm:
for k=1:cst.nRJs
   if RJ(k).drive==2
        Mmot     = B(RJ(k).b).Mm;
        e_loc    = RJ(k).e;
        a        = RJ(k).a;
        b        = RJ(k).b;
        A        = B(a).A;
        e        = A*e_loc;

        Mm       = e*Mmot;

        % stator body (a)
        y([4:6]+6*(a-1)) = y([4:6]+6*(a-1)) + Mm;
        % rotor body (b)
        y([4:6]+6*(b-1)) = y([4:6]+6*(b-1)) - Mm;
```

```
   end
end

% Fill in foot forces F:
if cst.FootForces==1
    %Left foot:
    i=F(1).bodyNo;
    y([1:3]+6*(i-1)) = y([1:3]+6*(i-1)) - F(1).Fsum;
    y([4:6]+6*(i-1)) = y([4:6]+6*(i-1)) - F(1).Msum;

    %Right foot:
    i=F(2).bodyNo;
    y([1:3]+6*(i-1)) = y([1:3]+6*(i-1)) - F(2).Fsum;
    y([4:6]+6*(i-1)) = y([4:6]+6*(i-1)) - F(2).Msum;
end
```

```
function [ C ]=SetupC()
% Sets up coefficient matrix for EOMs
% C-matrix contains the following unknown terms:
% m*rDD J*omegaD Rk MRk skew(s)*Rk Mg

global RJ B cst

% Initialize C:
C = zeros(cst.nBodies*6);
I = eye(3);

% Fill in Rk: identity-matrix in positions relating body_a and body_b
for k=1:cst.nRJs
    % Get numbers of bodies connected by RJ
    a = RJ(k).a;
    b = RJ(k).b;

    % body-a:
    C([1:3]+6*(a-1),[7:9]+6*(k-1)) = I;
    % body-b:
    C([1:3]+6*(b-1),[7:9]+6*(k-1)) = -I;
end


% Fill in skew(s)*Rk:
for k=1:cst.nRJs
    a = RJ(k).a;
    b = RJ(k).b;
    sa = B(a).A*-RJ(k).sa;
    sb = B(b).A*-RJ(k).sb;

    % body a:
    C([4:6]+6*(a-1),[7:9]+6*(k-1)) = skew(sa);
    % body-b:
    C([4:6]+6*(b-1),[7:9]+6*(k-1)) = -skew(sb);
end


% Fill in J*omegaD: relating omegaD_1 and psiDD_k
C([4:6],[4:6]) =-B(1).A*B(1).J*(B(1).A)';
for i=2:cst.nBodies

    % Get inertia matrix of current body
    J = B(i).A*B(i).J*(B(i).A)';

    % Insert J*omega1 (relating body_1 and body_i)
    C([4:6]+6*(i-1),[4:6]) = C([4:6]+6*(i-1),[4:6]) -J;

    % Get numbers of RJ's in path to current body.
    Joints = B(i).JointPath;
    m = length(Joints);

    % Loop over RJ's from pelvis to current body
    for j=1:m
       % Get data
       k       = Joints(j);
```

```
        A       = B(RJ(k).a).A;
        e_loc   = RJ(k).e;
        e       = A*e_loc;
        axis    = RJ(k).axis;

        % get body number and gearing
        if RJ(k).drive==2 || RJ(k).drive==0
            kk=k;
            u=1;
        elseif RJ(k).drive==1
            kk = RJ(k).drivenby;
            u = RJ(k).u;
        elseif RJ(k).drive==3
            kk = RJ(k).drivenby;
            u = -RJ(k).u;
        end
        C([4:6]+6*(i-1),[9+axis]+6*(kk-1)) = C([4:6]+6*(i-1),[9+axis]+6*(kk-1)) -J*e*↙
(1/u);
    end
end


% Fill in m*rDD: acceleration terms
C(1:3,1:3)=-I*B(1).m;
for i=2:cst.nBodies

    % Get: mass, d_0
    mass = B(i).m;
    d_0  = B(i).r - B(1).r;

    % Insert
    C([1:3]+6*(i-1),[1:3]) = C([1:3]+6*(i-1),[1:3]) -I*mass;
    C([1:3]+6*(i-1),[4:6]) = C([1:3]+6*(i-1),[4:6]) + skew(d_0)*mass;


    % Get numbers of RJ's in path to current body
    Joints = B(i).JointPath;
    m = length(Joints);

    % Loop over the chain of RJ's from pelvis to current body
    for j=1:m

        % Get book-keeping data
        k       = Joints(j);
        a       = RJ(k).a;
        A       = B(a).A;
        r_i     = B(i).r;
        r_k     = B(a).r-A*RJ(k).sa;
        e_loc   = RJ(k).e;
        e       = A*e_loc;
        axis    = RJ(k).axis;

        % Get geometry
        if j<m
           d_k  = r_i - r_k; % From current RJ to i'th body's CoM
        else
```

```
           d_k  = B(RJ(k).b).A*RJ(k).sb;
        end

        % Insert
        if RJ(k).drive==2 || RJ(k).drive==0
            kk = k;
            u  = 1;
        elseif RJ(k).drive==1
            kk = RJ(k).drivenby;
            u  = RJ(k).u;
        elseif RJ(k).drive==3
            kk = RJ(k).drivenby;
            u  = -RJ(k).u;
        end
        C([1:3]+6*(i-1),[9+axis]+6*(kk-1)) = C([1:3]+6*(i-1),[9+axis]+6*(kk-1)) + skew↙
(d_k)*e*mass/u;
    end
end

% Fill in MR_k
for k=1:cst.nRJs

    % Get data
    a       = RJ(k).a;
    b       = RJ(k).b;
    axis    = RJ(k).axis;
    notaxis = [1 2 3]'; notaxis(axis)=[];
    Aa      = B(a).A;
    Ab      = B(b).A;

    % body a:
    C([4:6]+6*(a-1),[9+notaxis(1)]+6*(k-1)) = C([4:6]+6*(a-1),[9+notaxis(1)]+6*(k-1)) +↙
Aa(:,notaxis(1));
    C([4:6]+6*(a-1),[9+notaxis(2)]+6*(k-1)) = C([4:6]+6*(a-1),[9+notaxis(2)]+6*(k-1)) +↙
Aa(:,notaxis(2));
    % body b:
    C([4:6]+6*(b-1),[9+notaxis(1)]+6*(k-1)) = C([4:6]+6*(b-1),[9+notaxis(1)]+6*(k-1)) -↙
Ab(:,notaxis(1));
    C([4:6]+6*(b-1),[9+notaxis(2)]+6*(k-1)) = C([4:6]+6*(b-1),[9+notaxis(2)]+6*(k-1)) -↙
Ab(:,notaxis(2));
end


% Fill in Mg
% drive=1: a=body w. gear stator ; b=body on gear output shaft
% drive=3: a=body on gear output shaft ; b=body w. gear stator
for k=1:cst.nRJs

    if RJ(k).drive==1 || RJ(k).drive==3
        axis  = RJ(k).axis;
        a     = RJ(k).a;
        b     = RJ(k).b;
        m     = RJ(RJ(k).drivenby).b; % motor rotor body
        A     = B(a).A;
        e     = A*RJ(k).e;
        u     = RJ(k).u;
```

```
        if B(m).Mm*RJ(k).psiD>0
           mu = RJ(k).mu;
        else
           mu = 1/RJ(k).mu;
        end
        u     = u*mu;

        switch RJ(k).drive
            case 1
                % e(1-1/Ug) on body with gear stator: a
                C([4:6]+6*(a-1),[9+axis]+6*(k-1)) = C([4:6]+6*(a-1),[9+axis]+6*(k-1)) -↙
e*(1-(1/u));
                % e(1/U) on driving motors rotor body: m
                C([4:6]+6*(m-1),[9+axis]+6*(k-1)) = C([4:6]+6*(m-1),[9+axis]+6*(k-1)) -↙
e*(1/u);
                % e on driven body: b
                C([4:6]+6*(b-1),[9+axis]+6*(k-1)) = C([4:6]+6*(b-1),[9+axis]+6*(k-1)) +↙
e;
            case 3
                % e(1-1/Ug) on body with gear stator: b
                C([4:6]+6*(b-1),[9+axis]+6*(k-1)) = C([4:6]+6*(b-1),[9+axis]+6*(k-1)) -↙
e*(1-(1/u));
                % e(1/U) on driving motors rotor body: m
                C([4:6]+6*(m-1),[9+axis]+6*(k-1)) = C([4:6]+6*(m-1),[9+axis]+6*(k-1)) -↙
e*(1/u);
                % e on driven body: a
                C([4:6]+6*(a-1),[9+axis]+6*(k-1)) = C([4:6]+6*(a-1),[9+axis]+6*(k-1)) +↙
e;
        end
    end
end
```

```
function [ qDD ]=ExtractAccelerations(x)
% Extracts accelerations from x.

global RJ B cst

% Extract q(r1 w1 psi1..17)
qDD(1:6,1) = x(1:6);

j=0;
for i=1:cst.nBodies
    if B(i).motor==1
        j        = j+1;
        axis     = RJ(B(i).PrevJoint).axis;
        qDD(6+j) = x(6*(i-1)+3+axis);
    end
end

qDD=[qDD; x(end-6); x(end)];
```

```
function [ qD ]=ExtractVelocities()
% Extracts velocities, for integration.

global RJ B cst

% Extract qD(r1D w1D psiD1..17)
qD(1:3,1) = B(1).rD;
qD(4:6,1) = B(1).omega;

j=0;
for k=1:cst.nRJs
    if RJ(k).drive==2 || RJ(k).drive==0
        j=j+1;
        qD(6+j,1) = RJ(k).psiD;
    end
end
```

```
function []=TrapezoidIntegration(tStep,qD,qD1,qDD,qDD1)
% Performs time integration by the trapezoidal rule.
% Updates base body positions and orientation

global RJ B cst

% Deal with 1st time step
if tStep==1
    qD1=qD;
    qDD1=qDD;
end

% get values
rDD    = (qDD(1:3)+qDD1(1:3))/2;
omegaD = (qDD(4:6)+qDD1(4:6))/2;
omega  = ( qD(4:6)+ qD1(4:6))/2;
psiDD  = (qDD(7:end)+qDD1(7:end))/2;
psiD   = ( qD(7:end)+ qD1(7:end))/2;
dt     = cst.dt;

% Integrate acceleration to obtain velocity and position
B(1).r   = B(1).rD*dt + B(1).r;
B(1).rD  = rDD*dt + B(1).rD;

% Integrate angular acceleration to obtain angular velocity
UpdateBaseBodyAngles(B(1).omega);
B(1).omega  = omegaD*dt + B(1).omega;

% Integrate relative angular accelerations
j=0;
for k=1:cst.nRJs
    if RJ(k).drive==2  || RJ(k).drive==0
       j = j+1;                 %motor no.
       psi = RJ(k).psi;

       RJ(k).psi   = RJ(k).psiD*dt + psi;
       RJ(k).psiD  = psiDD(j)*dt + RJ(k).psiD;
       RJ(k).psiDD = psiDD(j);
    end
end
```

```
function []=SortResults(x,qDD)
% Sorts results and updates variables.

global RJ B cst
rDD=qDD(1:3);
omegaD=qDD(4:6);
psiDD=qDD(7:end);

% Update base body accelerations
B(1).rDD = rDD;
B(1).omegaD = omegaD;

% Extract reaction forces and moments for all RJ
for k=1:cst.nRJs
    axis = RJ(k).axis;
    notaxis = [1 2 3]'; notaxis(axis)=[];

    RJ(k).R=x([7:9]+6*(k-1));
    RJ(k).M=x([9+notaxis(:)]+6*(k-1));
    if RJ(k).drive==1 || RJ(k).drive==3
        RJ(k).Mg=x([9+axis]+6*(k-1));
    end
end

% Update the relative angular coordinate psi for non-motor RJs
j=0;
for k=1:cst.nRJs
    if RJ(k).drive==2
       j = j+1;                 %motor no.
       i = RJ(k).drives;        %driven RJ no.
       if RJ(RJ(k).drives).drive==1
           u = RJ(i).u;   %gearing ratio
       else
           u = -RJ(i).u;  %gearing ratio
       end

       RJ(i).psiDD = RJ(k).psiDD/u;
       RJ(i).psiD  = RJ(k).psiD/u;
       RJ(i).psi   = RJ(k).psi/u;
    end
end
```

```
function []=CalcAccelerations()
% Calculate cartesian accelerations

global B RJ cst

% Angular:
for i=2:cst.nBodies

    joints=B(i).JointPath;
    B(i).omegaD = B(1).omegaD+B(i).gamma_w;

    for k=1:length(joints)
        j     = joints(k);
        psiDD = RJ(j).psiDD;
        A     = B(RJ(j).a).A;
        e     = A*RJ(j).e;
        B(i).omegaD = B(i).omegaD + psiDD*e;
    end
end


% Linear:
for i=2:cst.nBodies

   joints=B(i).JointPath;
   m=length(joints);
   d_0  = B(i).r - B(1).r;

   B(i).rDD = B(1).rDD -skew(d_0)*B(1).omegaD + B(i).gamma_r;

   for k=1:m

        % Get book-keeping data
        j       = joints(k);
        a       = RJ(j).a;
        A       = B(a).A;
        r_i     = B(i).r;
        r_k     = B(a).r-A*RJ(j).sa;
        e_loc   = RJ(j).e;
        e       = A*e_loc;
        psiDD   = RJ(j).psiDD;

        % Get geometry
        if k<m
           d_k  = r_i - r_k; % From current RJ to i'th body's CoM
        else
           d_k  = B(RJ(j).b).A*RJ(j).sb;
        end

        B(i).rDD = B(i).rDD - skew(d_k)*e*psiDD;
   end
end
```

```
function r_CoM=CalcCoM()
% Calculates the Center of Mass for the entire robot.

global cst B

nBodies=cst.nBodies;
TotMass=0;

for i=1:nBodies
    TotMass=TotMass+B(i).m;
end

sum_mr=0;
for i=1:nBodies
    m=B(i).m;
    r=B(i).r;
    sum_mr=sum_mr+m*r;
end
r_CoM=sum_mr/TotMass;
```

```
function [ZMP]=CalcZMP()
% Calculates the Zero Moment Point (ZMP)

global B RJ cst

% Calculate common denominator
den=0;
for i=1:cst.nBodies
    m=B(i).m;
    rDD_z=B(i).rDD(3);
    den=den + m*(rDD_z+cst.g);
end

% Calculate components of ZMP
num_x=0; num_y=0;
for i=1:cst.nBodies
    % Get input
    m=B(i).m;

    r_x=B(i).r(1);
    r_y=B(i).r(2);
    r_z=B(i).r(3);

    rDD_x=B(i).rDD(1);
    rDD_y=B(i).rDD(2);
    rDD_z=B(i).rDD(3);

    J=B(i).A*B(i).J*B(i).A';
    omega=B(i).omega;
    omegaD=B(i).omegaD;

    Ldot = J*omegaD + skew(omega)*J*omega;

    % Calculate numerator
    num_x = num_x + m*(-rDD_z+cst.g)*r_x + m*rDD_x*r_z - Ldot(2);
    num_y = num_y + m*(rDD_z+cst.g)*r_y - m*rDD_y*r_z + Ldot(1);
end

% Compute ZMP
zmp_x = num_x/den;
zmp_y = num_y/den;
zmp_z = 0;

% apply low pass filter
alpha = cst.dt/(5*cst.dt+cst.dt);
zmp_x2 = alpha*zmp_x + (1-alpha)*cst.r_zmp(1);
zmp_y2 = alpha*zmp_y + (1-alpha)*cst.r_zmp(2);

% Return results
ZMP(1:3,1)=[zmp_x2 zmp_y2 zmp_z];
```

```
function []=SaveResults2(INtStep,SaveSteps)
% Saves selected results to results struct.

global RJ B F E Results cst

if rem(INtStep,SaveSteps)==0

    % if only saving in intervals
    tStep=cst.SaveStep+1;
    cst.SaveStep=tStep;

    % Body related
    Results(tStep).A={B(:).A};
    Results(tStep).r={B(:).r};
    Results(tStep).rD={B(:).rD};
    Results(tStep).rDD={B(:).rDD};
    Results(tStep).omega={B(:).omega};
    Results(tStep).omegaD={B(:).omegaD};
    Results(tStep).Mm={B(:).Mm};
    Results(tStep).r_com=cst.r_com;
    Results(tStep).r_zmp=cst.r_zmp;
    Results(tStep).R_ext=cst.R_ext;

    % RJ related
    Results(tStep).R={RJ(:).R};
    Results(tStep).M={RJ(:).M};
    Results(tStep).Mg={RJ(:).Mg};
    Results(tStep).psi={RJ(:).psi};
    Results(tStep).psiD={RJ(:).psiD};
    Results(tStep).psiDD={RJ(:).psiDD};
    Results(tStep).psiRef={RJ(:).psiRef};

    % Foot related
    if cst.FootForces==1
    Results(tStep).F(1).Fsum=F(1).Fsum;
    Results(tStep).F(1).Fs=F(1).Fs;
    Results(tStep).F(1).Fd=F(1).Fd;
    Results(tStep).F(1).Ms=F(1).Ms;
    Results(tStep).F(1).Md=F(1).Md;
    Results(tStep).F(1).Msum=F(1).Msum;
    Results(tStep).F(1).contact=F(1).contact;
    Results(tStep).F(1).dTheta=F(1).dTheta;
    Results(tStep).F(1).rD=F(1).rD;
    Results(tStep).F(1).dL=F(1).dL;
    Results(tStep).F(1).omega=F(1).omega;

    Results(tStep).F(2).Fsum=F(2).Fsum;
    Results(tStep).F(2).Fs=F(2).Fs;
    Results(tStep).F(2).Fd=F(2).Fd;
    Results(tStep).F(2).Ms=F(2).Ms;
    Results(tStep).F(2).Md=F(2).Md;
    Results(tStep).F(2).Msum=F(2).Msum;
    Results(tStep).F(2).contact=F(2).contact;
    Results(tStep).F(2).dTheta=F(2).dTheta;
    Results(tStep).F(2).rD=F(2).rD;
    Results(tStep).F(2).dL=F(2).dL;
```

```
    Results(tStep).F(2).omega=F(2).omega;
    end

    % Energy related
    Results(tStep).E=E;

end
```

```
function [CstResults]=SaveConstants()
% saves time invariant results.

global cst B RJ F

CstResults.nSteps=cst.nSteps;
CstResults.nRJs=cst.nRJs;
CstResults.nBodies=cst.nBodies;
CstResults.ismotor={B(:).motor};
CstResults.s=cst.s;
CstResults.a={RJ(:).a};
CstResults.b={RJ(:).b};
CstResults.sa={RJ(:).sa};
CstResults.sb={RJ(:).sb};
CstResults.PrevJoint={B(:).PrevJoint};
CstResults.axis={RJ(:).axis};
CstResults.hand = cst.hand;
CstResults.toes = cst.toes;
CstResults.heel = cst.heel;
```

```
%%%%% CalcCalib_Main.m %%%%%
%Set up calibration matrix and calculate forces/moments
clear all;clc,close all;
%%% Plot results: Plot=1
PlotResults=1;

% load forces and moments
[F,Fn,Ftest]=ReactionForceMoment;

% Load measured strain
[S1]=Model1;
[S2]=Model2;
[S3]=Model3;
[S4]=Model4;
[S5]=Model5;
[S6]=Model6;
[ST1]=Test1;

% Measured strain for SC
V=[S1(:,3) S2(:,3) S3(:,3) S4(:,3) S5(:,3) S6(:,3)];

% Strain matrix (measured) for LSC
s=[S1 S2 S3 S4 S5 S6];
L=[F(:,:,1) F(:,:,2) F(:,:,3) F(:,:,4) F(:,:,5) F(:,:,6)];

% SC calibration
Kn=Fn*inv(V);

% LSC calibration
K=(L*s'*inv(s*s'));

% Calculated load (Load().App= measured strain)
for i=1:4
    LoadCalcn(:,i)=Kn*ST1(:,i); % SC method
    LoadCalc(:,i)=K*ST1(:,i); % LSC method
end
if PlotResults==1
PlotLoadCase(LoadCalc,LoadCalcn,Ftest);
end

PlotLoad=1;
if PlotLoad==1
    for i=1:4
    fprintf('    Actual       Calculated \n');
    fprintf('Fx  [N]= %6.1f    %6.1f    %6.1f    \n',Ftest(1,i),LoadCalc(1,i),LoadCalcn↙
(1,i));
    fprintf('Fy  [N]= %6.1f    %6.1f    %6.1f    \n',Ftest(2,i),LoadCalc(2,i),LoadCalcn↙
(2,i));
    fprintf('Fz  [N]= %6.1f    %6.1f    %6.1f    \n',Ftest(3,i),LoadCalc(3,i),LoadCalcn↙
(3,i));
    fprintf('Mx [Nm]= %6.1f    %6.1f    %6.1f    \n',Ftest(4,i),LoadCalc(4,i),LoadCalcn↙
(4,i));
    fprintf('My [Nm]= %6.1f    %6.1f    %6.1f    \n',Ftest(5,i),LoadCalc(5,i),LoadCalcn↙
(5,i));
    fprintf('Mz [Nm]= %6.1f    %6.1f    %6.1f    \n\n',Ftest(6,i),LoadCalc(6,i),↙
LoadCalcn(6,i));
```

```
    end
end
% Root-mean square error in the measurments
for i=1:6
    for k=1:4
    A=[LoadCalc(i,k) Ftest(i,k)]';
    B=[LoadCalcn(i,k) Ftest(i,k)]';
    VarX1(i,k)=var(A);
    VarX2(i,k)=var(B);
    RmsError1(i,k)=sqrt(VarX1(i,k));
    RmsError2(i,k)=sqrt(VarX2(i,k));
    end
    MeanError1(i)=mean(RmsError1(i,:));
    MeanError2(i)=mean(RmsError2(i,:));
    Percent1(i)=abs(MeanError1(i)*100/mean(Ftest(i,:))); % RMS-error for SC
    Percent2(i)=abs(MeanError2(i)*100/mean(Ftest(i,:))); % RMS-error for LSC
end
for k=1:6
    Plotdeviations=0; % if 1 plot RMS-error in percent
    if Plotdeviations==1
        switch k
            case 1
                fprintf('Fx, RMS-error \n');
                fprintf('First method of calibration, per cent =     %6.3f   \n',↙
Percent1(k));
                fprintf('Least square calibration, percent =         %6.3f   \n\n',↙
Percent2(k));
            case 2
                fprintf('Fy, RMS-error \n');
                fprintf('First method of calibration, per cent =     %6.3f   \n',↙
Percent1(k));
                fprintf('Least square calibration, percent =         %6.3f   \n\n',↙
Percent2(k));

            case 3
                fprintf('Fz, RMS-error \n');
                fprintf('First method of calibration, per cent =     %6.3f   \n',↙
Percent1(k));
                fprintf('Least square calibration, percent =         %6.3f   \n\n',↙
Percent2(k));
            case 4
                fprintf('Mx, RMS-error \n');
                fprintf('First method of calibration, per cent =     %6.3f   \n',↙
Percent1(k));
                fprintf('Least square calibration, percent =         %6.3f   \n\n',↙
Percent2(k));
            case 5
                fprintf('My, RMS-error \n');
                fprintf('First method of calibration, per cent =     %6.3f   \n',↙
Percent1(k));
                fprintf('Least square calibration, per cent =        %6.3f   \n\n',↙
Percent2(k));
            case 6
                fprintf('Mz, RMS-error \n');
                fprintf('First method of calibration, per cent =     %6.3f   \n',↙
Percent1(k));
```

```
                fprintf('Least square calibration, percent =         %6.3f   \n\n',↙
Percent2(k));
        end
    end
end
```

```
function[Strain]=Model1()

% M1 Load Case
load TestData/Model1/5kg.mat
Load(1).M1=data(1:501,2:7);

load TestData/Model1/15kg.mat
Load(2).M1=data(1:501,2:7);

load TestData/Model1/20kg.mat
Load(3).M1=data(1:501,2:7);

load TestData/Model1/30kg.mat
Load(4).M1=data(1:501,2:7);
% The average value of 10s load measurment
for i=1:4
    Strain(:,i)=mean(Load(i).M1);

end
% Check linearity of applied load
PlotStrain=0;
if PlotStrain==1
    kg=[ 5 15 20 30];
    for i=1:6
    figure
    plot(kg,Strain(i,:))
    end
end
```

```
function[Strain]=Model2()

% M2 Load Case
load TestData/Model2/5kg.mat
Load(1).M2=data(1:501,2:7);

load TestData/Model2/15kg.mat
Load(2).M2=data(1:501,2:7);

load TestData/Model2/20kg.mat
Load(3).M2=data(1:501,2:7);

load TestData/Model2/30kg.mat
Load(4).M2=data(1:501,2:7);


for i=1:4
    Strain(:,i)=mean(Load(i).M2);
end

PlotStrain=0;
if PlotStrain==1
    kg=[ 5 15 20 30];
    for i=1:6
    figure
    plot(kg,Strain(i,:))
    end
end
```

```
function[Strain]=Model3()

% M1 Load Case
load TestData/Model3/5kg.mat
Load(1).M3=data(1:501,2:7);

load TestData/Model3/15kg.mat
Load(2).M3=data(1:501,2:7);

load TestData/Model3/20kg.mat
Load(3).M3=data(1:501,2:7);

load TestData/Model3/30kg.mat
Load(4).M3=data(1:501,2:7);


for i=1:4
    Strain(:,i)=mean(Load(i).M3);
end

PlotStrain=0;
if PlotStrain==1
    kg=[5 15 20 30];
    for i=1:6
    figure
    plot(kg,Strain(i,:))
    end
end
```

```
function[Strain]=Model4()

% M4 Load Case
load TestData/Model4/5kg.mat
Load(1).M4=data(1:501,2:7);

load TestData/Model4/15kg.mat
Load(2).M4=data(1:501,2:7);

load TestData/Model4/20kg.mat
Load(3).M4=data(1:501,2:7);

load TestData/Model4/30kg.mat
Load(4).M4=data(1:501,2:7);


for i=1:4
    Strain(:,i)=mean(Load(i).M4);

end

PlotStrain=0;
if PlotStrain==1
    kg=[ 5 15 20 30];
    for i=1:6
    figure
    plot(kg,Strain(i,:))
    end
end
```

```
function[Strain]=Model5()

% M5 Load Case
load TestData/Model5/5kg.mat
Load(1).M5=data(1:501,2:7);

load TestData/Model5/15kg.mat
Load(2).M5=data(1:501,2:7);

load TestData/Model5/20kg.mat
Load(3).M5=data(1:501,2:7);

load TestData/Model5/30kg.mat
Load(4).M5=data(1:501,2:7);


for i=1:4
    Strain(:,i)=mean(Load(i).M5);

end

PlotStrain=0;
if PlotStrain==1
    kg=[ 5 15 20 30];
    for i=1:6
    figure
    plot(kg,Strain(i,:))
    end
end
```

```
function[Strain]=Model6()

% M6 Load Case
load TestData/Model6/5kg.mat
Load(1).M6=data(1:501,2:7);

load TestData/Model6/15kg.mat
Load(2).M6=data(1:501,2:7);

load TestData/Model6/20kg.mat
Load(3).M6=data(1:501,2:7);

load TestData/Model6/30kg.mat
Load(4).M6=data(1:501,2:7);


for i=1:4
    Strain(:,i)=mean(Load(i).M6);

end

PlotStrain=0;
if PlotStrain==1
    kg=[ 5 15 20 30];
    for i=1:6
    figure
    plot(kg,Strain(i,:))
    end
end
```

```
function [Load,Loadn, LoadT]=ReactionForceMoment
% Calculate reaction forces and moments in FTS
clc; clear all;
% Support-plate
L3=0.18; %[m]
% Distance piece
Lm123=0.05;%[m] for model 1,2,3
Lm456=0.075;
LT1=0.025;
% Angle of support-plate
SP1=asind(Lm123/L3);%[deg]
SP2=asind(Lm456/L3)+0.5;
SPT1=asind(LT1/L3)+0.5;
SPX=-1.5;
% Rotation of FTS
theta_x=[ SPX    SPX    SPX    SPX   SPX   SPX];  %[dec]
theta_y=[-SP1  -SP1  -SP1  -SP2 -SP2 -SP2];
theta_z=[-10  -20  -40 -10 -20 -40];
% Rotation of FTS in test
theta_x_T=0;  %[deg]
theta_y_T=-SPT1;
theta_z_T=-20;
% Length of beams
L1=0.4675; %[m]
L2=0.0605; %[m]
L4=0.071; %[m]
%local s-vector
s=[0 -L2 L1]';
s_=skew(s);
ST=[0 -L4 L1]';
ST_=skew(ST);
% Applied force
a=9.82; %[m/s^2]
M=[5 15 20 30]; %[kg]
% Test force
MT=[10 25 40 50]; %[kg]
for k=1:4
    Fn(:,k)=[0 0 -M(k)*a]'; %[N] calibration forces
    FnT(:,k)=[0 0 -MT(k)*a]'; %[N] test forces
end
% calculate reaction forces and moments
for k=1:4
    for i=1:6
    q=[theta_x(i) theta_y(i) theta_z(i)]';
    [Am]=BryantTrMatrix(deg2rad(q));
    Force(:,k,i)=Am'*Fn(:,k);
    Moment(:,k,i)=s_*Am'*Fn(:,k);
    Load(:,k,i)=[Am'*Fn(:,k);s_*Am'*Fn(:,k)];
    end
end
% Claculate test force and moment
for k=1:4
    for i=1:1
        qT=[theta_x_T(i) theta_y_T(i) theta_z_T(i)]';
        [AmT]=BryantTrMatrix(deg2rad(qT));
        LoadT(:,k,i)=[AmT'*FnT(:,k);ST_*AmT'*FnT(:,k)];
```

```
    end
end
% Calibration load for SC method
Loadn=[Load(:,3),Load(:,7),Load(:,11),Load(:,15),Load(:,19),Load(:,23)];
%Force
%Moment
%LoadT
```

```
function[Strain]=Test1()

% Test1 Load Case

load TestData/Test1/10kg.mat
Load(1).T1=data(1:501,2:7);
load TestData/Test1/25kg.mat
Load(2).T1=data(1:501,2:7);
load TestData/Test1/40kg.mat
Load(3).T1=data(1:501,2:7);
load TestData/Test1/50kg.mat
Load(4).T1=data(1:501,2:7);

for i=1:4
Strain(:,i)=mean(Load(i).T1);
end

PlotStrain=0;
if PlotStrain==1
    kg= [10 25 40 50];
    for i=1:6
    figure
    plot(kg,Strain(i,:))
    end
end
```

```
%%%%% ComplexBanana_main %%%%%
clc;clear all; close all;
% Search for optimum of the banana function
% Optimization parameter
m=2;
% Boundaries
low=[-1,-1];
high=[2,2];
% Number of points
n=4;
CalcF=@(x)100*(x(2)-x(1)^2)^2+(1-x(1))^2;
x0=random('normal',0,1,m,n);% Starting point, randomly generated
[x0]=CheckLimit2(x0,high,low,n,m);% Check if boundries are respected
for j=1:n; F(:,j)=CalcF(x0(:,j)); end % Calc objective function
[x_rO,x_cO,PlaceVO,PlaceBO]=WorstPoint2(F,x0,n,m); % find worst place (PlaceV)-Old(O),↙
best place(PlaceB),centroid (x_c) and reflected point (x_r)
[x0]=Boundaries2(x0,x_rO,high,low,PlaceVO,m); % check if the boundaries are maintained↙
and return the updated vector
F(:,PlaceVO)=CalcF(x0(:,PlaceVO));% calc objective-function at the new point
%%%%%%PLOT%%%%%%%
plot_x=(-1:0.05:2);
plot_y=(-1:0.05:2);
for i=1:length(plot_x)
    for j=1:length(plot_y)
        plot_F(i,j)=CalcF([plot_x(i);plot_y(j)]);
    end
end
figure
hold on; grid on
surfc(plot_x,plot_y,plot_F);
zlim([0,10])
view(35,30)
for i=1:n
plot3(x0(i),x0(i),F(i),'go')
end
xlabel('x')
ylabel('y')
%%%%%%%%%%

kr=0;
for j=1:10000
    [x_r,x_c,PlaceV,PlaceB]=WorstPoint2(F,x0,n,m);
    [x0]=Boundaries2(x0,x_r,high,low,PlaceV,m);

    %F(:,PlaceV)=CalcF(x0(:,PlaceV)); % Calc new value

    if PlaceVO==PlaceV % if the same place is the worst; kr+1
            kr=kr+1;
    else
            kr=0;
    end

    if PlaceVO==PlaceV
        n0=4;
        epsx=(n0/(n0+kr-1))^((n0+kr-1)/n0);
        R=rand(1);
```

```
        x_r=(x_r+epsx*x_c'+(1-epsx)*x0(:,PlaceB))/2+((x_c'-x0(:,PlaceB))*(1-epsx)*(2*R-↙
1));% x_r_new

    end
    [x0]=Boundaries2(x0,x_r,high,low,PlaceV,m);
    F(PlaceV)=CalcF(x0(:,PlaceV));


    Skr(j)=kr;
    x0p=x0(:,PlaceV);
    plot3(x0p(1),x0p(2),F(PlaceV),'rx')
    pause(0.00001)
    PlaceVO=PlaceV;

     if (abs(max(F)-min(F))<0.0004), break, end
end
fprintf('Number of iteration: %i\n',j)
fprintf('Value of the variabels')
x0
```

```
function [z]=Boundaries2(z,zr,high,low,PlaceV,m)
a_old=1;b_old=1;b_new=0;a_new=0;
for i=1:m % check if boundaries is maintained.
    if ((zr(i)<high(i)) && (zr(i)>low(i)))
        z(i,PlaceV)=zr(i);              % if maintained; x_worst=x_mir
    elseif (zr(i))>(high(i))
        b=high(i)/zr(i);
        if b<b_old
            b_new=b;
        end
        b_old=b;

    elseif (zr(i))<(low(i))
        a=low(i)/zr(i);
        if a<a_old
            a_new=a;
        end
        a_old=a;
    end
end


C=b_new+a_new;
  if ((C>0) && (b_new < a_new)) % explicit constraints
    z1(1)=z(1)*b_new*0.99999;
    z1(2)=z(2)*b_new*0.99999;

  end

  if ((C>0) && (b_new > a_new))
    z1(1)=z(1)*a_new*1.000001;
    z1(2)=z(2)*a_new*1.000001;

  end


if C>0
  z(:,PlaceV)=z1';
end
```

```
function [z]=CheckLimit2(zIn,high,low,n,m)%F,CalcF
z=zIn;
for i=1:m
    for j=1:n % check if the value respects the boundry-limit
         if (z(i,j))>high(1)
             z(i,j)=high(1);
            % F(i,j)=CalcF(high(1));
         elseif zIn(i,j)<low(1)
             z(i,j)=low(1);
             %F(i,j)=CalcF(low(1));
         end
    end
end
```

```
function [zr,z_c,PlaceV,PlaceB]=WorstPoint2(F,z0,n,m)

[x_worst,PlaceV]=max((F));
[x_best,PlaceB]=min((F));
alpha=1.3;
for i=1:m
z_sum=sum(z0(i,:)); % sum of the points( without the worst point)
z_c(i)=1/(n-1)*(z_sum-z0(i,PlaceV));% Centroid
end
zr=z_c'+alpha*(z_c'-z0(:,PlaceV)); %Mirrored point,new value for worst point
```

```
%%%%% ComplexAnsys %%%%%
clc;clear all; close all;
% Objective penalties
ObjVolu=100; %maximum volume
ObjStress=150; %maximum stress

 %Optimization parameter
m=2;
% boundaries
low=[2,2];
high=[50,50];
% Number of points
n=5;
% Starting point, randomly generated
x0=high(1)/2*random('normal',0,1,m,n);
% Calculate the value of the objection function
[x0]=CheckLimitAnsys(x0,high,low,n,m);% Check if boundaries are respected
StartXo=x0; % save start guess
fprintf('Number of points: %i',n)
for i=1:n
    WriteAnsysFile(x0(:,i),i); % write Ansys file
    !RunAnsys.bat;
    [Vol,MaxStress]=ReadAnsysResult(i); % read results and calculate objective-function
    F(i)=ObjectFunction(ObjVolu,Vol,ObjStress,MaxStress);
    SVol_s(i)=Vol;SMaxStress_s(i)=MaxStress; % Save start volume and max stresses for↙
each guess
    clc;
    fprintf('Calculate objection function for start point no:  %i',i)
end
SF_s=F; % Save start value for objective-function
[x_rO,x_zO,PlaceVO,PlaceBO]=WorstPointAnsys(F,x0,n,m); % find worst place (PlaceV)-Old↙
(O), best place(PlaceB),centroid (x_c) and reflected point (x_r)
[x0]=BoundariesAnsys(x0,x_rO,high,low,PlaceVO,m); % check if the boundaries are↙
maintained and return the updated vector
WriteAnsysFile(x0(:,PlaceVO));
!RunAnsys.bat;
[VolO,MaxStressO]=ReadAnsysResult;% calc objective-function at the new point
F(PlaceVO)=ObjectFunction(ObjVolu,VolO,ObjStress,MaxStressO);
ItrNo=1;
clc;
fprintf('Number of iteration:  %i',1);
kr=0; % Start value for repeated place counter
tEnd=0;
for j=1:3000
    clo=clock; % Time counter
    tStart=clo(6);
    [x_r,x_c,PlaceV,PlaceB]=WorstPointAnsys(F,x0,n,m); % Find worest place, best place,↙
centroid, and reflected point
    [x0]=BoundariesAnsys(x0,x_r,high,low,PlaceV,m); % Check boundaries
    ItrNo=j+1; % iteration counter
    clc
    fprintf('Number of iteration: %i\n',ItrNo);
    fprintf('Calculation time: %6.2f\n',tEnd);
    fprintf('Worst value KP1:    %6.3f\n',x0(2,PlaceV));
    fprintf('Worst value KP2:    %6.3f\n',x0(1,PlaceV));
    fprintf('Worst volume:mm^3   %6.3f\n',Vol);
```

```
    fprintf('Worst stress:MPa    %6.2f\n',MaxStress);
    fprintf('Worst and best Place:  %i   %i\n',PlaceV,PlaceB);
    x0
    F
    if PlaceVO==PlaceV % if the same place is the worst, then penalty constant; kr+1
        kr=kr+1;
    else
        kr=0;
    end

    if PlaceVO==PlaceV
        n0=4;
        epsx=(n0/(n0+kr-1))^((n0+kr-1)/n0);
        R=rand(1,2);
        for i=1:m
            x_new(:,i)=(x_r(i)+epsx*x_c(i)+(1-epsx)*x0(i,PlaceB))/2+((x_c(i)-x0(i,↙
PlaceB))*(1-epsx)*(2*R(1)-1));
        end
        x_r=x_new;

    end
    [x0]=BoundariesAnsys(x0,x_r,high,low,PlaceV,m);
    WriteAnsysFile(x0(:,PlaceV));
    !RunAnsys.bat;
    [Vol,MaxStress]=ReadAnsysResult;% calc objective-function at new point
    F(PlaceV)=ObjectFunction(ObjVolu,Vol,ObjStress,MaxStress);
    PlaceVO=PlaceV; % Save place as old place
    SP(j)=PlaceV; % Save place
    Skr(j)=kr; % Save penalty constant
    Sx0(:,j)=x0(:,PlaceV); % Save value of worst coordinate
    SMaxStress(j)=MaxStress; % Save max. stress
    SVol(j)=Vol; % Save volume of model
    SF(ItrNo)=max(F); % Save max value of objective-function
    if abs(max(F)-min(F))<0.04  , break, end % if difference between max value of F and↙
min value of F is under 0.004; stop
    clo=clock;
    tEnd=clo(6)-tStart;
end
%%%%%%PLOT
figure
hold on
grid on
xplot=(1:j);
plot(xplot,Sx0(2,:),'rx')
plot(xplot,Sx0(1,:),'go')
legend('KP 1','KP 2')
xlabel('Iteration No')
ylabel('Hight of key-point')
%%%%%%%%%%
```

```
function [z]=BoundariesAnsys(z,zr,high,low,PlaceV,m)
for i=1:m
    if ((zr(i)<high(i)) && (zr(i)>low(i))) % check if implicit boundaries
             z(i,PlaceV)=zr(i);  % if maintained; x_worst=x_mir
    elseif zr(i)>=high(i)
             z(i,PlaceV)=high(i);
    elseif zr(i)<=low(i)
             z(i,PlaceV)=low(i);
    elseif (zr(1)+zr(3))>high(3);% explicit values respect boundary
             z(1,PlaceV)=0.5*high(1);
             z(3,PlaceV)=0.5*high(3);
    elseif (zIn(2)+zIn(4))>high(4);
             z(2,PlaceV)=0.5*high(2);
             z(4,PlaceV)=0.5*high(4);
    end
end
```

```
function [z]=CheckLimitAnsys(zIn,high,low,n,m)%
z=zIn;
for i=1:m
    for j=1:n % check if the implicit values respects boundary
         if (z(i,j))>high(m)
             z(i,j)=high(m);

         elseif zIn(i,j)<low(m)
             z(i,j)=low(m);
             %F(i,j)=CalcF(low(1));
         end
    end
end
```

```
function [Obj]=ObjectFunction(ObjVolu,Vol,ObjStress,StressMax)

if StressMax<=ObjStress
    PenStress=0;
else
    PenStress=exp((StressMax-ObjStress)*10);
end

Obj=(Vol/100)^2+PenStress; % Objective function for beam model
```

```
function [Vol,StressMax]=ReadAnsysResult(i)
% Read results
% von Mises stress at each element

    outputfile = 'D:\Ansys\Model1Stress.1' ;
    fid = fopen(outputfile,'r');

    %Read one line at the time
    while (feof(fid)~=1)
        line=fgetl(fid);
        numline = str2num(line);
        %if the line contains two numbers then save stresses:
        if (length(numline)==2)
            elem=int8(numline(1,1));
            Stress(elem)=numline(1,2);
        end
    end
    fclose(fid);

  % Volume of each element
  outputfile = 'D:\Ansys\Model1Volu.1'  ;
    fid = fopen(outputfile,'r');
    while (feof(fid)~=1)
        line=fgetl(fid);
        numline = str2num(line);
        if (length(numline)==2)
            elem=int8(numline(1,1));
            Volu(elem)=numline(1,2);
        end
    end
    fclose(fid);


    Vol=sum(Volu); %[mm^2] Volume of geometry
    StressMax=max(Stress); % Maximum stress
```

```
@ echo off
C:
cd C:\Program Files\Ansys Inc\v100\ANSYS\bin\intel\
set ANSYS_LOCK=OFF
ansys100 -b -p ansysrf -i D:\Ansys\Model1.lgw -o D:\Ansys\Model1.out
```

```
function [zr,z_c,PlaceV,PlaceB]=WorstPointAnsys(F,z0,n,m)
% Find worst beam design and reflect it
[x_worst,PlaceV]=max((F));
[x_best,PlaceB]=min((F));
alpha=1.4;
for i=1:m
z_sum=sum(z0(i,:)); % sum of the point( without the worst point)
z_c(i)=1/(n-1)*(z_sum-z0(i,PlaceV));% Centroid
zr(i)=z_c(i)+alpha*(z_c(i)-z0(i,PlaceV)); %Mirrored point,new value for worst point
end
```

```
function []=WriteAnsysFile(x0,i)
KP1y=x0(1);
KP2y=x0(2);
%Generate filename as string
    filename ='D:\Ansys\Model1.lgw'  ;
      %Open file
    fid = fopen(filename,'w');

% Batch mode
fprintf(fid,'/BATCH  \n');
fprintf(fid,'/COM,ANSYS RELEASE 10.0A1  UP20060105\n');
fprintf(fid,'/input,menust,tmp,'',,,,,,,,,,,,,,1   \n');
fprintf(fid,'/CWD,''D:\\Ansys'' \n');
fprintf(fid,'/CPLANE,1');
fprintf(fid,'WPSTYLE,,,,,,,,0\n');
fprintf(fid,'/PREP7\n')  ;
fprintf(fid,'K,1,50,%6.2f,,\n',KP1y);   % key-points
fprintf(fid,'K,2,0,%6.2f,,  \n',KP2y);
fprintf(fid,'K,3,0,0,, \n');
fprintf(fid,'K,4,50,0,,  \n');
fprintf(fid,'LSTR,       1,       2  \n'); %set line
fprintf(fid,'LSTR,       2,       3  \n');
fprintf(fid,'LSTR,       3,       4  \n');
fprintf(fid,'LSTR,       4,       1  \n');
fprintf(fid,'FLST,2,4,4  \n');    % create area
fprintf(fid,'FITEM,2,1   \n');
fprintf(fid,'FITEM,2,2   \n');
fprintf(fid,'FITEM,2,3   \n');
fprintf(fid,'FITEM,2,4   \n');
fprintf(fid,'AL,P51X \n');
fprintf(fid,'ET,1,PLANE182\n');   % chose element
fprintf(fid,'KEYOPT,1,1,0\n');
fprintf(fid,'KEYOPT,1,3,3\n');
fprintf(fid,'KEYOPT,1,6,0\n');
fprintf(fid,'KEYOPT,1,10,0\n');
fprintf(fid,'R,1,10,\n');  % thickness of element (10)
fprintf(fid,'MPTEMP,,,,,,,,\n');
fprintf(fid,'MPTEMP,1,0  \n');
fprintf(fid,'MPDATA,EX,1,,70000  \n');% Elastic Modulus
fprintf(fid,'MPDATA,PRXY,1,,0.33\n'); % Poisson ratio
fprintf(fid,'MSHAPE,0,2D \n');   % Meshing the area
fprintf(fid,'MSHKEY,0\n');
fprintf(fid,'CM,_Y,AREA\n'); % size of mesh
fprintf(fid,'ASEL, , , ,       1\n');
fprintf(fid,'CM,_Y1,AREA \n');
fprintf(fid,'CHKMSH,''AREA''   \n');
fprintf(fid,'CMSEL,S,_Y  \n');
fprintf(fid,'AMESH,_Y1   \n');
fprintf(fid,'CMDELE,_Y   \n');
fprintf(fid,'CMDELE,_Y1  \n');
fprintf(fid,'CMDELE,_Y2  \n');%--------
fprintf(fid,'/UI,MESH,OFF\n');%
fprintf(fid,'DK,2, , , ,0,ALL, , , , , , \n'); % Displacement, DOF
fprintf(fid,'DK,3, , , ,0,ALL, , , , , , \n'); % Displacement, DOF
fprintf(fid,'FK,1,FY,-1000 \n'); %  Fy=-1000N
fprintf(fid,'/SOL\n');% solve
```

```
fprintf(fid,'/STATUS,SOLU\n');
fprintf(fid,'SOLVE  \n');
fprintf(fid,'/POST1\n');
fprintf(fid,'Model1.macro \n');

% Close file
fclose(fid);

%Generate filename as string
    filename ='D:\Ansys\Model1.macro' ;

fid = fopen(filename,'w');

for i=1:1
    fprintf(fid,'SET,%i\n',i);
    fprintf(fid,'/output,''D:\\ANSYS\\Model1Volu.%i''\n',i);
    fprintf(fid,'PRESOL,VOLU\n'); % Volume of each element
    fprintf(fid,'ETABLE,SEQV,S,EQV\n');
    fprintf(fid,'/output,''D:\\ANSYS\\Model1Stress.%i''\n',i);
    fprintf(fid,'PRETAB,SEQV\n'); % Element solution, von Mises stress
    fprintf(fid,'/out\n');
    fprintf(fid,'\n');
end
fclose(fid);
```

```
% Complex method
clc;clear all; close all;
% Objective function, max volume and stress
ObjU=0.4321; % Displacemant of the original design
ObjStress=205; %[MPa] maximum stress
VolGear=44^2*pi*3/4*20; %[mm^3] volume of gear
 %Optimization parameters
m=4;
% Boundaries (r1 r2 x1 x2)
lowX= [2,   2,  1,  1];
highX=[22, 22, 24, 24];
% Number of points
n=9;
% Starting point, randomly generated
x0_s=rand(m,n);
% Calculate the value of the objection function
for i=1:n
    for j=1:m
    x0(j,i)=x0_s(j,i)*highX(j);
    end
end
initialx0=x0;

[x0]=CheckLimitPelvis(x0,highX,lowX,n,m);% Check boundaries
StartX0=x0; % save start guess
fprintf('Number of points: %i',n)
for i=1:n
    WritePelvisFile(x0(:,i)); % wirte Ansys file
    !RunAnsysPelvis.bat;
    [Vol,MaxStress,Umax]=ReadPelvisResult(i); % read results and calculate objective-↙
function
    F(i)=ObjectFunction(Vol,ObjStress,MaxStress,ObjU,Umax,VolGear);
    SVol_s(i)=Vol-VolGear;SMaxStress_s(i)=MaxStress; SUmax(i)=Umax; % Save start volume↙
and max stresses for each guess
    Volume(i)=Vol-VolGear;
    clc;
    fprintf('Calculate objection function for start point no:  %i',i)
end
SF_s=F; % Save start value for objective function
[x_r,x_z,PlaceVO,PlaceBO]=WorstPointAnsys(F,x0,n,m); % find worst place (PlaceV)-Old↙
(O), best place(PlaceB),centroid (x_c) and reflected point (x_r)
[x0]=BoundariesPelvisOld(x0,x_r,highX,lowX,PlaceVO,m); % check if the boundaries are↙
maintained and return the updated vector
WritePelvisFile(x0(:,PlaceVO));
!RunAnsysPelvis.bat;
[Vol,MaxStress,Umax]=ReadPelvisResult;% calc objective-function at the new point
F(PlaceVO)=ObjectFunction(Vol,ObjStress,MaxStress,ObjU,Umax,VolGear);
ItrNo=1;
Volume(PlaceVO)=Vol-VolGear;
clc;
fprintf('Number of iteration:  %i',1);
kr=0; krRw=0.1;l=0;krWP=0.1;K=0;% Start value for repeated place counter
tEnd=0;

for j=1:3000 % number of iterations
    clo=clock; % Time counter
```

```
    tStart=clo(6);
    [x_r,x_c,PlaceV,PlaceB]=WorstPointAnsys(F,x0,n,m); % Find worst place, best place, centroid, and reflected point
    [x0,GoTo]=BoundariesPelvisNew(x0,x_r,highX,lowX,PlaceV,m); % Check boundaries
    if (max(Volume)-min(Volume))<0.0004  , break, end % if difference between max value of F and min value of F is under 0.004; stop abs(max(F)-min(F))
    ItrNo=j+1; % iteration counter
    if PlaceVO==PlaceV % if the same place is the worst, then penalty constant; kr+1
        GoTo=1;
        kr=kr+1;
    else
        kr=0;
    end
    clc
    fprintf('Number of iteration:                %i\n',ItrNo);
    fprintf('Calculation time:                   %6.2f\n',tEnd);
    fprintf('                          New     Old     Best   xr\n' );
    fprintf('Radius,worst/best r1:    %6.3f, %6.3f, %6.3f, %6.3f [mm]\n',x0(1,PlaceV), x0(1,PlaceVO),x0(1,PlaceB),x_r(1));
    fprintf('Radius,worst/best r2:    %6.3f, %6.3f, %6.3f, %6.3f [mm]\n',x0(2,PlaceV), x0(2,PlaceVO),x0(2,PlaceB),x_r(2));
    fprintf('Height,worst/best x1:    %6.3f, %6.3f, %6.3f, %6.3f [mm]\n',x0(3,PlaceV), x0(3,PlaceVO),x0(3,PlaceB),x_r(3));
    fprintf('Height,worst/best x2:    %6.3f, %6.3f, %6.3f, %6.3f [mm]\n',x0(4,PlaceV), x0(4,PlaceVO),x0(4,PlaceB),x_r(4));
    fprintf('Worse/best place:        %6.1i         %6.1i\n',PlaceV,PlaceB);
    fprintf('Deflection:              %6.3f [mm]\n',Umax);
    fprintf('Worse stress:            %6.3f [MPa]\n',MaxStress);
    fprintf('Repeat-counter:          %6.1i \n',kr);
    fprintf('Volume of model:         %6.3f [mm^3]\n',Vol-VolGear);
    F
    Volume
    x0

    if kr>1
    krRw=rem(kr,22);
    krWP=rem(kr,10);
    end
       while  GoTo==1 % if a constraint is violated or the worst place repeats itself, run this routine

            K=K+1;
            k_new=kr+K-1;
            clc
            fprintf('Number of l iteration:                  %i\n',K);
            fprintf('                           New     Old     Best   xr\n' );
            fprintf('Radius,worst/best r1:    %6.3f, %6.3f, %6.3f, %6.3f [mm]\n',x0(1,PlaceV),x0(1,PlaceVO),x0(1,PlaceB),x_r(1));
            fprintf('Radius,worst/best r2:    %6.3f, %6.3f, %6.3f, %6.3f [mm]\n',x0(2,PlaceV),x0(2,PlaceVO),x0(2,PlaceB),x_r(2));
            fprintf('Height,worst/best x1:    %6.3f, %6.3f, %6.3f, %6.3f [mm]\n',x0(3,PlaceV),x0(3,PlaceVO),x0(3,PlaceB),x_r(3));
            fprintf('Height,worst/best x2:     %6.3f, %6.3f, %6.3f, %6.3f [mm]\n',x0(4,PlaceV),x0(4,PlaceVO),x0(4,PlaceB),x_r(4));
            fprintf('Worset/best place:        %6.1i         %6.1i\n',PlaceV,PlaceB);
            fprintf('Repeat-counter:          %6.1i \n',kr);
```

```
            fprintf('Deflection:              %6.3f [mm]\n',Umax);
            F

             if ((kr==7)||(krWP==0))
                 [x_r,PlaceV,PlaceB]=WorstPointAnsys_new(F,x0,n,m,kr);
             end

             n0=5;
             epsx=(n0/(n0+k_new-1))^((n0+k_new-1)/n0);
             R=rand(1);
             x_r=(x_r+epsx*x_c'+(1-epsx)*x0(:,PlaceB))/2+((x_c'-x0(:,PlaceB))*(1-epsx)*↙
(2*R-1));%x_r_new
             if krRw==0
             x_r=x0(:,PlaceB);
             GoTo=0;
             end
             [x0,GoTo]=BounderiesPelvisNew(x0,x_r,highX,lowX,PlaceV,m);

        end


    WritePelvisFile(x0(:,PlaceV));
    !RunAnsysPelvis.bat;
    [Vol,MaxStress,Umax]=ReadPelvisResult;% calc objective-function at the new point
    F(PlaceV)=ObjectFunction(Vol,ObjStress,MaxStress,ObjU,Umax,VolGear);
    PlaceVO=PlaceV; % replace the PlaceV as PlaceVO
    Volume(PlaceV)=Vol-VolGear;
    % save values
    if ((PlaceBO>PlaceB) || (PlaceBO<PlaceB))
    SVB(j)=Vol; % save best volume
    end
    PlaceBO=PlaceB;

    SP(j)=PlaceV; % Save worst place
    SPb(j)=PlaceB; % Save best place
    Skr(j)=kr; % Save penalty constant
    SK(j)=K; % Save counter
    K=0;
    Sx0(:,j)=x0(:,PlaceVO); % Save value of worst coordinate
    Sx0b(:,j)=x0(:,PlaceB); % Save value of best coordinate
    SMaxStress(j)=MaxStress; % Save max. stress
    SVol(j)=Vol-VolGear; % Save volume of model
    SBVol(j)=min(Volume); % save best volume
    SF(ItrNo)=max(F); % Save max value of objective function
    SFb(ItrNo)=min(F); % Save min value of objective function

    clo=clock;
    tEnd=clo(6)-tStart;
end
OrVol=[58378,58378];
XVol=[0,j];

%%%%%%PLOT
figure
grid on
xplot=(1:j-1);
```

```
xplot1=(1:j);
plot(xplot,Sx0(1,:),'go',xplot,Sx0(2,:),'cs',xplot,Sx0(3,:),'b+',xplot,Sx0(4,:),'k*');
legend('r1','r2','x1','x2');
title('Worst Height/radius')
xlabel('Iteration No')
ylabel('Height/radius of variables [mm]')
figure
grid on
plot(xplot,Sx0b(1,:),'go',xplot,Sx0b(2,:),'cs',xplot,Sx0b(3,:),'b+',xplot,Sx0b↙
(4,:),'k*');
legend('r1','r2','x1','x2');
title('Best Height/radius')
xlabel('Iteration No')
ylabel('Height/radius of variables [mm]')
figure
grid on
subplot(2,1,1)
plot(xplot,SVol,'bo',XVol,OrVol,'r')
xlabel('Iteration No')
ylabel('Volume [mm^3]')
title('Volume of worst design')
subplot(2,1,2)
plot(xplot1,SFb,'bo')
title('Function value of best design')
xlabel('Iteration No')
ylabel('Function value')

%%%%%%%
```

```
function [x0,GoTo]=BoundariesPelvisNew(x0,x_r,highX,lowX,PlaceV,m)

GoTo=0;
for i=1:m
    Xaa=0;Xb=0;Xbb=0;Xcc=0;Xa=0;

    if ((x_r(3))<=highX(3) && ((x_r(3))>=lowX(3)))% check if boundaries are maintained.
        Xa=1;
    end

    if ((x_r(4))<=highX(4) && ((x_r(4))>=lowX(4)))% check if boundaries are maintained.
        Xb=1;
    end

    if ((x_r(1)<x_r(3)) && (x_r(1)>=lowX(1))) %   ((zr(1)<zr(3)) && (zr(1)>=lowX(1)))
         Xaa=1;
    end

    if ((x_r(2)<x_r(4)) && (x_r(2)>=lowX(2)))%
        Xbb=1;
    end

    A=Xa+Xb+Xaa+Xbb+Xcc;
    if A==4;
       x0(:,PlaceV)=x_r';
    else
       GoTo=1;
    end

end
```

```
function [z]=BoundariesPelvisOld(z,zr,highX,lowX,PlaceV,m)
B=0;
for i=1:m % check if boundaries are maintained.
    Xaa=0;Xb=0;Xbb=0;Xa=0;Xcc=0;
    if ((zr(4))<=highX(4) && ((zr(4))>=lowX(4)))
        Xb=1;
    end

    if ((zr(3))<=highX(3) && ((zr(3))>=lowX(3)))
        Xa=1;
    end

    if ((zr(1)<zr(3)) && (zr(1)>=lowX(1)))
         Xaa=1;
    end

    if ((zr(2)<zr(4)) && (zr(2)>=lowX(2)))
        Xbb=1;
    end
    A=Xa+Xb+Xaa+Xbb+Xcc;
    if A==4;
       z(:,PlaceV)=zr';
    else
        if (zr(i))>(highX(i))
             z(i,PlaceV)=highX(i)*0.9999;
        end
        if zr(i)<lowX(i)
             z(i,PlaceV)=lowX(i)*1.1;
        end
        B=1;
    end

end

if B==1

        if (z(1)>=z(3));
           z(3)=z(1)+0.6;
        end

        if (z(2)>=z(4));
            z(4)=z(2)+0.6;
        end

end
```

```
function [KP]=CalcKeyPoints(x0)
% guess
r0=x0(1);%2.5;
r1=x0(1);%2.5; % [mm]x0(1)
r2=x0(2); % [mm]
r3=x0(2);
x1=x0(3); % [mm] hight of half hole (downwards)
x2=x0(4); % [mm] hight of half hole (upwards)
% Geometric constants
Rb=50; % [mm] Radius of gear boundaries
Cb1=140; % [mm] x-coordinate for center of boundary2
LR0=Rb+r0;
LR1=Rb+r1;
hcb0=Rb/2+x1-r0;
hcb1=Rb/2+x1-r1;
LR2=Rb+r2;
LR3=Rb+r3;
% Key-Points
KP=zeros(2,21);
%%%%%% KP 10,17,18
KP(2,10)=Rb/2-x1;
KP(2,18)=KP(2,10)+r0;
KP(1,18)=sqrt(LR0^2-(hcb0)^2);
KP(1,10)=KP(1,18);
alpha0=acosd(hcb0/LR0); beta0=90-alpha0; %angles
KP(2,17)=KP(2,18)+r0*sind(beta0);
KP(1,17)=sqrt(Rb^2-(Rb-KP(2,17))^2);

%%%%%%% KP 11,12,19
KP(2,11)=Rb/2-x1;
KP(2,19)=KP(2,11)+r1;
KP(1,19)=Cb1-sqrt(LR1^2-(hcb1)^2);
KP(1,11)=KP(1,19);
alpha1=acosd(hcb1/LR1); beta1=90-alpha1; %angles
KP(2,12)=KP(2,19)+r1*sind(beta1);
KP(1,12)=Cb1-sqrt(Rb^2-(Rb-KP(2,12))^2);

%%%%%%%% KP 13,14,20
KP(2,14)=Rb/2+x2;
KP(2,20)=KP(2,14)-r3;
KP(1,20)=Cb1-sqrt(LR3^2-(Rb-KP(2,20))^2);
KP(1,14)=KP(1,20);
alpha3=asin((Rb-KP(2,20))/LR3);
KP(2,13)=KP(2,20)+(r3*sin(alpha3));
KP(1,13)=Cb1-sqrt(Rb^2-(Rb-KP(2,13))^2);
%%%%%%%% KP 15,16,21
KP(2,15)= Rb/2+x2;
KP(2,21)=KP(2,15)-r2;
KP(1,21)= sqrt(LR2^2-(Rb-KP(2,21))^2);
KP(1,15)=KP(1,21);
alpha2=asin((Rb-KP(2,21))/LR2);
KP(2,16)=KP(2,21)+(r2*sin(alpha2));
KP(1,16)=sqrt(Rb^2-(Rb-KP(2,16))^2);
```

```
function [z]=CheckLimitPelvisNew(zIn,highX,lowX,n,m)
z=zIn;
for i=1:m
    for j=1:n % check if the implicite values respects the bounderie-limit
         if (zIn(i,j))>highX(i)
             z(i,j)=highX(i)*0.99999;
         end
         if zIn(i,j)<lowX(i)
             z(i,j)=lowX(i)+rand(1);
         end
    end
end


for j=1:n
     if (z(1,j)>=z(3,j));% check if the explicite values respects the bounderie-limit
        z(3,j)=z(1,j)+0.2+rand(1);
     end
     if (z(2,j)>=z(4,j));% check if the explicite values respects the bounderie-limit
         z(4,j)=z(2,j)+0.2+rand(1);
     end

end
```

```
function [Obj]=ObjectFunction(Vol,ObjStress,StressMax,ObjU,MaxU,VolGear)

if StressMax<=ObjStress % Penalty; if the stress exceed the limit
    PenStress=0;
else
    PenStress=(exp((StressMax-ObjStress)*10));
end

if MaxU<=ObjU % Penalty; if the displacement exceed the limit
    PenDisplacement=0;
else
    PenDisplacement=(exp(1000*(MaxU-ObjU)));
end

Obj=(Vol-VolGear)+PenStress+PenDisplacement;
```

```
function [Vol,StressMax,Umax]=ReadPelvisResult(i)
% Read results - von Mises stress at each element

    outputfile = 'D:\Ansys\PelvisStress.1' ;
    fid = fopen(outputfile,'r');
    p=0; pp=0;
    %Read one line at the time
    while (feof(fid)~=1)
        line=fgetl(fid);
        if strcmp(line,' MAXIMUM VALUES')
           p=p+1;
        end

        if p>0
            pp=pp+1;
        end

        if pp==3
            line;
            line(1:6)=[];
        numline = str2num(line);
        StressMax=numline; % [MPa]
        end
     end
    fclose(fid);

  % Volume of each element
      outputfile = 'D:\Ansys\PelvisVolu.1'  ;
     fid = fopen(outputfile,'r');
     p=0; pp=0;
    %Read one line at the time
    while (feof(fid)~=1)
        line=fgetl(fid);
if length(line)>21
    p=p+1;
        line(1:20)=[];
        Numline=str2num(line);
        if p==5
            Vol=Numline;
        end
end

     end
    fclose(fid);

  outputfile = 'D:\Ansys\PelvisDisp.1';
  fid = fopen(outputfile,'r');
  p=0; pp=0;

    %Read one line at the time
    while (feof(fid)~=1)
        disp=fgetl(fid);

        if strcmp(disp,' MAXIMUM ABSOLUTE VALUES')
           p=p+1;
        end
```

```
        if p>0
            pp=pp+1;
        end

        if pp==3
            disp;
            disp(1:6)=[];
        numline = str2num(disp);
        Umax=numline(4); % [mm]
        end
     end
    fclose(fid);
```

```
@ echo off
C:
cd C:\Program Files\Ansys Inc\v100\ANSYS\bin\intel\
set ANSYS_LOCK=OFF
ansys100 -b -p ansysrf -i D:\Ansys\Pelvis.lgw -o D:\Ansys\Pelvis.out
```

```
function [zr,z_c,PlaceV,PlaceB]=WorstPointAnsys(F,z0,n,m)

[x_worst,PlaceV]=max((F));
[x_best,PlaceB]=min((F));
alpha=1.4;
for i=1:m
z_sum=sum(z0(i,:)); % sum of the point( without the worst point)
z_c(i)=1/(n-1)*(z_sum-z0(i,PlaceV));% Centroid
end
zr=z_c'+alpha*(z_c'-z0(:,PlaceV)); %Mirrored point, new value for worst point
```

```
function [zr,PlaceV,PlaceB]=WorstPointAnsys_new(F,z0,n,m,k)

[x_worst,PlaceV]=max((F));
[x_best,PlaceB]=min((F));
if k==5
    alpha=0.5;
else
    alpha=0.25;
end
for i=1:m
z_sum=sum(z0(i,:)); % sum of the point( without the worst point)
z_c(i)=1/(n-1)*(z_sum-z0(i,PlaceV));% Centroid
end
zr=z_c'+alpha*(z_c'-z0(:,PlaceV)); %Mirrored point, new value for worst point
```

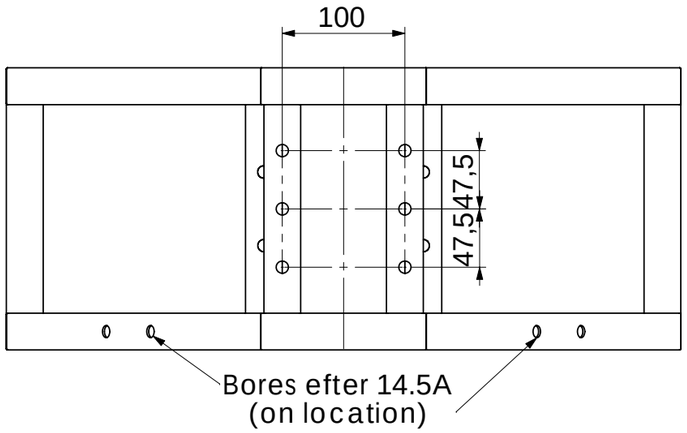
```
function []=WritePelvisFile(x0)
%x0=[10 15 14 17.5];
%x0=[2.03 6.8 23.99 8.14];
KP=CalcKeyPoints(x0);
 r0=x0(1);% Radius 1
 r1=x0(1); % Radius 1
 r2=x0(2); % Radius 2
 r3=x0(2);% radius 2

% r0=2.01;
% r1=2.01;
% r2=x0(2);
% r3=x0(1);
 h=10;%x0(5);
%Generate filename
    filename ='D:\Ansys\Pelvis.lgw'  ;
      %Open file
    fid = fopen(filename,'w');

% Batch mode
fprintf(fid,'/BATCH  \n');
fprintf(fid,'/COM,ANSYS RELEASE 10.0A1  UP20060105\n');
fprintf(fid,'/input,menust,tmp,'',,,,,,,,,,,,,,1   \n');
fprintf(fid,'/CWD,''D:\\Ansys'' \n');% Working directory
fprintf(fid,'/CPLANE,1');
fprintf(fid,'WPSTYLE,,,,,,,,0\n');
fprintf(fid,'/PREP7\n')  ;
fprintf(fid,'K,1,,,,,\n');  % Key points (KP)
fprintf(fid,'K,2,140,0,,,\n');
fprintf(fid,'K,3,190,50,,,\n');
fprintf(fid,'K,4,184,50,,,\n');
fprintf(fid,'K,5,140,50,,,\n');
fprintf(fid,'K,6,96,50,,,\n');
fprintf(fid,'K,7,44,50,,,\n');
fprintf(fid,'K,8,0,50,,,\n');
fprintf(fid,'K,9,0,6,,,\n');
fprintf(fid,'K,10,%6.3f,%6.3f,,  \n',KP(1,10),KP(2,10));
fprintf(fid,'K,11,%6.3f,%6.3f,,  \n',KP(1,11),KP(2,11));
fprintf(fid,'K,12,%6.3f,%6.3f,,  \n',KP(1,12),KP(2,12));
fprintf(fid,'K,13,%6.3f,%6.3f,,  \n',KP(1,13),KP(2,13));
fprintf(fid,'K,14,%6.3f,%6.3f,,  \n',KP(1,14),KP(2,14));
fprintf(fid,'K,15,%6.3f,%6.3f,,  \n',KP(1,15),KP(2,15));
fprintf(fid,'K,16,%6.3f,%6.3f,,  \n',KP(1,16),KP(2,16));
fprintf(fid,'K,17,%6.3f,%6.3f,,  \n',KP(1,17),KP(2,17));
fprintf(fid,'K,18,%6.3f,%6.3f,,  \n',KP(1,18),KP(2,18));
fprintf(fid,'K,19,%6.3f,%6.3f,,  \n',KP(1,19),KP(2,19));
fprintf(fid,'K,20,%6.3f,%6.3f,,  \n',KP(1,20),KP(2,20));
fprintf(fid,'K,21,%6.3f,%6.3f,,  \n',KP(1,21),KP(2,21));
fprintf(fid,'K,22,140,6,,,\n');
fprintf(fid,'K,23,70,0,,,\n');
fprintf(fid,'K,24,70,%6.3f,,,\n',KP(2,10));
fprintf(fid,'K,25,70,%6.3f,,,\n',KP(2,14));
fprintf(fid,'K,26,70,50,,,\n');
fprintf(fid,'LSTR,1,23\n'); % Line between KPs
fprintf(fid,'LSTR,23,2\n');
fprintf(fid,'LSTR,3,4\n');
```

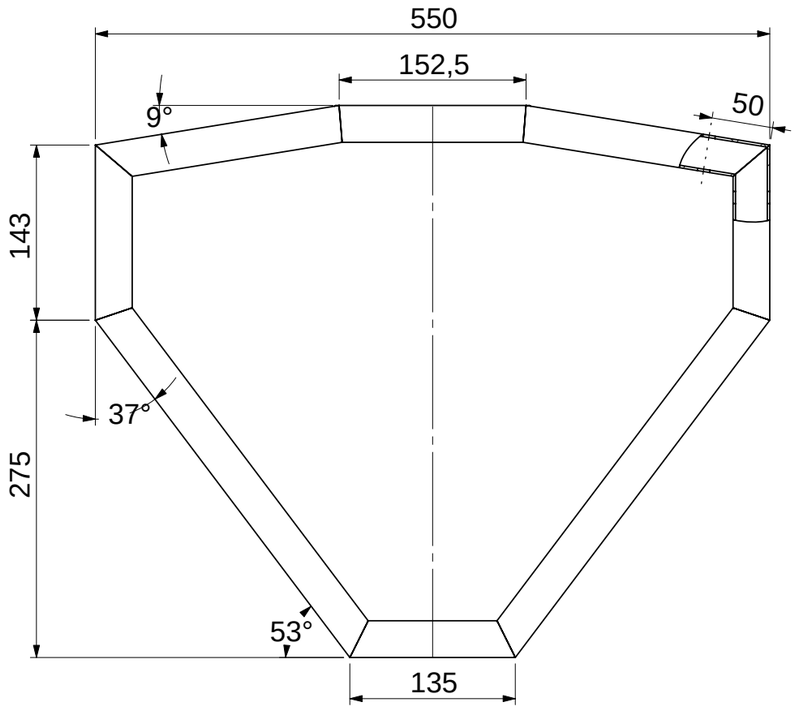
```
fprintf(fid,'LSTR,4,5\n');
fprintf(fid,'LSTR,5,6\n');
fprintf(fid,'LSTR,6,26\n');
fprintf(fid,'LSTR,26,7\n');
fprintf(fid,'LSTR,7,8\n');
fprintf(fid,'LSTR,8,9\n');
fprintf(fid,'LSTR,9,1\n');
fprintf(fid,'LSTR,10,24\n');
fprintf(fid,'LSTR,24,11\n');
fprintf(fid,'LSTR,14,25\n');
fprintf(fid,'LSTR,25,15\n');
fprintf(fid,'LSTR,24,23\n');
fprintf(fid,'LSTR,25,26\n');
fprintf(fid,'LSTR,5,22\n');
fprintf(fid,'LSTR,22,2\n');
fprintf(fid,'LARC,2,3,5,50\n'); % Arc between two KP with center at a 3th KP and a arc-↙
lenght
fprintf(fid,'LARC,7,9,8,44\n');
fprintf(fid,'LARC,10,17,18,%6.2f \n',r0); % set arc length
fprintf(fid,'LARC,11,12,19,%6.2f,,  \n',r1);
fprintf(fid,'LARC,12,13,5,50\n');
fprintf(fid,'LARC,13,14,20,%6.2f,,  \n',r3);%r2
fprintf(fid,'LARC,15,16,21,%6.2f,,  \n',r2);
fprintf(fid,'LARC,16,17,8,50\n');
fprintf(fid,'FLST,2,2,8 \n');
fprintf(fid,'FITEM,2,140,50,0\n'); % half circle
fprintf(fid,'FITEM,2,96,50,0  \n');
fprintf(fid,'CIRCLE,P51X, , , ,180, ,\n');
fprintf(fid,'NUMMRG,KP,0.5, , ,LOW \n'); % Merge KP in the range of 0.05
fprintf(fid,'/AUTO,1 \n');

fprintf(fid,'FLST,2,3,4  \n'); % pick lines for area 1
fprintf(fid,'FITEM,2,9   \n');
fprintf(fid,'FITEM,2,20  \n');
fprintf(fid,'FITEM,2,8  \n');
fprintf(fid,'AL,P51X \n');

fprintf(fid,'FLST,2,3,4  \n'); % pick lines for area 2
fprintf(fid,'FITEM,2,5   \n');
fprintf(fid,'FITEM,2,27  \n');
fprintf(fid,'FITEM,2,17  \n');
fprintf(fid,'AL,P51X \n');

fprintf(fid,'FLST,2,3,4  \n'); % pick lines for area 3
fprintf(fid,'FITEM,2,4   \n');
fprintf(fid,'FITEM,2,17  \n');
fprintf(fid,'FITEM,2,28  \n');
fprintf(fid,'AL,P51X \n');

fprintf(fid,'FLST,2,11,4  \n');% pick lines for area 4
fprintf(fid,'FITEM,2,10   \n');
fprintf(fid,'FITEM,2,1   \n');
fprintf(fid,'FITEM,2,15  \n');
fprintf(fid,'FITEM,2,11  \n');
fprintf(fid,'FITEM,2,21  \n');
fprintf(fid,'FITEM,2,26  \n');
```

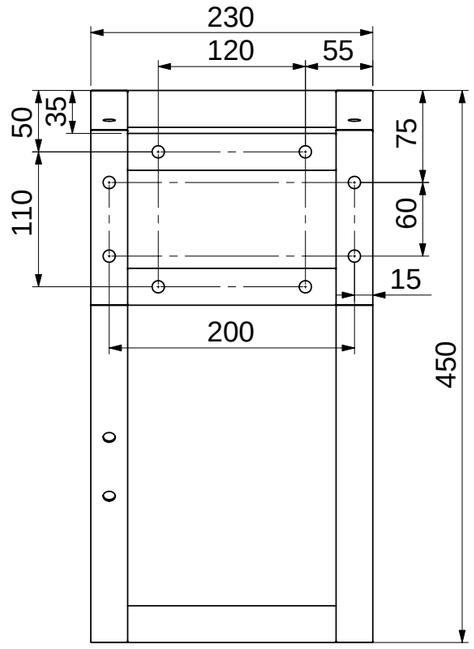
```
fprintf(fid,'FITEM,2,25  \n');
fprintf(fid,'FITEM,2,14  \n');
fprintf(fid,'FITEM,2,16  \n');
fprintf(fid,'FITEM,2,7  \n');
fprintf(fid,'FITEM,2,20  \n');
fprintf(fid,'AL,P51X \n');

fprintf(fid,'FLST,2,11,4 \n');% pick lines for area 5
fprintf(fid,'FITEM,2,15  \n');
fprintf(fid,'FITEM,2,12  \n');
fprintf(fid,'FITEM,2,2  \n');
fprintf(fid,'FITEM,2,18   \n');
fprintf(fid,'FITEM,2,27  \n');
fprintf(fid,'FITEM,2,6  \n');
fprintf(fid,'FITEM,2,16   \n');
fprintf(fid,'FITEM,2,13  \n');
fprintf(fid,'FITEM,2,24  \n');
fprintf(fid,'FITEM,2,23  \n');
fprintf(fid,'FITEM,2,22  \n');
fprintf(fid,'AL,P51X \n');

fprintf(fid,'FLST,2,4,4  \n'); % pick lines for area 6
fprintf(fid,'FITEM,2,18   \n');
fprintf(fid,'FITEM,2,19  \n');
fprintf(fid,'FITEM,2,3  \n');
fprintf(fid,'FITEM,2,28  \n');
fprintf(fid,'AL,P51X \n');

fprintf(fid,'VOFFST,1,%6.2f, ,  \n',h); % extrude area
fprintf(fid,'VOFFST,1,%6.2f, ,  \n',-h);
fprintf(fid,'VOFFST,2,%6.2f, ,  \n',h);
fprintf(fid,'VOFFST,2,%6.2f, ,  \n',-h);
fprintf(fid,'VOFFST,3,%6.2f, ,  \n',h);
fprintf(fid,'VOFFST,3,%6.2f, ,  \n',-h);
fprintf(fid,'VOFFST,4,%6.2f, ,  \n',h);
fprintf(fid,'VOFFST,4,%6.2f, ,  \n',-h);
fprintf(fid,'VOFFST,5,%6.2f, , \n',h);
fprintf(fid,'VOFFST,5,%6.2f, , \n',-h);
fprintf(fid,'VOFFST,6,%6.2f, , \n',h);
fprintf(fid,'VOFFST,6,%6.2f, , \n',-h);
fprintf(fid,'ET,1,SOLID92\n'); % Element, solid element
fprintf(fid,'MPTEMP,,,,,,,,\n');
fprintf(fid,'MPTEMP,1,0  \n');
fprintf(fid,'MPDATA,EX,1,,70000  \n');% Elastic Modulus
fprintf(fid,'MPDATA,PRXY,1,,0.33\n'); % Poisson ratio
fprintf(fid,'MPTEMP,,,,,,,,\n');
fprintf(fid,'MPTEMP,1,0  \n');
fprintf(fid,'MPDATA,EX,2,,2100000  \n');% Elastic Modulus
fprintf(fid,'MPDATA,PRXY,2,,0.3\n'); % Poisson ratio
fprintf(fid,'FLST,5,6,6,ORDE,2 \n');
fprintf(fid,'FITEM,5,7   \n');
fprintf(fid,'FITEM,5,-12 \n');
fprintf(fid,'CM,_Y,VOLU  \n');
fprintf(fid,'VSEL, , , ,P51X \n');
fprintf(fid,'CM,_Y1,VOLU \n');
fprintf(fid,'CMSEL,S,_Y  \n');
```

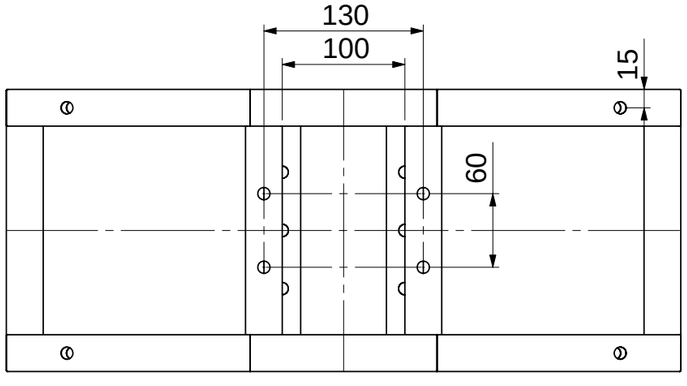
```
fprintf(fid,'CMSEL,S,_Y1 \n');
fprintf(fid,'VATT,       1, ,   1,       0 \n');
fprintf(fid,'CMSEL,S,_Y  \n');
fprintf(fid,'CMDELE,_Y   \n');
fprintf(fid,'CMDELE,_Y1  \n');
fprintf(fid,'FLST,5,6,6,ORDE,2 \n');
fprintf(fid,'FITEM,5,1   \n');
fprintf(fid,'FITEM,5,-6  \n');
fprintf(fid,'CM,_Y,VOLU  \n');
fprintf(fid,'VSEL, , , ,P51X \n');
fprintf(fid,'CM,_Y1,VOLU \n');
fprintf(fid,'CMSEL,S,_Y   \n');
fprintf(fid,'CMSEL,S,_Y1 \n');
fprintf(fid,'VATT,       2, ,   1,       0 \n');
fprintf(fid,'CMSEL,S,_Y  \n');
fprintf(fid,'CMDELE,_Y   \n');
fprintf(fid,'CMDELE,_Y1  \n');
fprintf(fid,'MSHAPE,1,3D \n'); % Mesh areas
fprintf(fid,'MSHKEY,0\n');
fprintf(fid,'FLST,5,12,6,ORDE,2\n');
fprintf(fid,'FITEM,5,1   \n');
fprintf(fid,'FITEM,5,-12  \n');
fprintf(fid,'CM,_Y,VOLU  \n');
fprintf(fid,'VSEL, , , ,P51X \n');
fprintf(fid,'CM,_Y1,VOLU \n');
fprintf(fid,'CHKMSH,''VOLU''   \n');
fprintf(fid,'CMSEL,S,_Y  \n');
fprintf(fid,'VMESH,_Y1   \n');
fprintf(fid,'CMDELE,_Y   \n');
fprintf(fid,'CMDELE,_Y1  \n');
fprintf(fid,'CMDELE,_Y2  \n');
fprintf(fid,'/UI,MESH,OFF\n');
fprintf(fid,'NUMMRG,NODE,0.1, , ,LOW   \n');
fprintf(fid,'DA,1,ALL,\n'); % displacement, DOF all, area 9
fprintf(fid,'FK,5,FZ,1518\n'); %force 1/4force
fprintf(fid,'FK,4,FZ,-767\n');
fprintf(fid,'Fk,6,Fz,767\n');
fprintf(fid,'FK,22,FZ,-1023\n');
fprintf(fid,'/SOL\n');% solve
fprintf(fid,'/STATUS,SOLU\n');
fprintf(fid,'SOLVE  \n');
fprintf(fid,'/POST1\n');
fprintf(fid,'Pelvis.macro \n');

% Close file
fclose(fid);

%Generate filename as string
    filename ='D:\Ansys\Pelvis.macro' ;

fid = fopen(filename,'w');

for i=1:1
    fprintf(fid,'SET,%i\n',i);
    fprintf(fid,'/output,''D:\\ANSYS\\PelvisVolu.%i''\n',i);
    fprintf(fid,'PRESOL,VOLU\n'); % Volume of each element
```

```
    fprintf(fid,'ETABLE,SEQV,S,EQV\n');
    fprintf(fid,'/output,''D:\\ANSYS\\PelvisStress.%i''\n',i);
    fprintf(fid,'PRETAB,SEQV\n'); % Element solution, von Mises stress
    fprintf(fid,'/output,''D:\\ANSYS\\PelvisDisp.%i''\n',i);
    fprintf(fid,'/FORMAT,,,20\n');
    fprintf(fid,'PRNSOL,U,COMP\n'); % Displacment of nodes
    fprintf(fid,'/PREP7\n')  ;
    fprintf(fid,'/output,''D:\\ANSYS\\PelvisVolu.%i''\n',i);
    fprintf(fid,'VSUM\n'); % Volume of each element

    fprintf(fid,'/out\n');
    fprintf(fid,'\n');
end
fclose(fid);
```

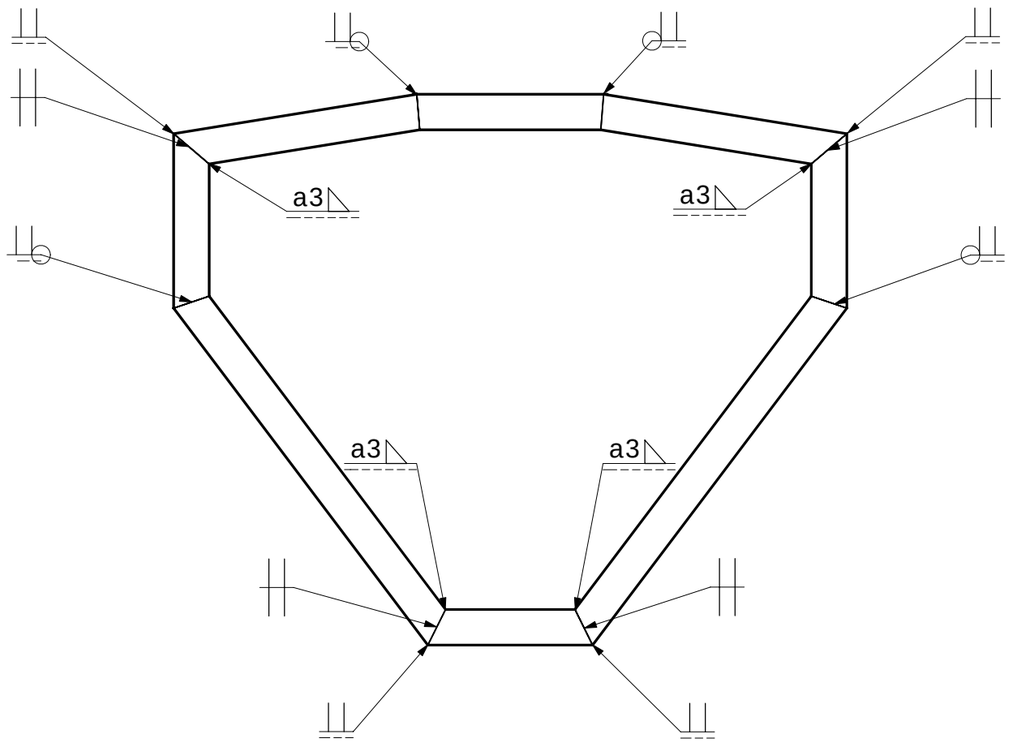


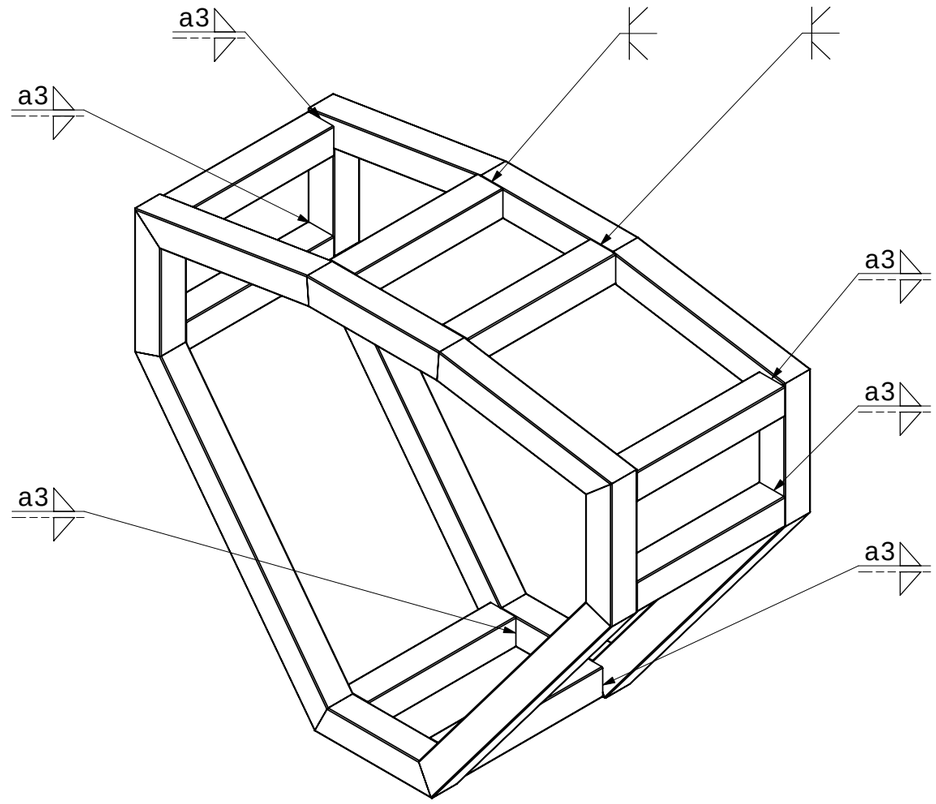






NOTE: For ikke tolerence specificeret mål benyttes: DS/EN ISO 13920 A Samtlige huller er Ø10mm. Efterfølgende svejses bøsninger (14.6) i alle huller således de ligger glat med overfladen af rammen. Svejsningerne planslibes efterfølgende.	MATERIAL: Al (Svejsbart) NUMBER: 1 stk. SCALE: 1:5	14.1B Torso Aalborg Universitet DMS10 Grp.43A, Pon 103 DWG NO. 14_1B_Torso — A3 SCALE:1:5 — SHEET 1 OF 1





NOTE:
De to rammer svejses først, hvorefter afstandsstykkerne svejses.
Svejsninger på anlægsflader / samlingsflader planslibes.

MATERIAL:
Al (Svejsbart)

NUMBER:
1 stk.

SCALE:
1:5

14.1C Torso

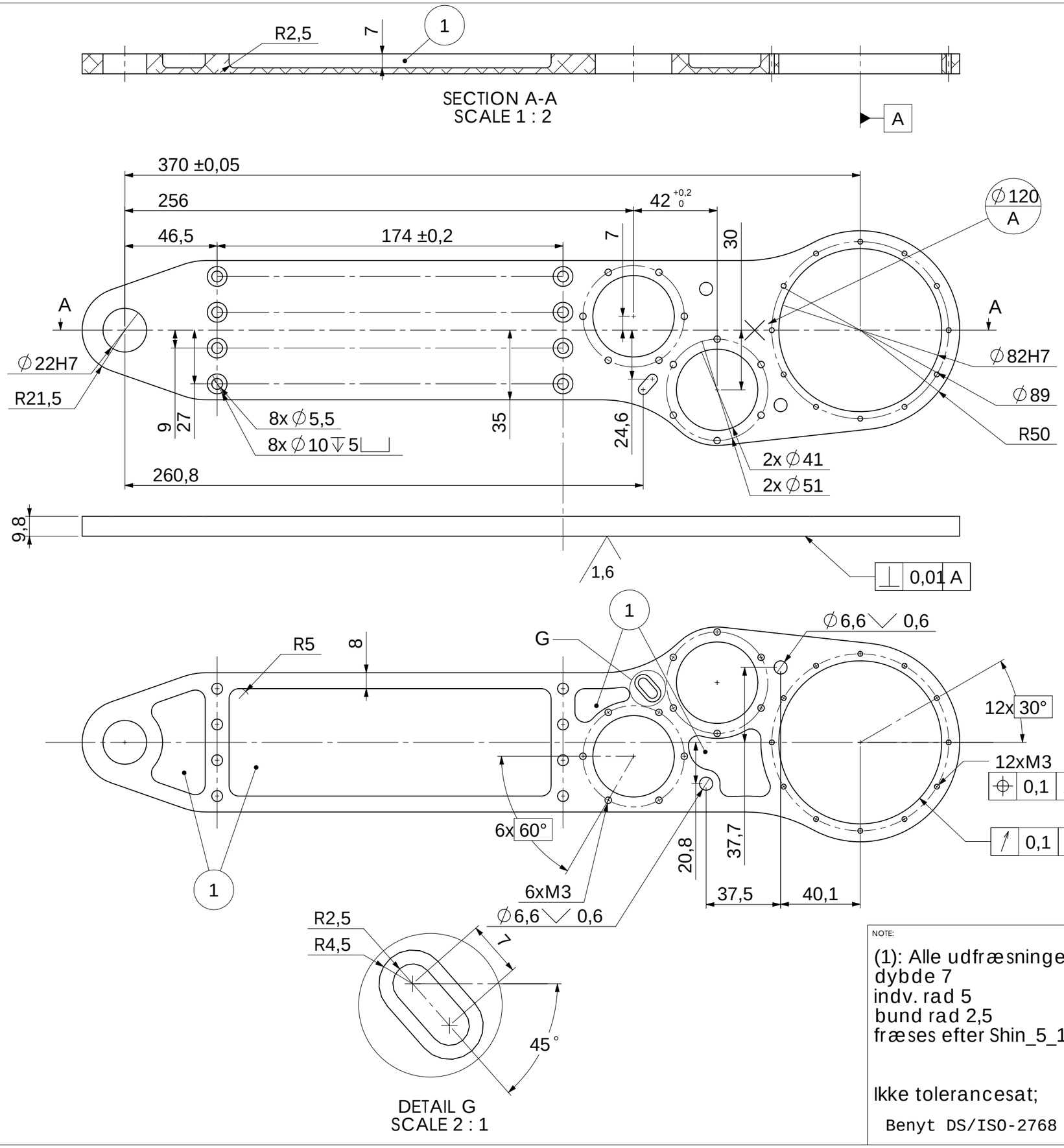
Aalborg Universitet
DMS10
Grp.43A, Pon 103

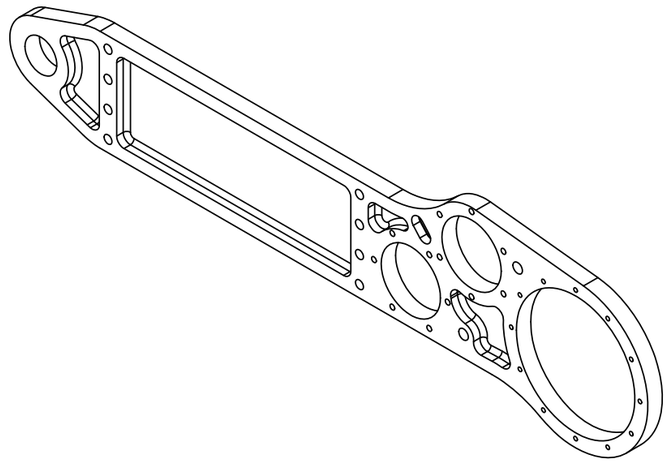
DWG NO. 14_1C_Torso

A3

SCALE:1:5

SHEET 1 OF 1





NOTE:
(1): Alle udfræsninger:
dybde 7
indv. rad 5
bund rad 2,5
fræses efter Shin_5_1a.iges

Ikke tolerancesat;
Benyt DS/ISO-2768

MATERIAL:
AW-6082 T6

NUMBER:
1 stk.

SCALE:
1:2

5.1a Shin

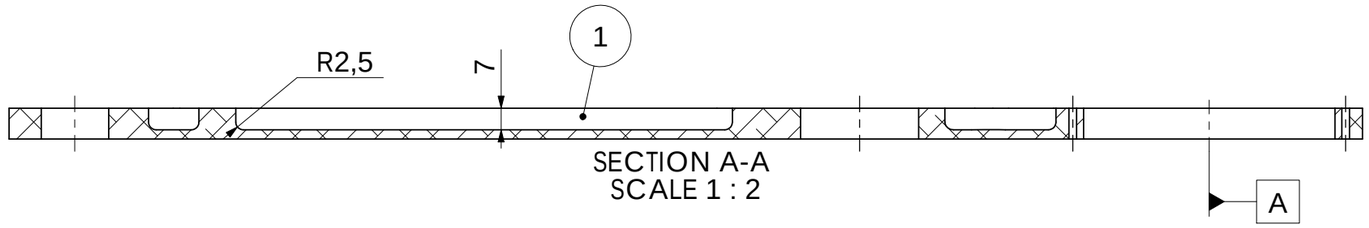
Aalborg Universitet
DMS10
Grp.43A, Pon 103

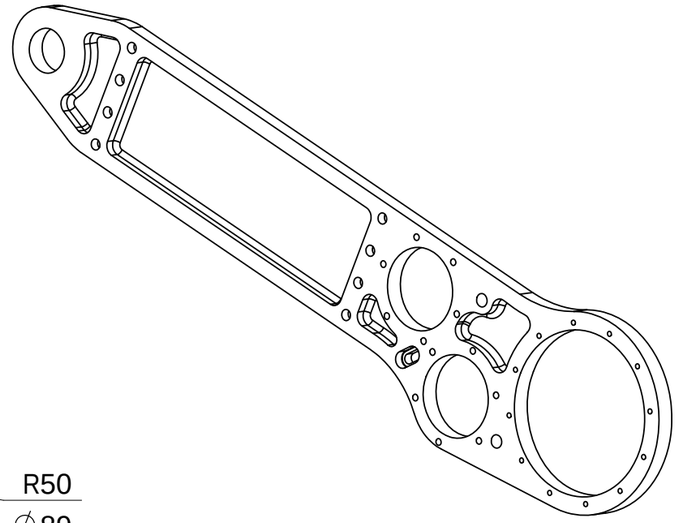
DWG NO.
Shin

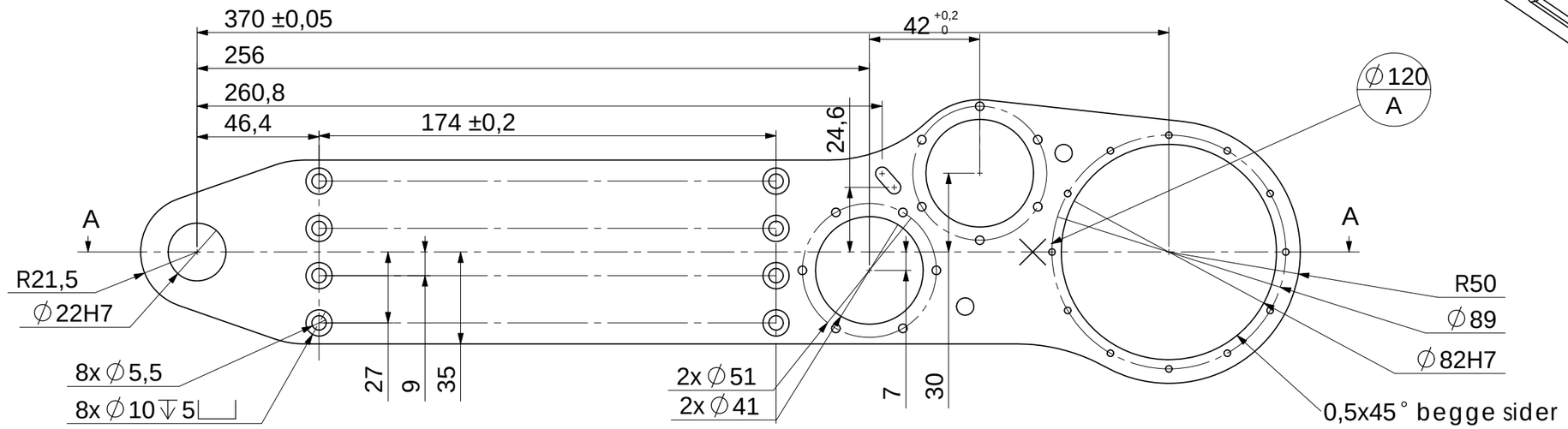
A3

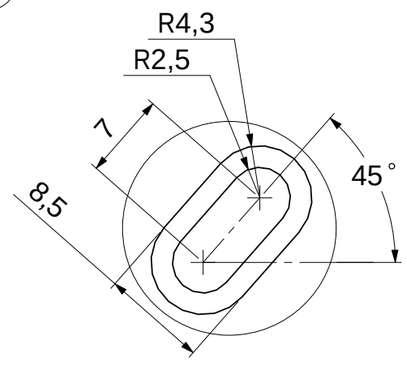
SCALE:1:5

SHEET 1 OF 6









DETAIL E
SCALE 2 : 1

NOTE:
(1): Alle udfræsninger:
dybde 7
indv. rad 5
bund rad 2,5
fræses efter Shin_5_1b.iges

Ikke tolerancesat;
benyt DS/ISO-2768

MATERIAL:
AW-6082 T6

NUMBER:
1 stk.

SCALE:
1:2

5.1b Shin

Aalborg Universitet
DMS10
Grp.43A, Pon 103

DWG NO. Shin

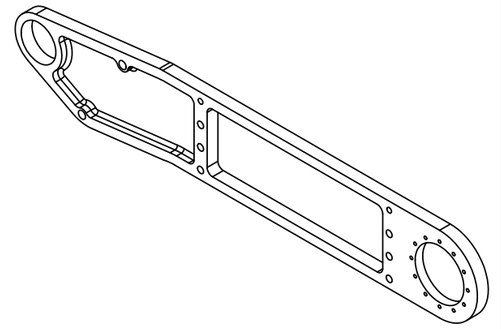
A3

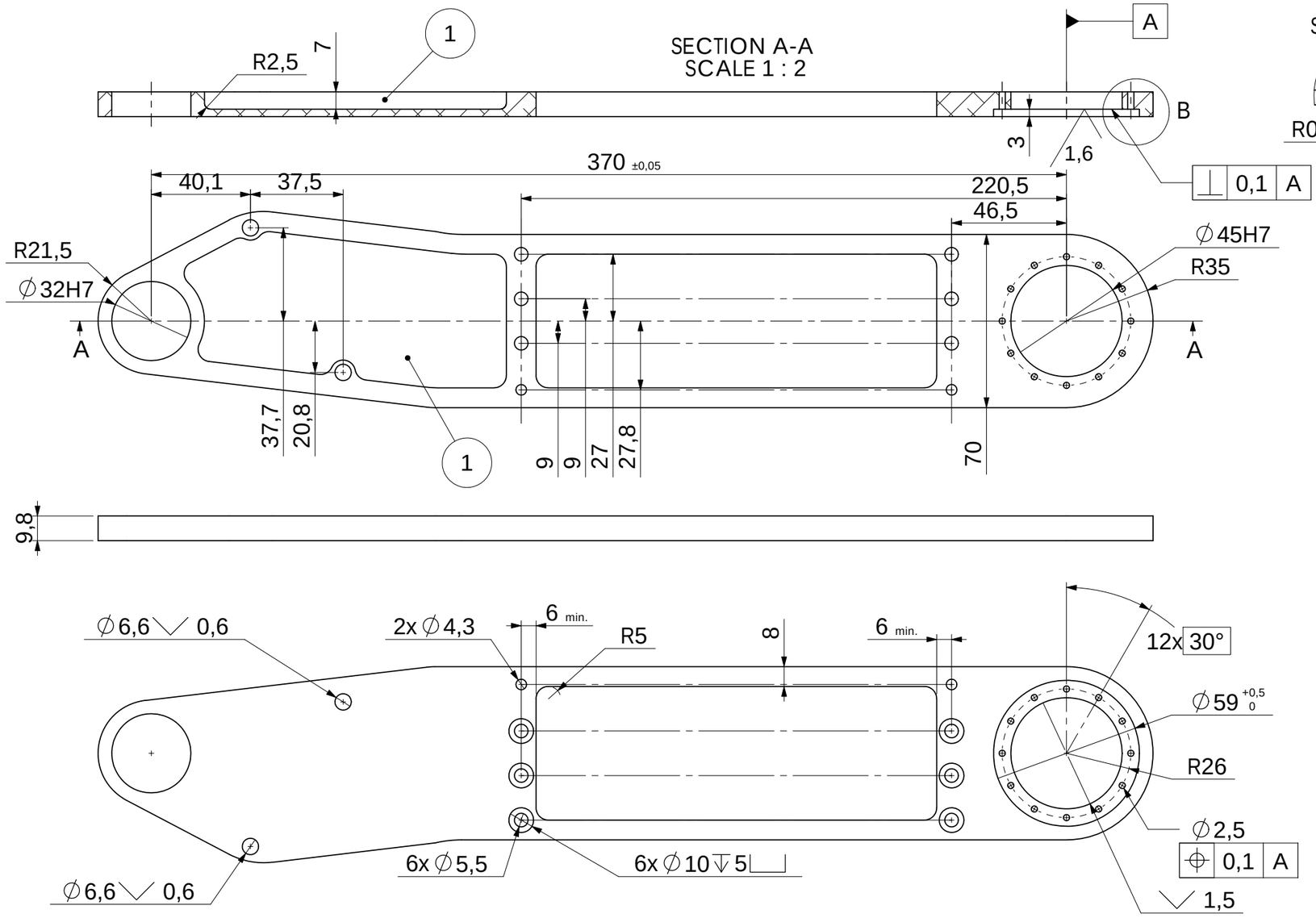
SCALE:1:1

SHEET 2 OF 6

SECTION A-A
SCALE 1 : 2

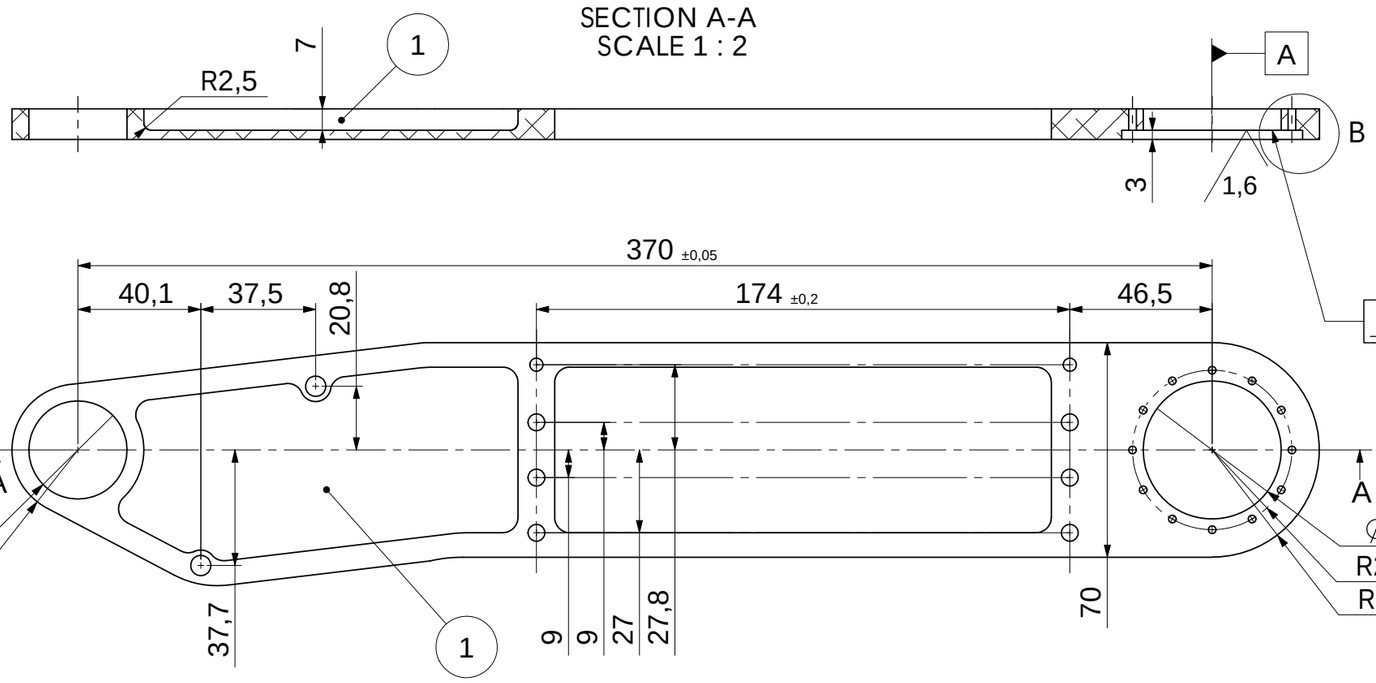
DETAIL B
SCALE 1 : 1



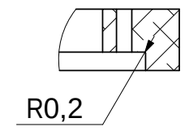


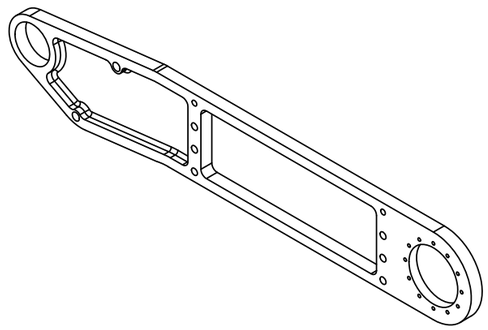
NOTE: Udfræsningen (1): indv. rad. 5 bund rad. 6 fræses efter Shin_5_2a.iges Ikke tolerancesat: Benyt DS/ISO-2768	MATERIAL: AW-6082 T6 NUMBER: 1 stk. SCALE: 1:2	5.2a Shin Aalborg Universitet DMS10 Grp.43A, Pon 103 DWG NO. Shin A3 SCALE:1:5 SHEET 3 OF 6

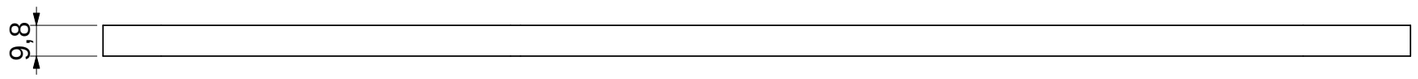
SECTION A-A
SCALE 1 : 2

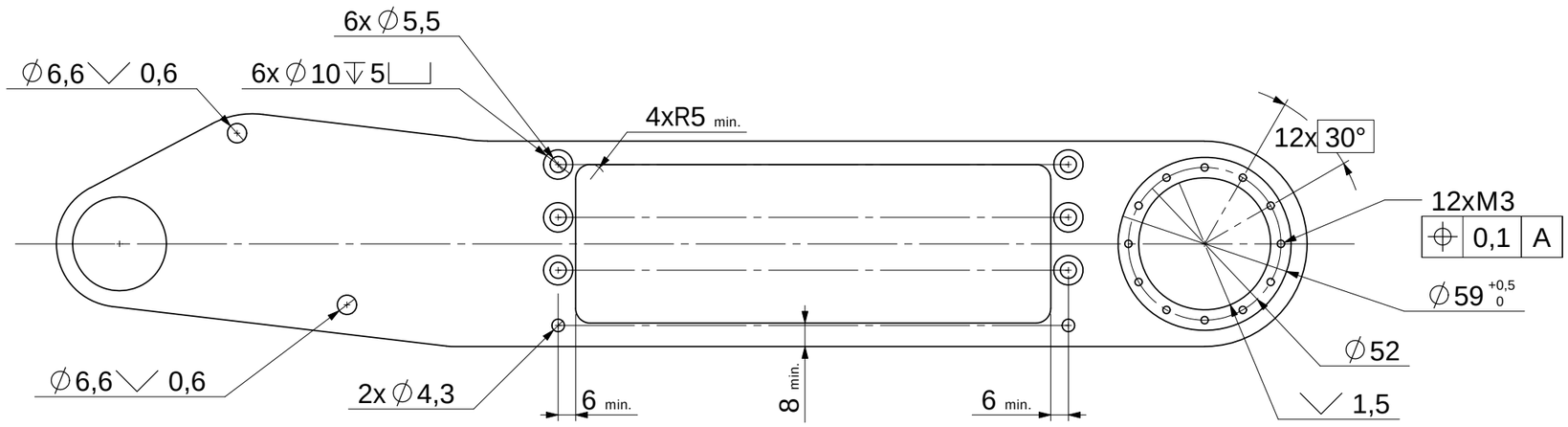


DETAIL B
SCALE 1 : 1









NOTE:
Udfræsningen (1):
indv. rad. 5
bund rad. 2,5
fræses efter Shin_5_2b.iges

Ikke tolerancesat:
Benyt DS/ISO -2768

MATERIAL:
AW-6082 T6

NUMBER:
1 stk.

SCALE:
1:2

5.2b Shin

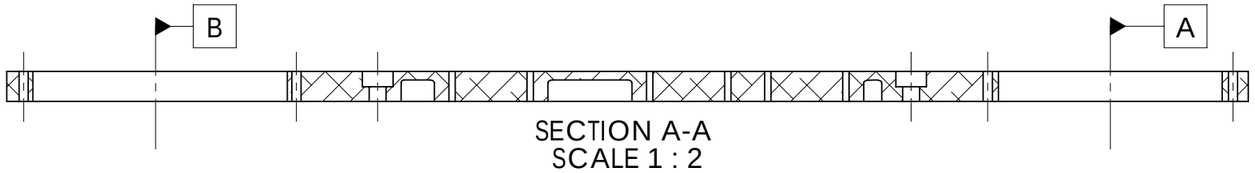
Aalborg Universitet
DMS10
Grp.43A, Pon 103

DWG NO.
Shin

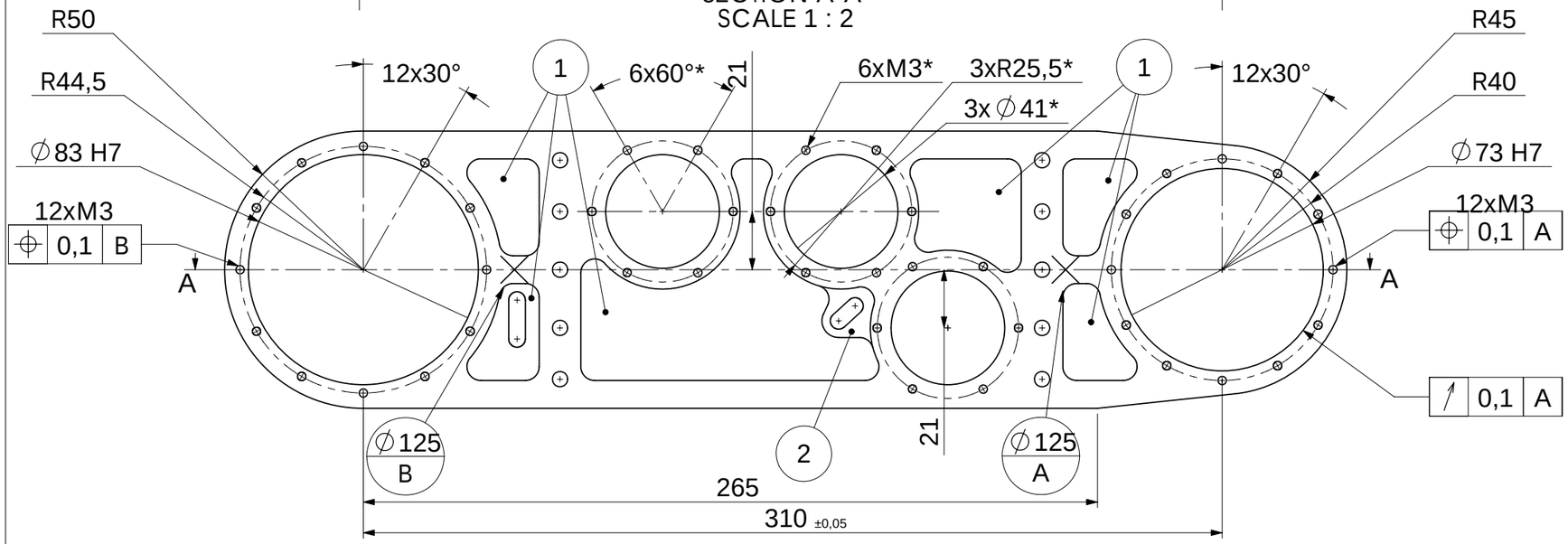
A3

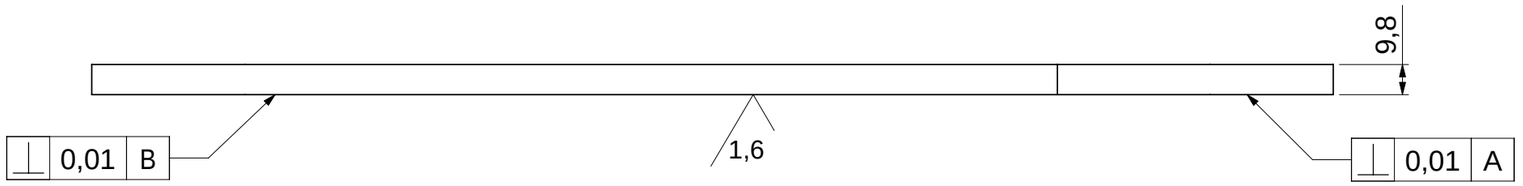
SCALE:1:5

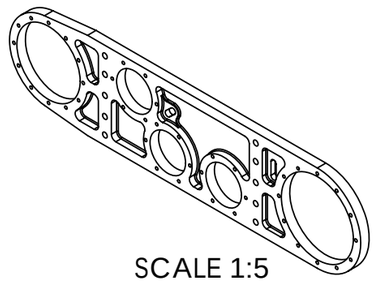
SHEET 4 OF 6



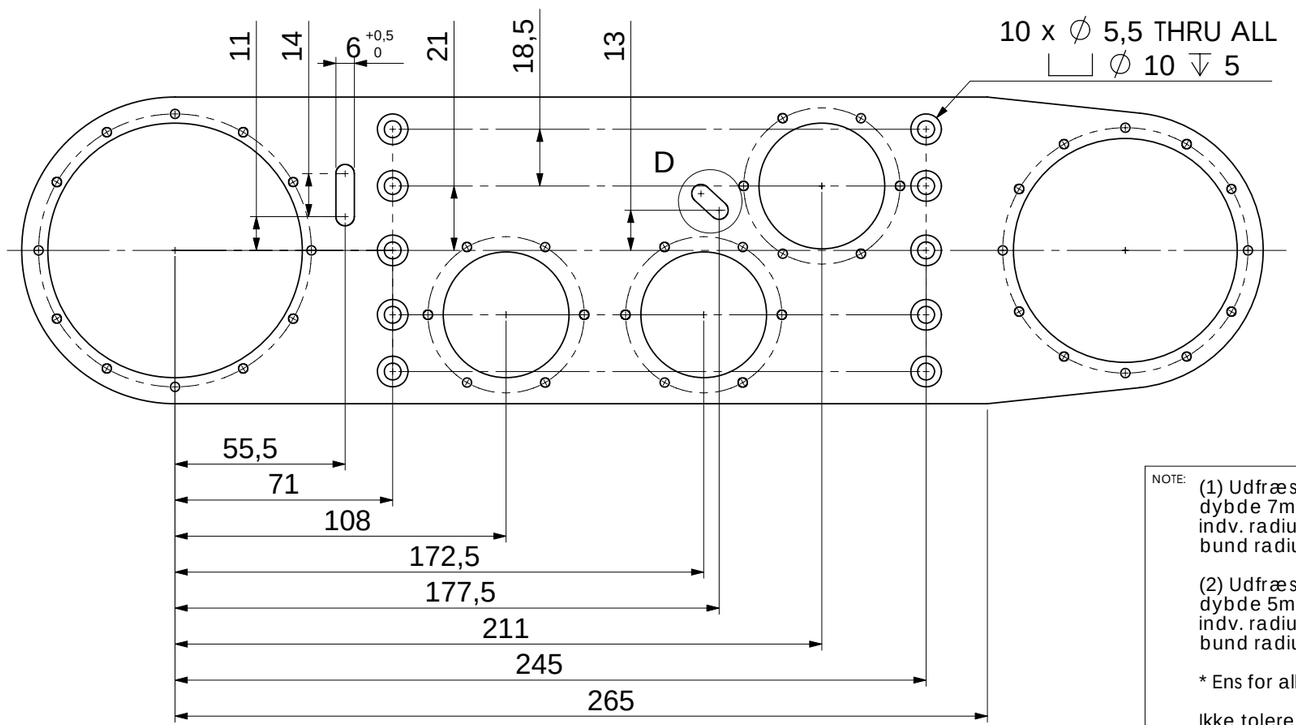
SECTION A-A
SCALE 1 : 2

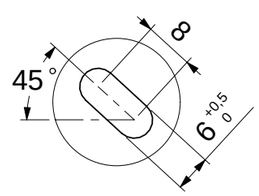






SCALE 1:5





DETAIL D
SCALE 1 : 1

NOTE:
(1) Udfræsninger:
dybde 7mm
indv. radius 5mm
bund radius 2,5mm

(2) Udfræsning
dybde 5mm
indv. radius 5mm
bund radius 2,5mm

* Ens for alle 3 huller

Ikke tolerencesat; benyt DS/ISO 2768-m

fræses efter Thigh_6_1a.iges

MATERIAL: AW-6082 T6

NUMBER: 1 stk.

SCALE: 1:2

6.1a Thigh

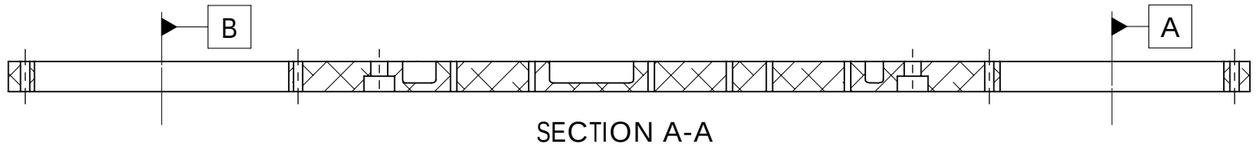
Aalborg Universitet
DMS10
Grp.43A, Pon 103

DWG NO. 6_1_a_Thigh

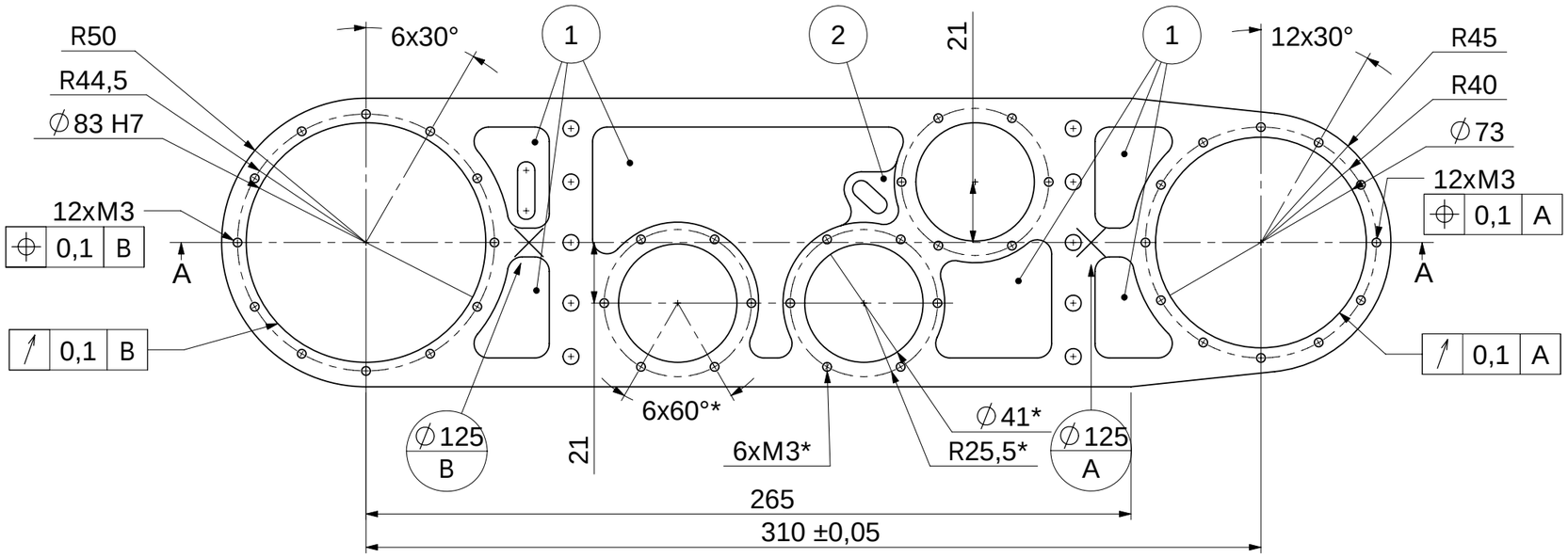
A3

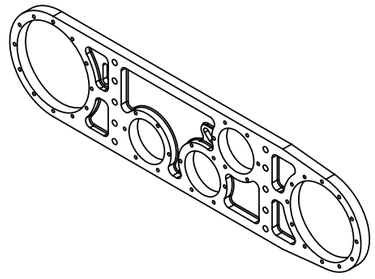
SCALE:1:5

SHEET 1 OF 1

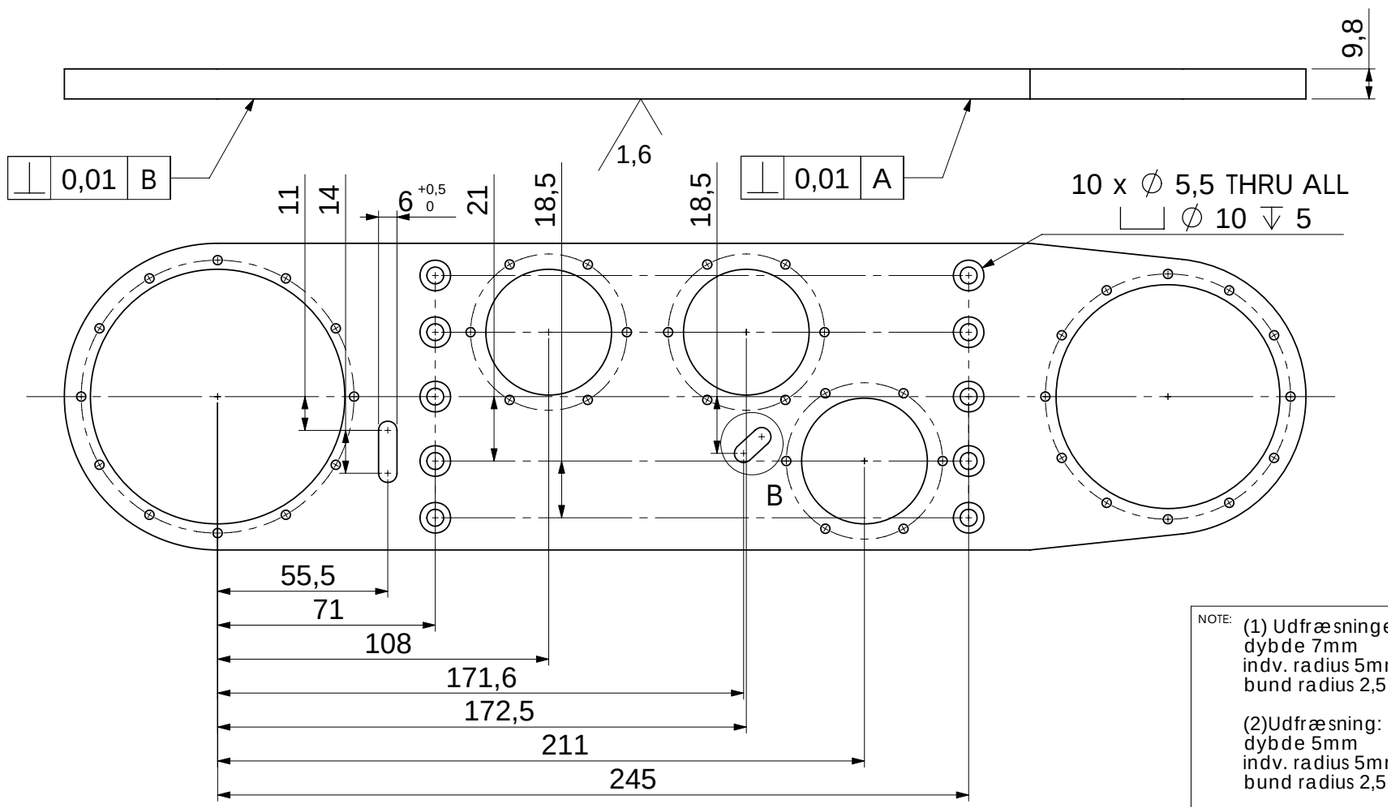


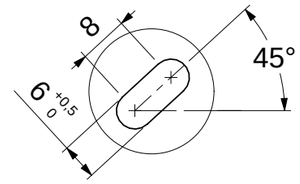
SECTION A-A
SCALE 1 : 2





Scale 1:5





DETAIL B
SCALE 1 : 1

NOTE:
(1) Udfræsninger:
dybde 7mm
indv. radius 5mm
bund radius 2,5mm

(2)Udfræsning:
dybde 5mm
indv. radius 5mm
bund radius 2,5mm

* Ens for alle 3 huller

Ikke tolerencesat; benyt DS/ISO 2768-m

Fræses efter Thigh_6_1_b.iges

MATERIAL: AW-6082 T6

NUMBER: 1 stk.

SCALE: 1:2

6.1b Thigh

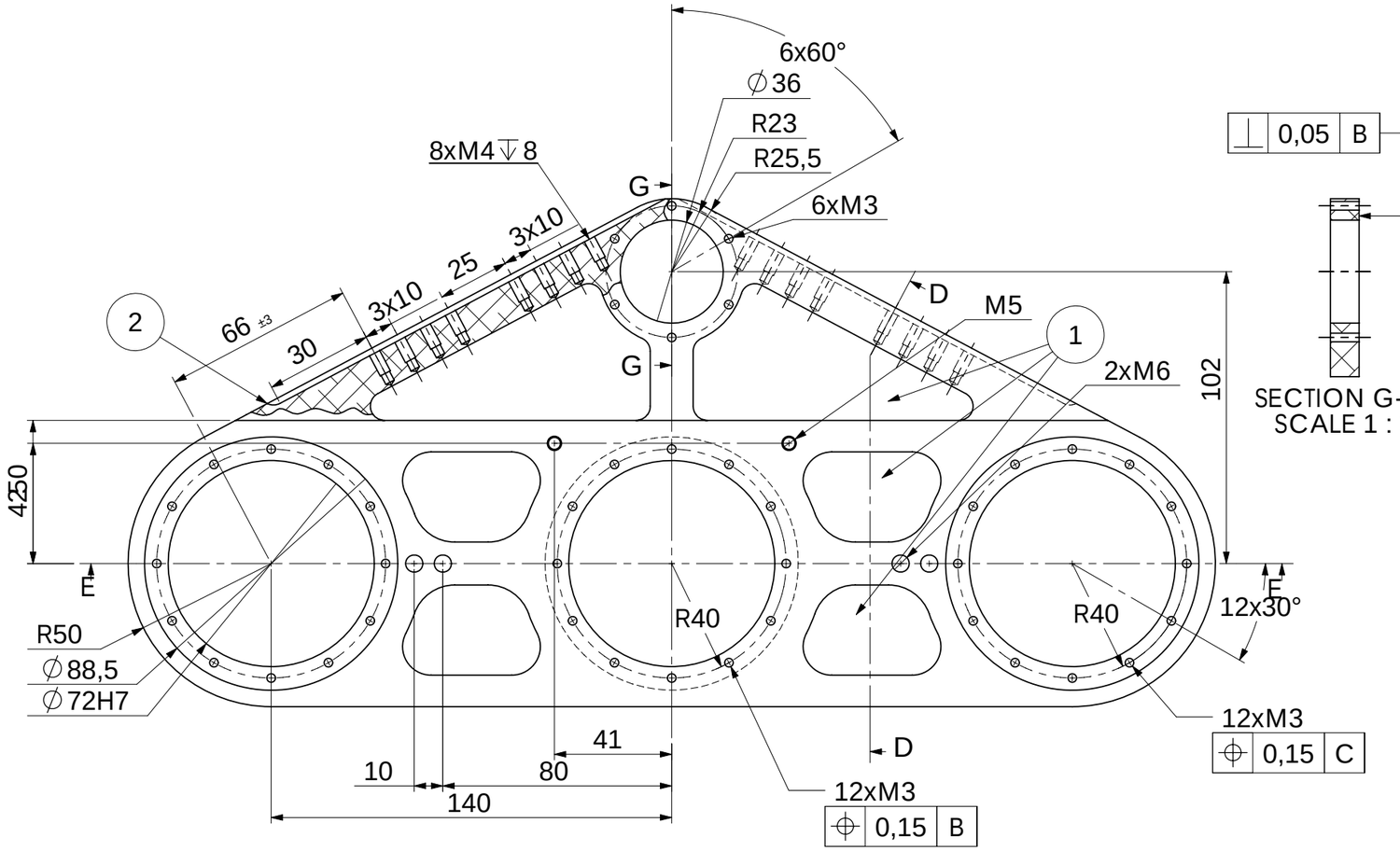
Aalborg Universitet
DMS10
Grp.43A, Pon 103

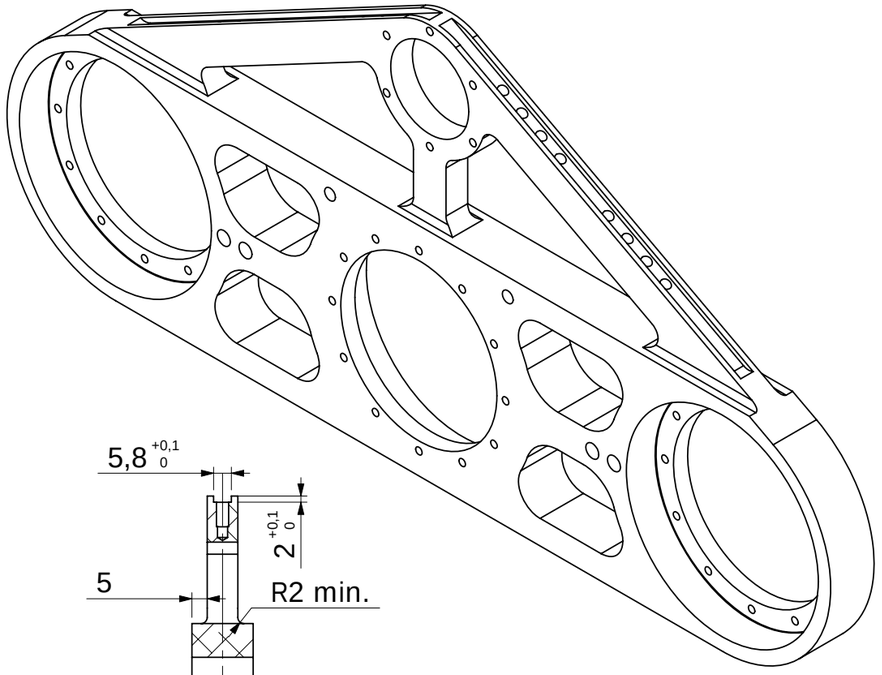
DWG NO. 6_1_b_Thigh

A3

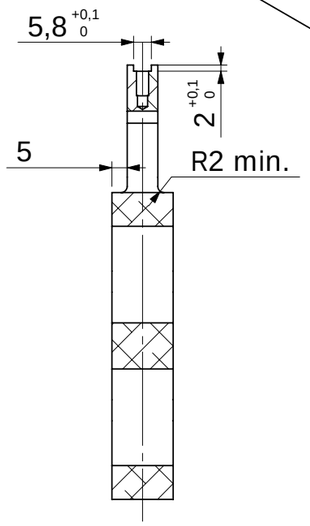
SCALE:1:5

SHEET 1 OF 1



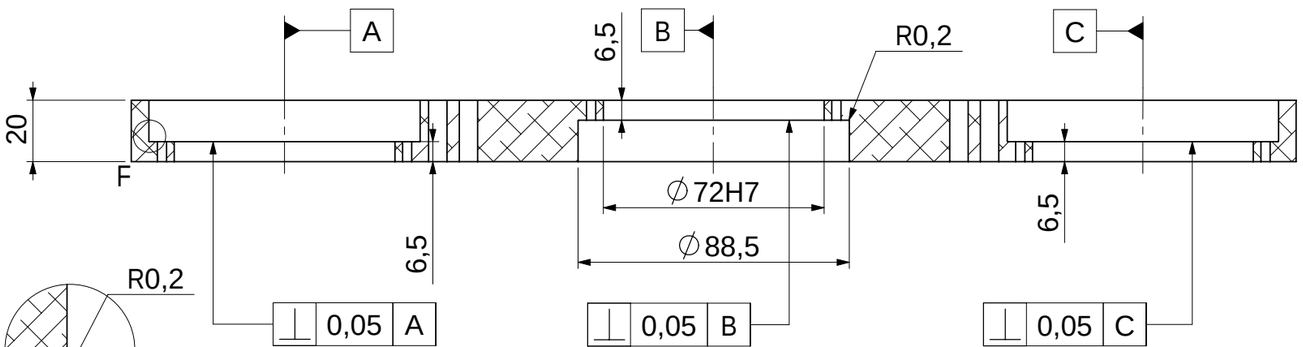


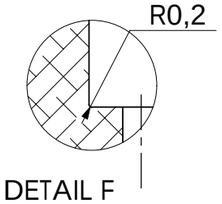
SECTION G-G
SCALE 1 : 2



SECTION D-D
SCALE 1 : 2

SECTION E-E
SCALE 1 : 2





DETAIL F
SCALE 2 : 1

NOTE:
(1): Udfræsninger fræses efter Pelvis_9_1.iges

(2): position og geometri af afslutning på rille er ikke væsentlig.

Fræses efter: Pelvis_9_1.iges

Ikke tolerancesat, benyt DS/ISO 2768-m

MATERIAL:
AW-6082 T6

NUMBER:
1 stk.

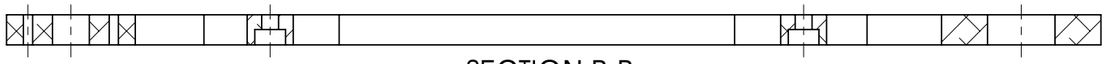
SCALE:
1:2

9.1 Pelvis

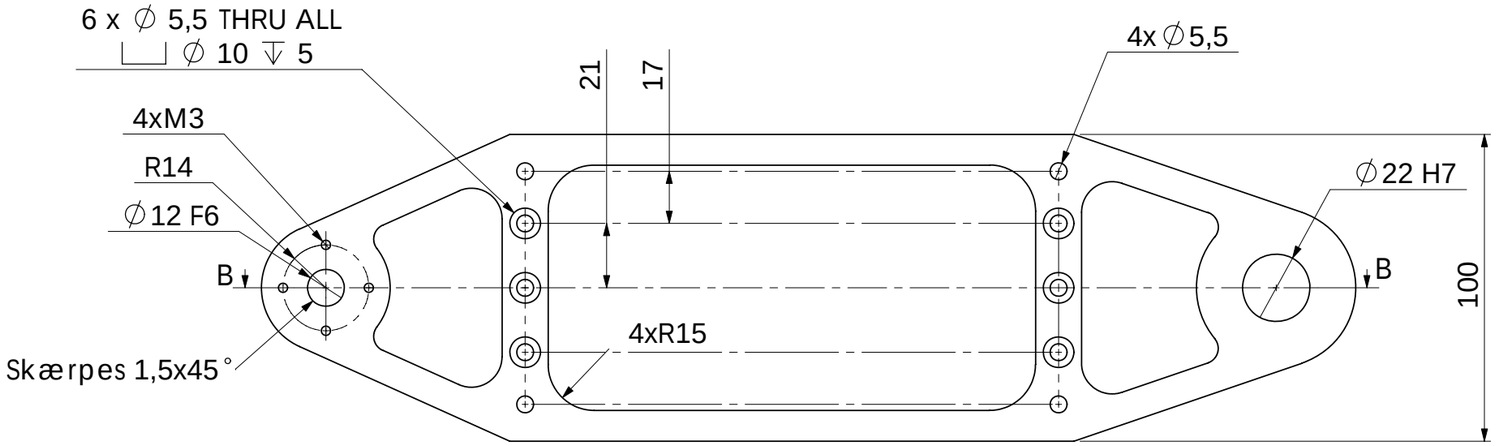
Aalborg Universitet
DMS10
Grp.43A, Pon 103

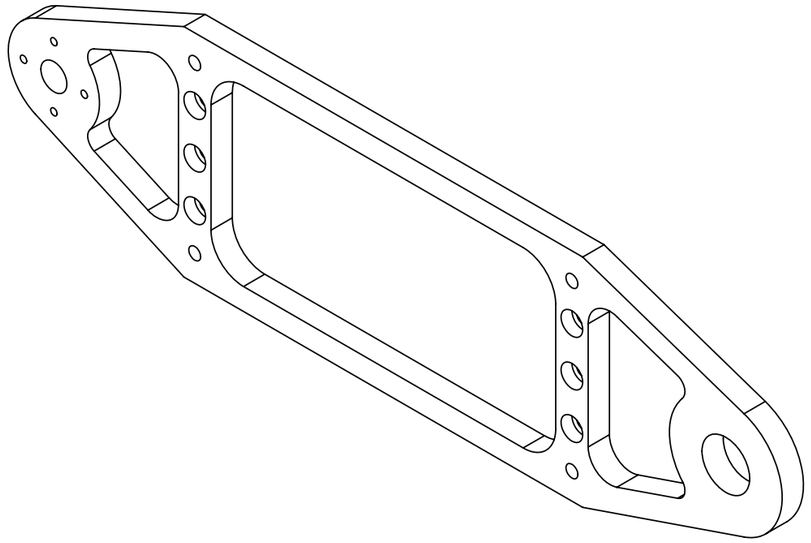
DWG NO.
Pelvis

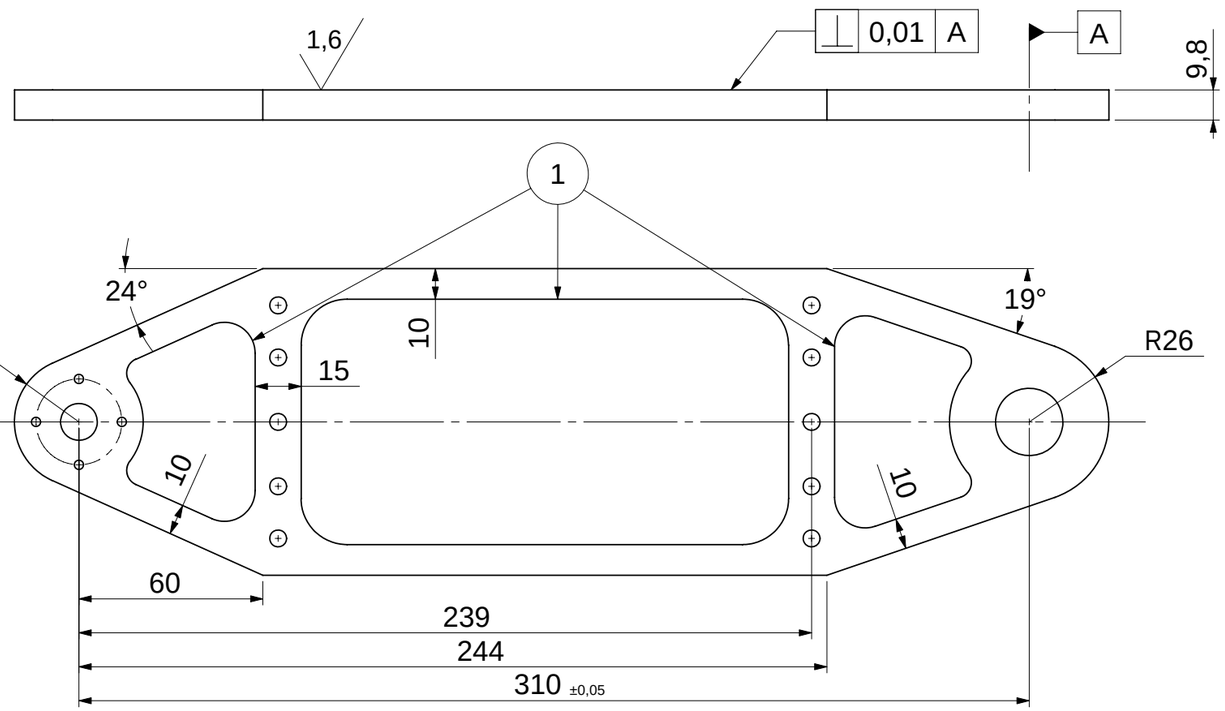
A3



SECTION B-B
SCALE 1 : 2







NOTE:
(1) Udfræsninger fræses efter: ThighPlateBearing_6_4.iges

Ikke tolerencesat; benyt DS/ISO 2768-m

MATERIAL:
AW 6082 T6

NUMBER:
2 stk

SCALE:
1:2

6.4 Thigh plate bearing side

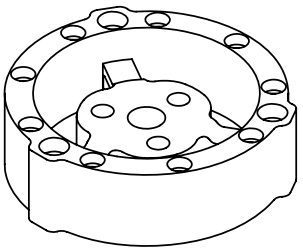
Aalborg Universitet
DMS10
Grp.43A, Pon 103

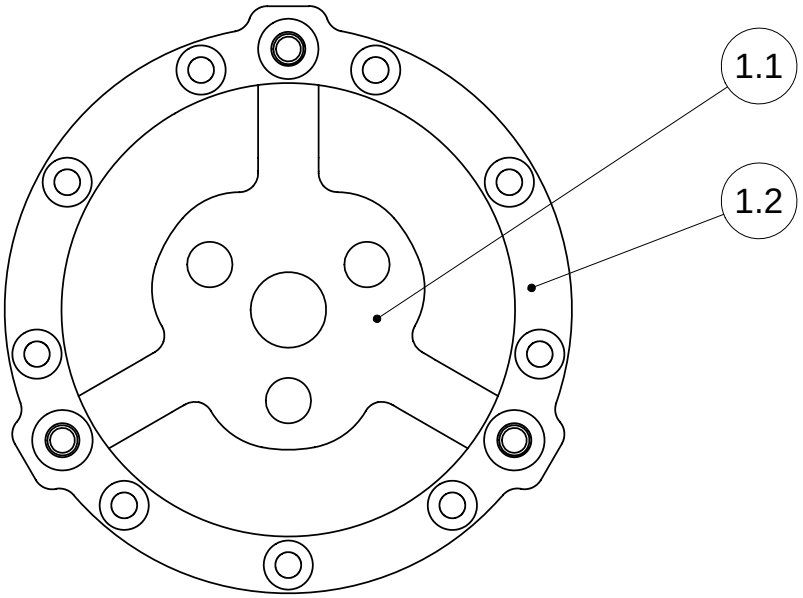
DWG NO 6_4_ThighPlateBearingSide

A3

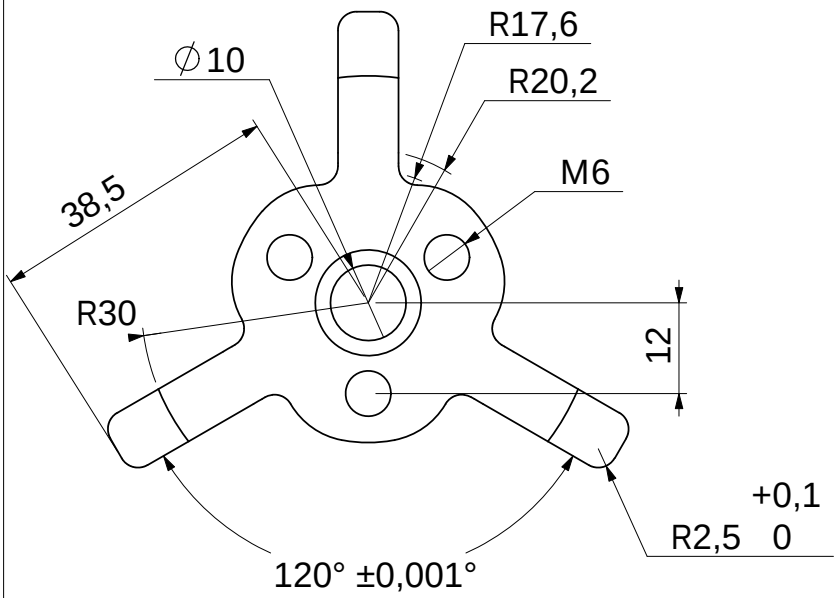
SCALE:1:5

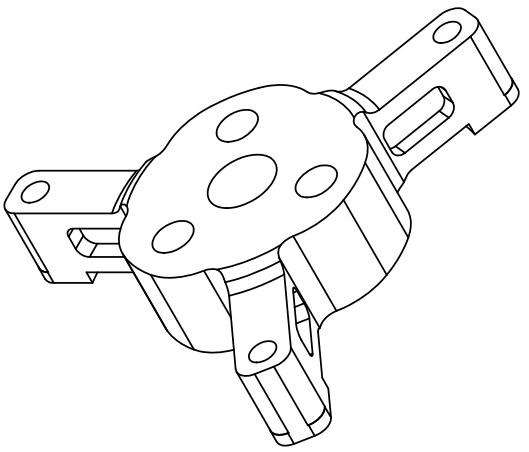
SHEET 1 OF 1

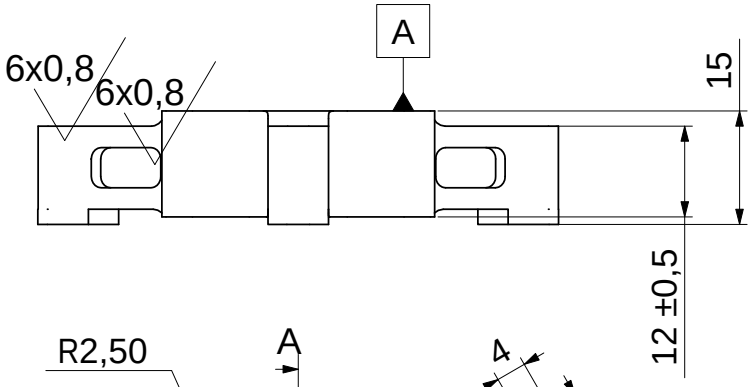


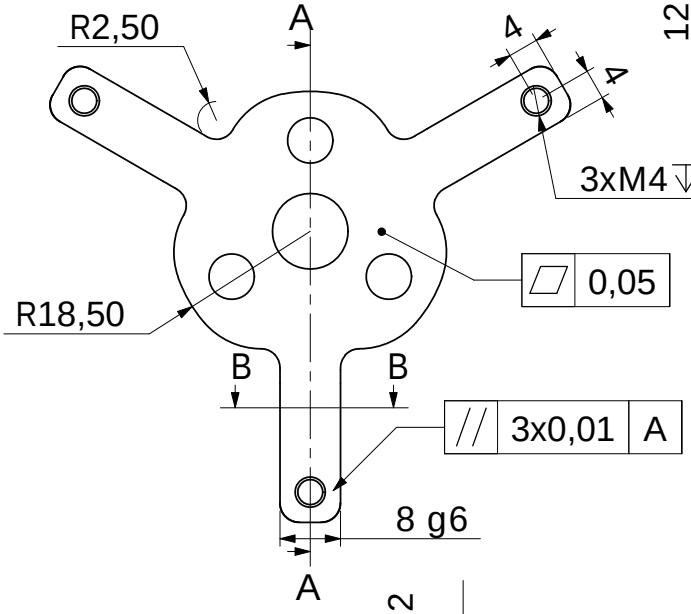


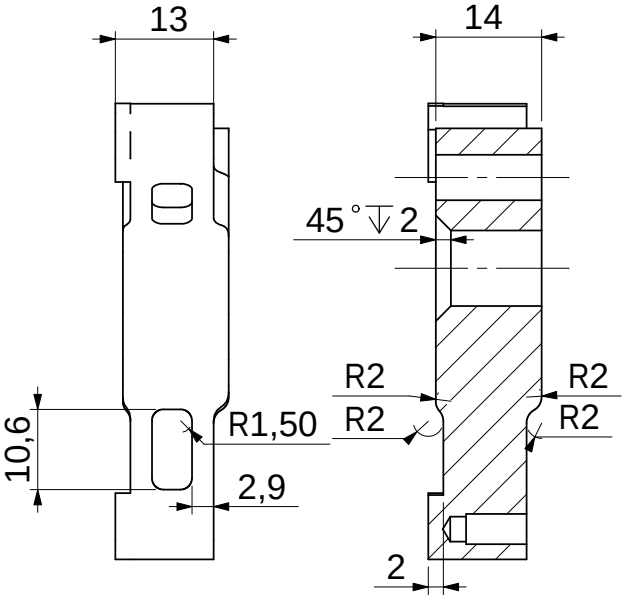
Note:	DIMENSIONS ARE IN mm. TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±		1.0 FTS test
	MATERIAL	DRAWN / CHECKED — NAME / DATE	Aalborg Universitet DMS10 Grp.43A, Pon 103
	Number 1 stk.	Forventet færdig:	SIZE A — DWG. NO. FTS4 — REV.
	SCALE:1:1	COMMENTS:	WEIGHT: — SHEET 1 OF 1



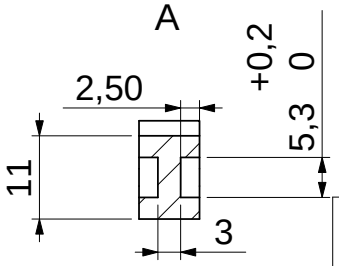








SECTION A-A



SECTION B-B

For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-f

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL Stål:C45

FINISH

DO NOT SCALE DRAWING

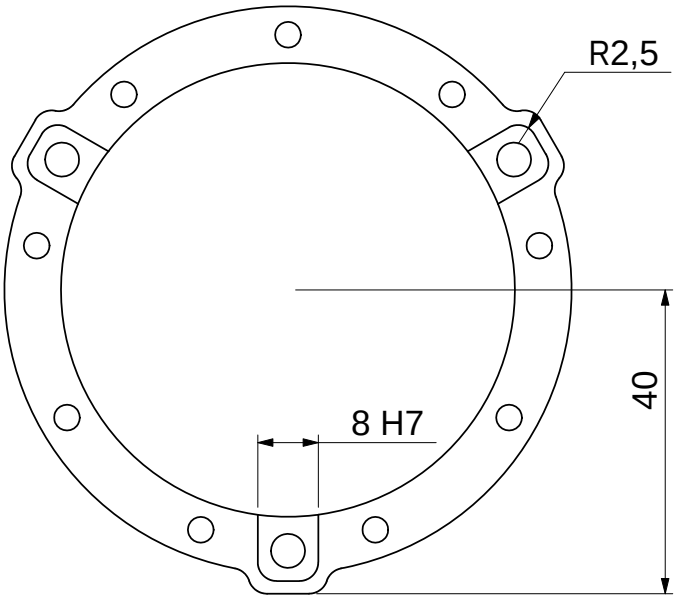
	NAME	DATE
DRAWN		
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

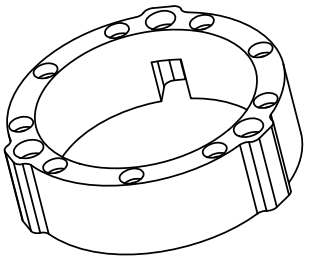
COMMENTS:

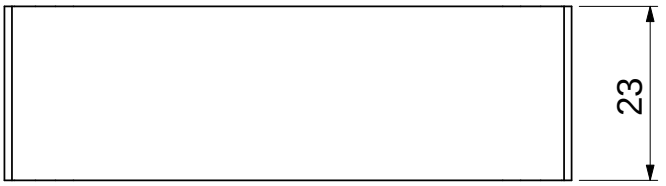
1.1 FTS test

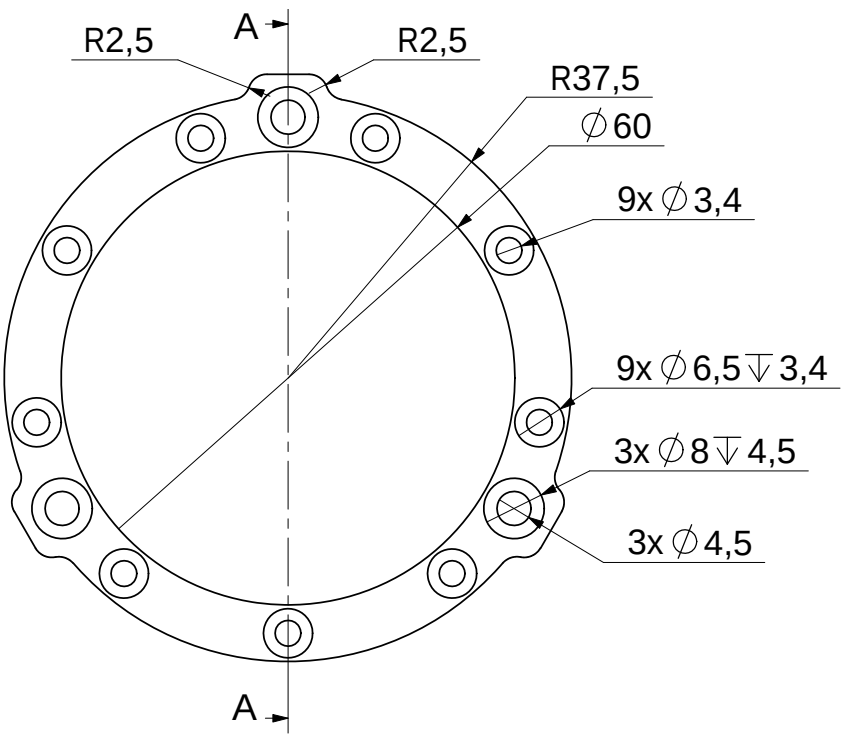
DMS 10
GRP. 126

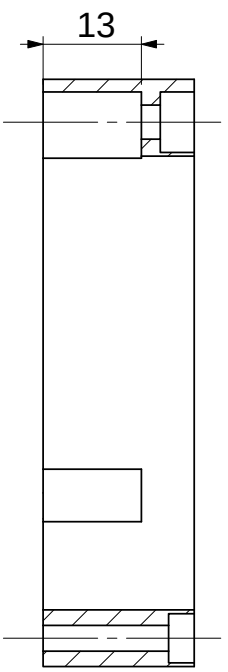
SIZE	DWG. NO.	REV.
A	FTS_midt4	
SCALE:1:1	WEIGHT: 0,13 kg	SHEET 1 OF 1











SECTION A-A
SCALE 1 : 1

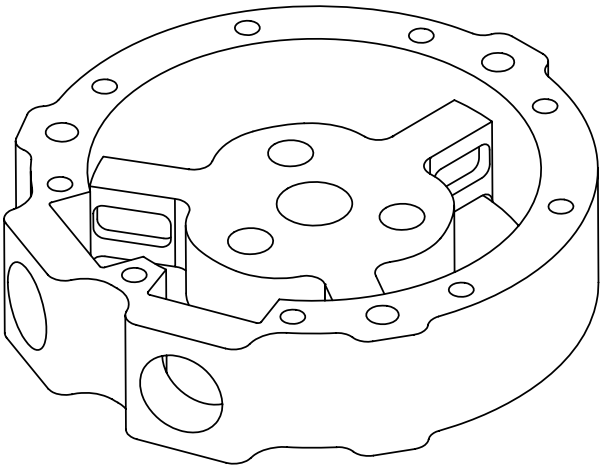
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-f

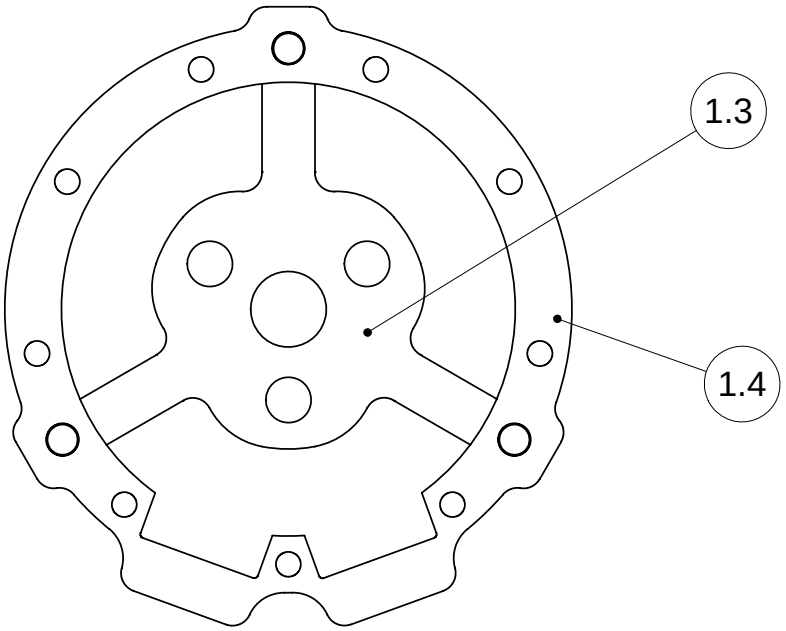
DIMENSIONS ARE IN mm.		NAME / DATE
TOLERANCES:	DRAWN	
FRACTIONAL ±	CHECKED	
ANGULAR: MACH ± BEND ±	ENG APPR.	
TWO PLACE DECIMAL ±	MFG APPR.	
THREE PLACE DECIMAL ±	Q.A.	
MATERIAL Stål: C45	COMMENTS:	
FINISH		
DO NOT SCALE DRAWING		

1.2 FTS_test

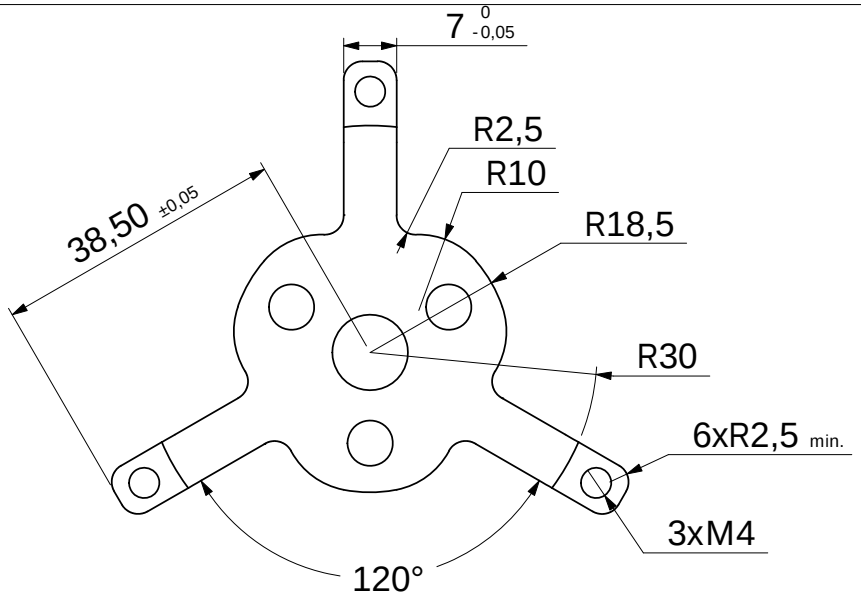
DMS10
GRP 126

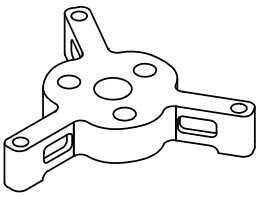
SIZE	DWG. NO.	REV.
A	FTS_yderring4	
SCALE:1:2	WEIGHT: 0,215kg	SHEET 1 OF 1

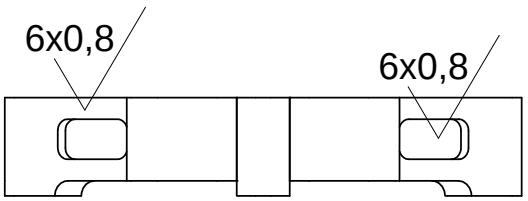


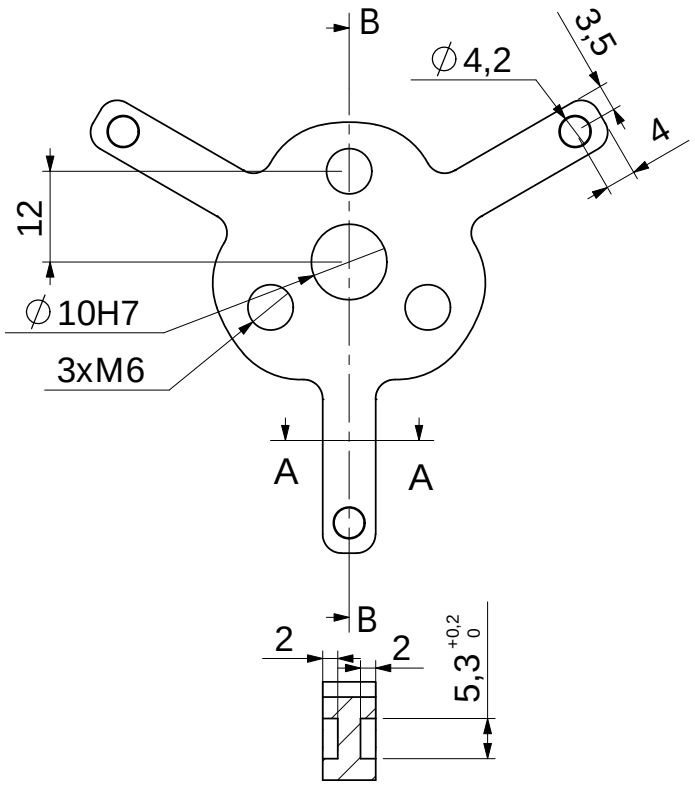


Note:	DIMENSIONS ARE IN mm. TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	NAME / DATE DRAWN CHECKED Forventet færdig: 15/6 COMMENTS:	1.0 New FTS Aalborg Universitet DMS10 Grp.43A, Pon 103
	MATERIAL		
	Number 2 stk.		SIZE A · DWG. NO. FTSnew · REV.
	SCALE:1:2		WEIGHT: · SHEET 1 OF 1

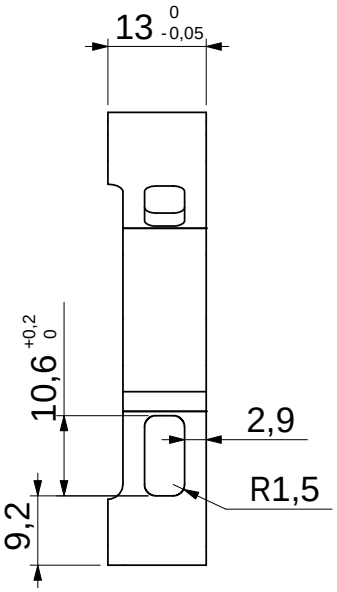


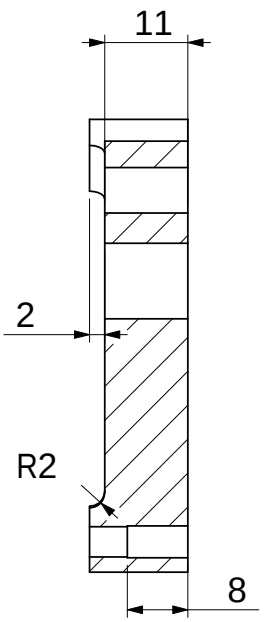






SECTION A-A
SCALE 1 : 1





SECTION B-B
SCALE 1 : 1

Note:
For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

Geometri-fil: NewFTS1_1.igs

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Stål:C45

Number: 2 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

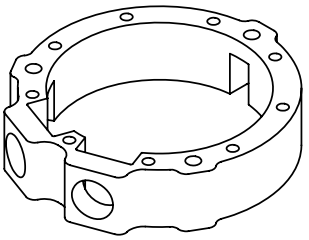
Forventet færdig: 15/6

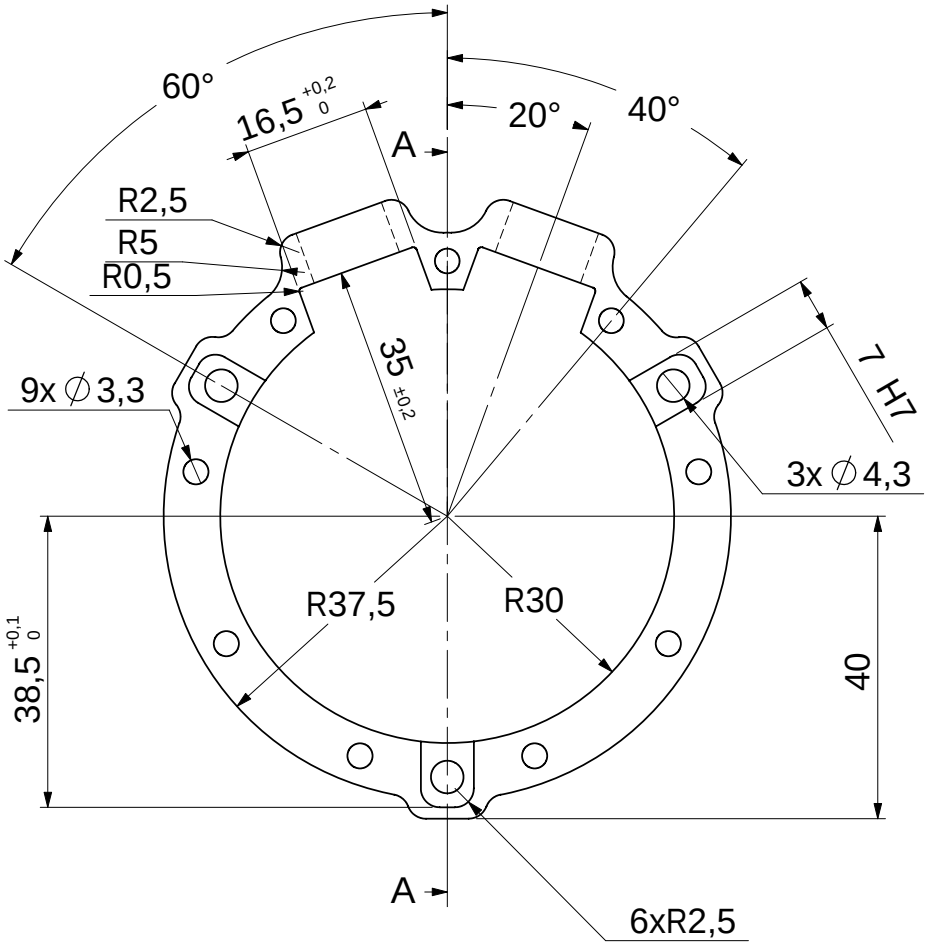
COMMENTS:

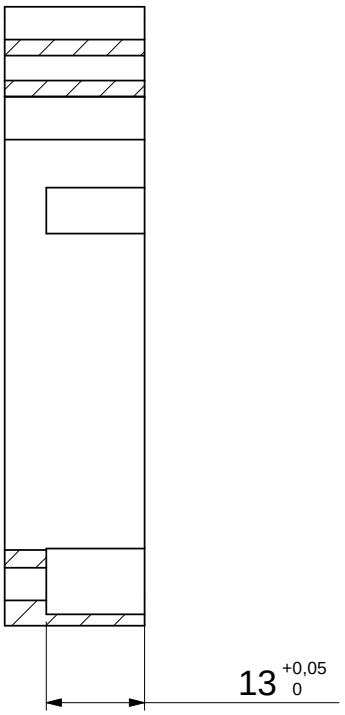
1.3 New FTS

Aalborg Universitet
DMS10
Grp.43A, Pon 103

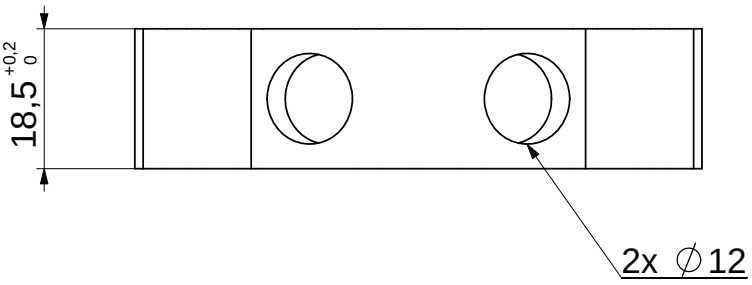
SIZE	DWG. NO.	REV.
A	FTS_midt4Ny	
	WEIGHT:	SHEET 1 OF 1







SECTION A-A
SCALE 1 : 1



Note:
For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

Geometri-fil: NewFTS1_2.igs

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 2 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig: 15/6

COMMENTS:

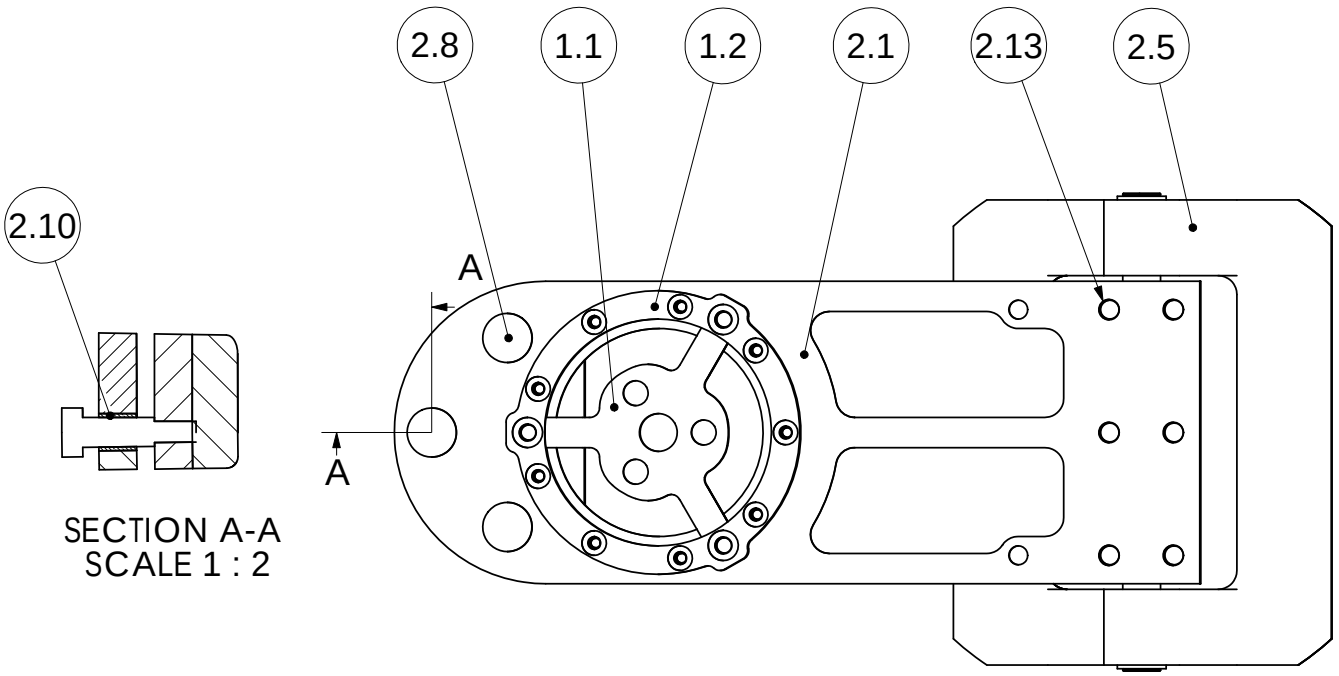
1.4 New FTS

Aalborg Universitet
DMS10
Grp.43A, Pon 103

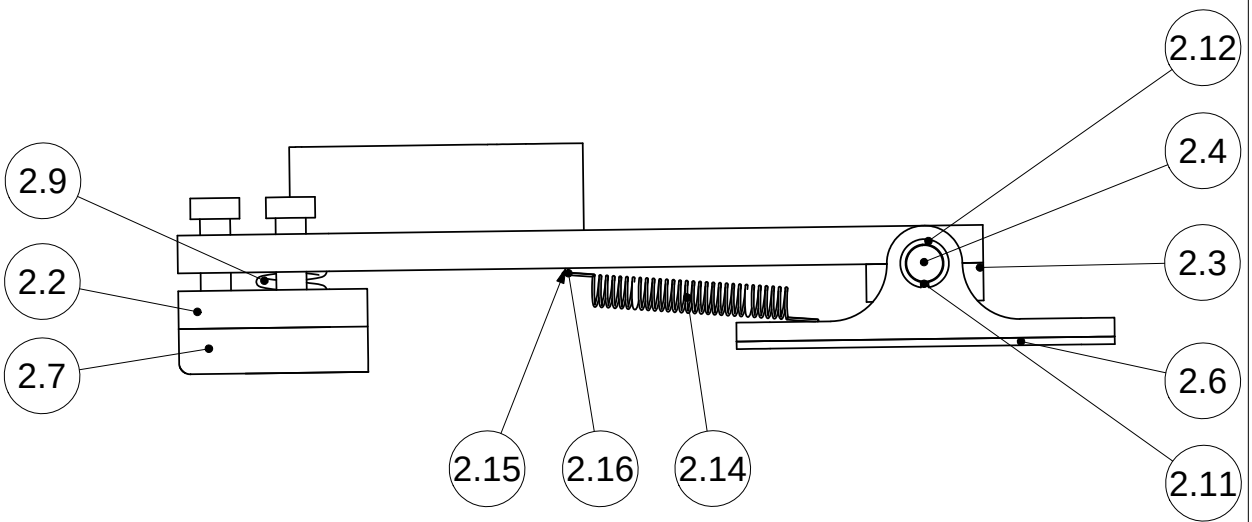
SIZE	DWG. NO.	REV.
A	FTS_yderring4Ny	

WEIGHT:

SHEET 1 OF 1



SECTION A-A
SCALE 1 : 2



Note:	DIMENSIONS ARE IN mm. TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	NAME / DATE DRAWN CHECKED Forventet færdig: COMMENTS:	2.A Foot Aalborg Universitet DMS10 Grp.43A, Pon 103
	MATERIAL Number 2 stk SCALE:1:2		SIZE A · DWG. NO. FootV3 · REV. WEIGHT: · SHEET 1 OF 1

Nummer	Betegnelse (*)	Materiale	Antal
2.1	Plade: 10mm	EN AW-6082 T6	2
2.2	Plade:20mm/ Aksel:Ø90	Al	2
2.3	Plade: 10 mm	Al	2
2.4	Aksle:Ø10	Stål	2
2.5	Plade: 30mm/ Aksel: Ø150	Al	2
2.6	Plade: 2mm	Gummi	2
2.7	Plade:12mm/ evt.støbes	Gummi	2
2.8	Pasbolt: M6-30mm	Vare nr.570895**	6
2.9	Trykfjeder: 2,5/18,5/41 (d/De/L0)	Lager nummer: 13000***	2
2.10	Glideleje: 8/10x10 (d/DxL)	ESSEM oliebronze: SS 777****	6
2.11	Glideleje: 10/12x20	ESSEM oliebronze: SS 777****	4
2.12	Sikringsring	Din 471-10x1	2
2.13	Unbrako bolt	5Mx20	12
2.14	Trækfjeder: 0,6/8/59	Lager nummer: 504***	2
2.15	Unbrako bolt	5Mx6	4
2.16	Spændskive	ø5	4

Note:

* Forslag til dimension af råemne
** Sanistål
*** SODEMANN Industrifjedre A/S
**** Dansk Materiale Teknik A/S

	NAME	DATE
DRAWN		
CHECKED		

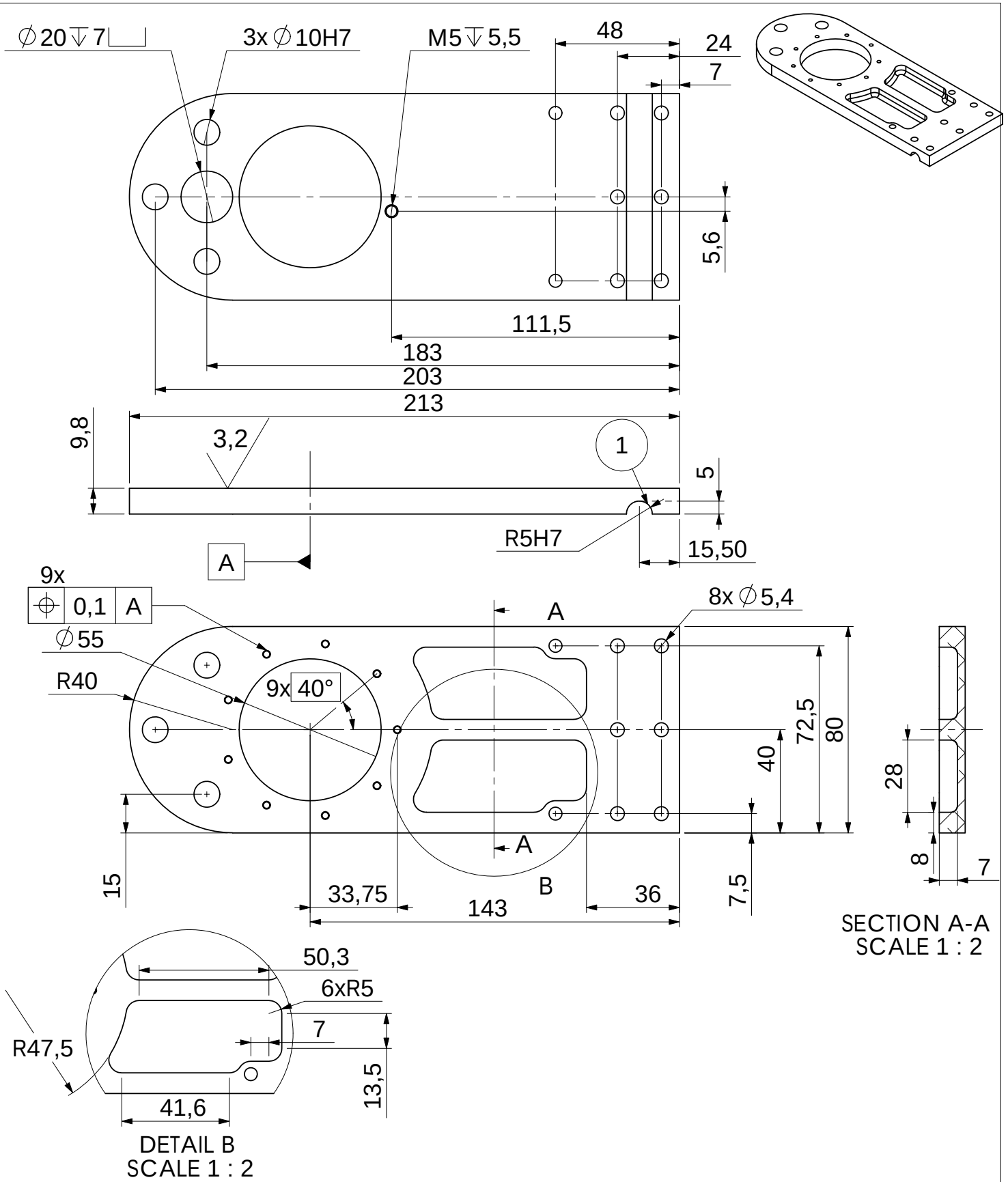
Forventet færdig:

COMMENTS:

2.0 Foot

Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	2_0_Foot	
	WEIGHT:	SHEET 1 OF 1



Note:

(1) Hullet bores efter at (2.3) er monteret.

Mål angivet ved Detail B, er alle med en tolerance ±0,5

For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 2 stk

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

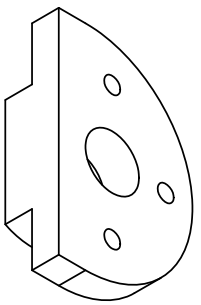
Forventet færdig:

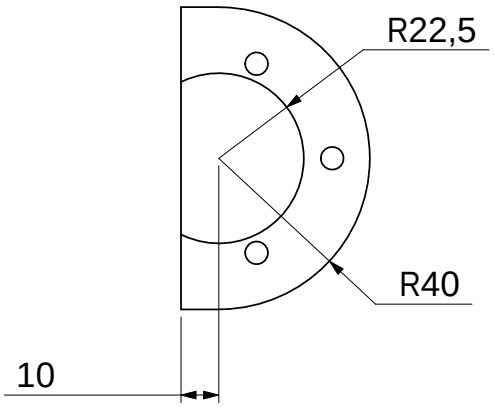
COMMENTS:

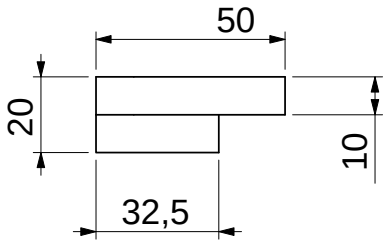
2.1 Foot

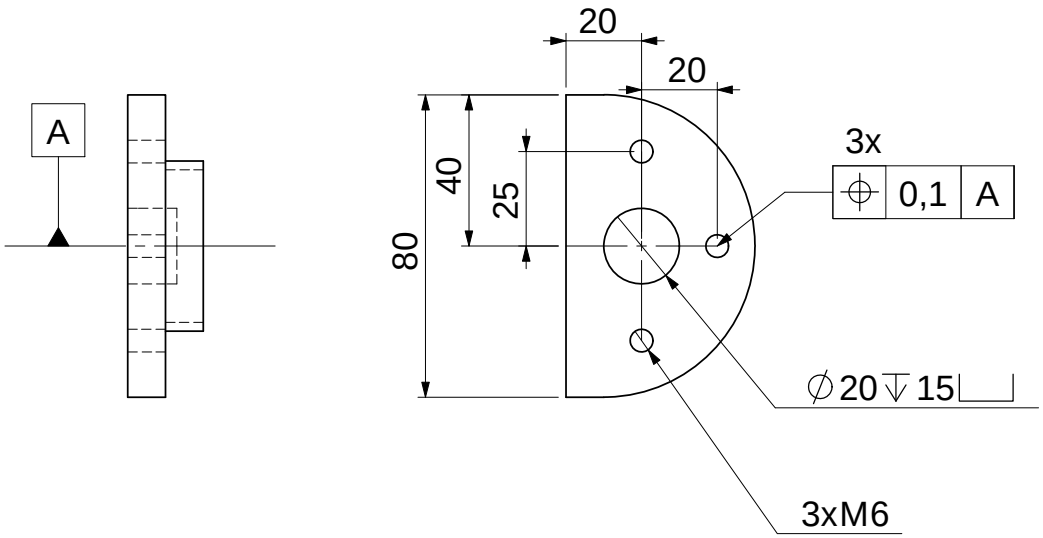
Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	2_1_Foot	
	WEIGHT:	SHEET 1 OF 1









Note:

For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Al

Number: 2 stk

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

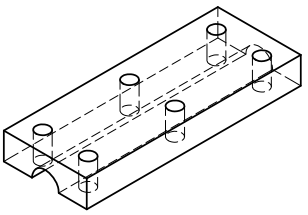
Forventet færdig:

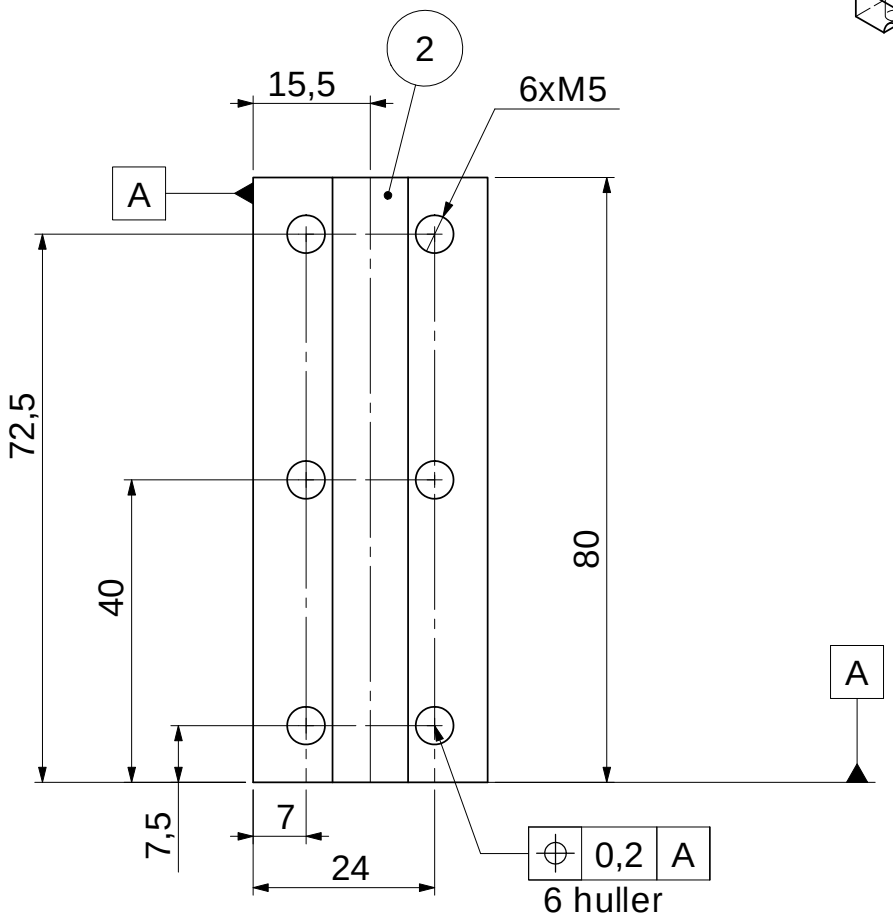
COMMENTS:

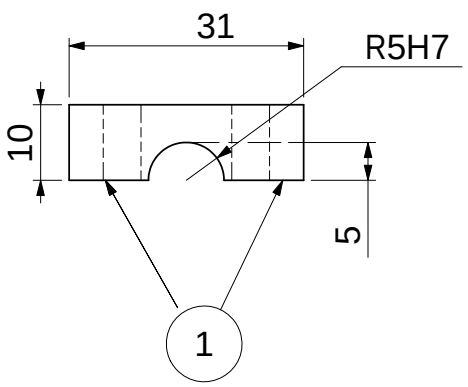
2.2 Foot

Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	2_2_Foot	
	WEIGHT:	SHEET 1 OF 1







Note:

(1) Flade planslibes 0,1 mm efter bearbejdning af hullet (2)

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Al

Number: 2 stk

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

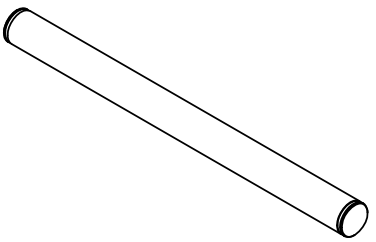
Forventet færdig:

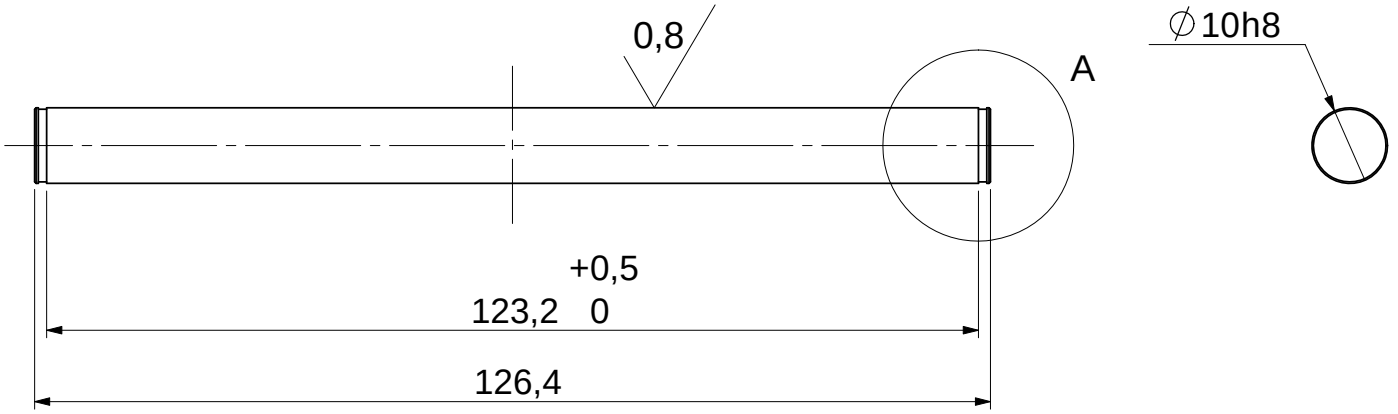
COMMENTS:

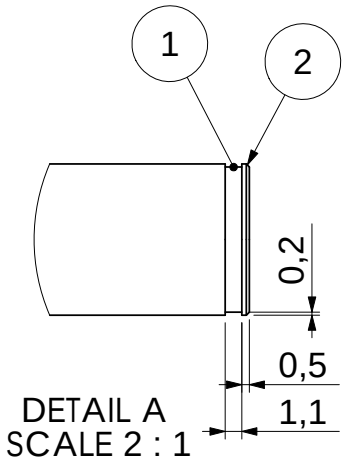
2.3 Foot

Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	2_3_Foot	
	WEIGHT:	SHEET 1 OF 1







Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

(1) DIN 471-10x1
(2) Kant reifes 45°

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Stål

Number: 1 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

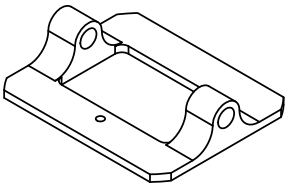
Forventet færdig:

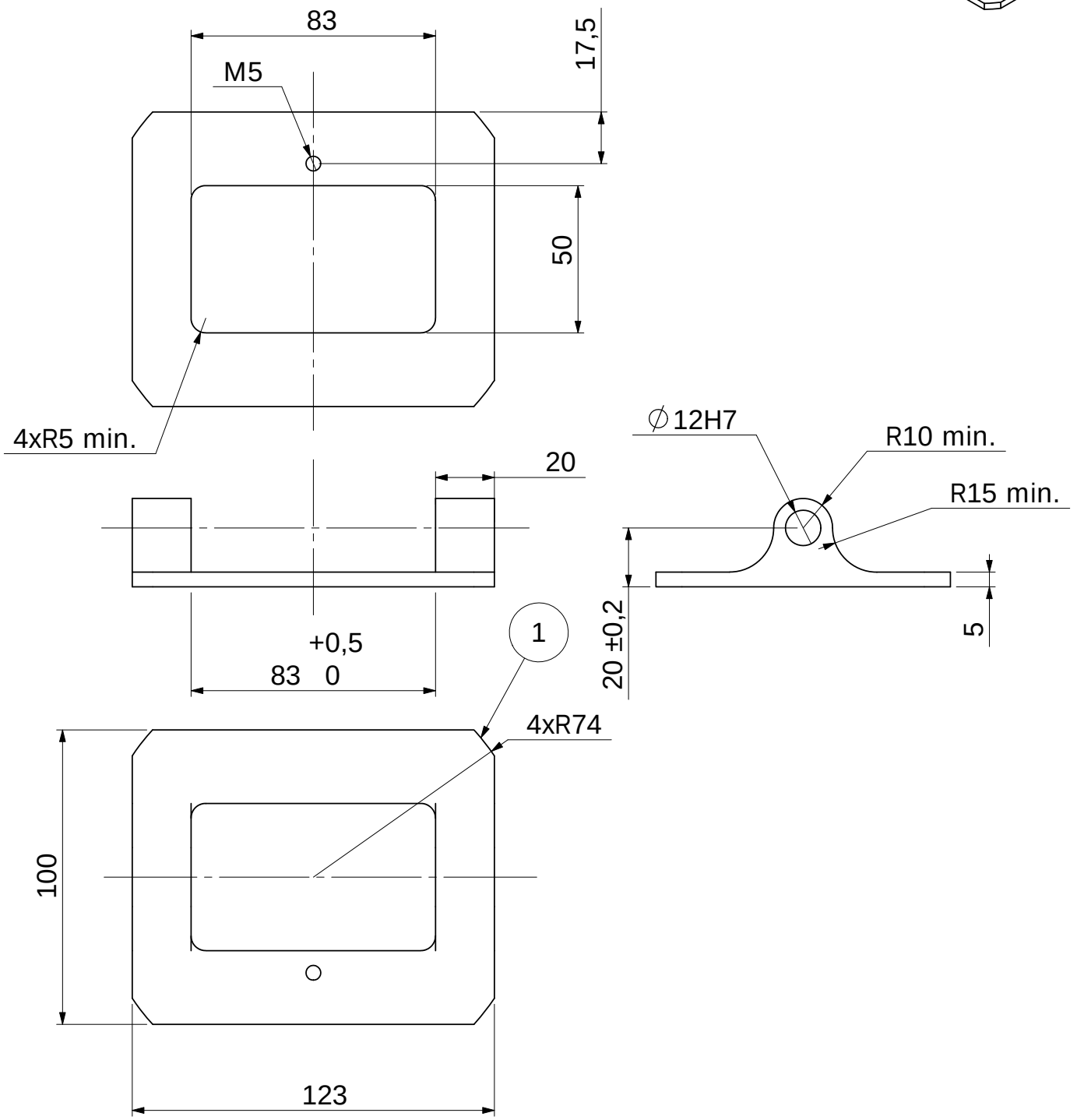
COMMENTS:

2.4 Foot

Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	2_4_Foot	
	WEIGHT:	SHEET 1 OF 1





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

(1) Hjørnet kan evt. laves 7x45°.

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Al

Number: 1 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

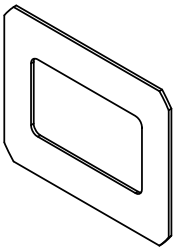
Forventet færdig:

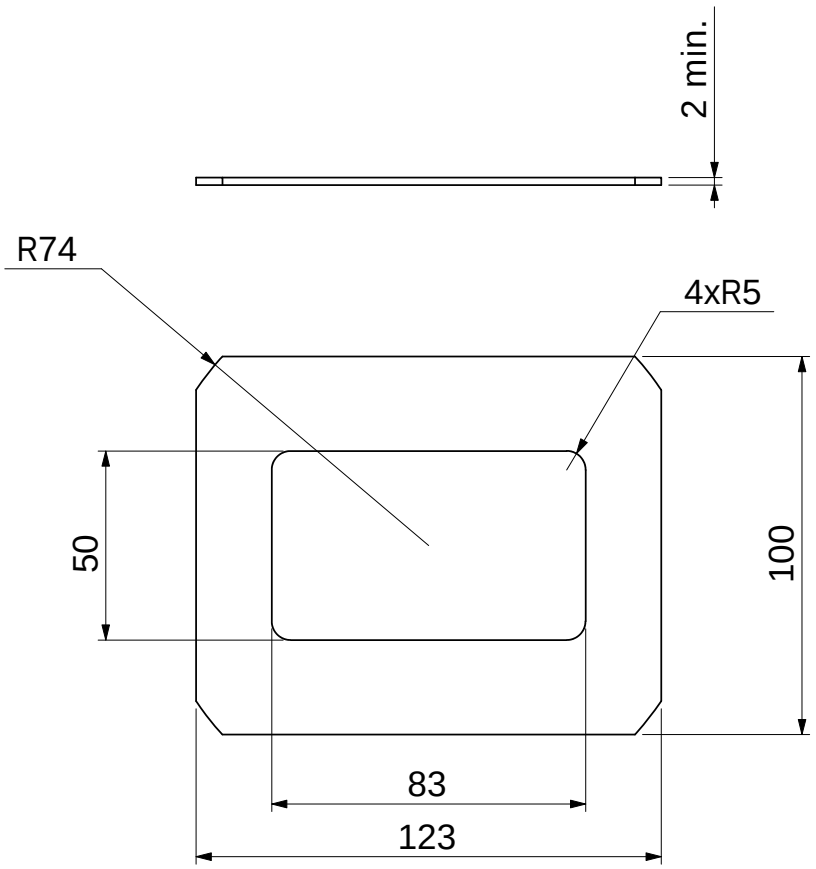
COMMENTS:

2.5 Foot

Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	2_5_Foot	
	WEIGHT:	SHEET 1 OF 1





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-G

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Gummi

Number: 2 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

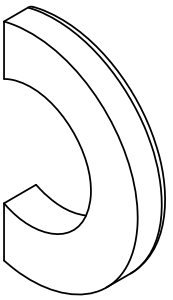
Forventet færdig:

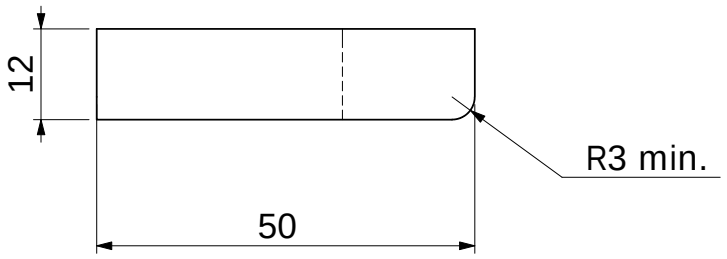
COMMENTS:

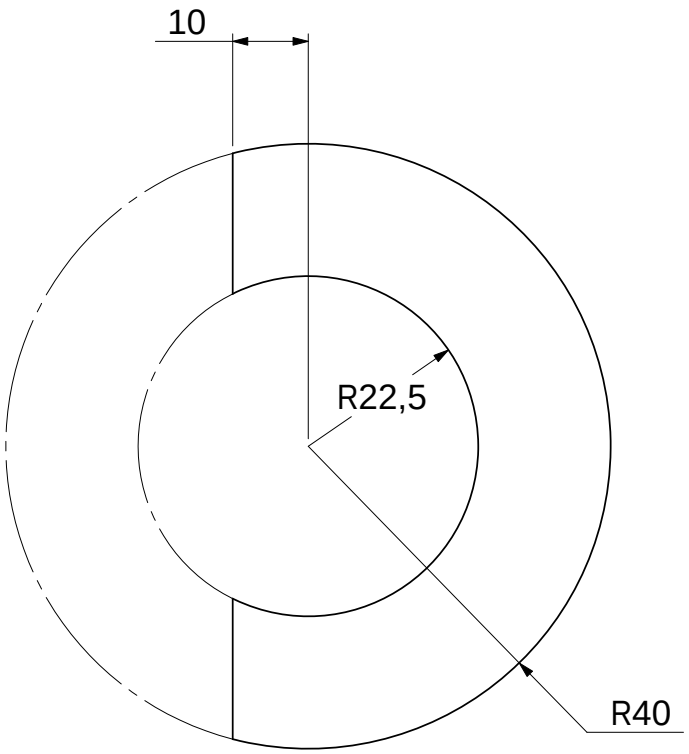
2.6 Foot

Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	2_6_Foot	
	WEIGHT:	SHEET 1 OF 1







Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-G

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Gummi

Number: 2 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

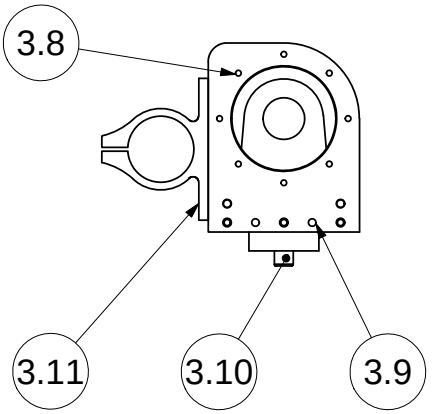
Forventet færdig:

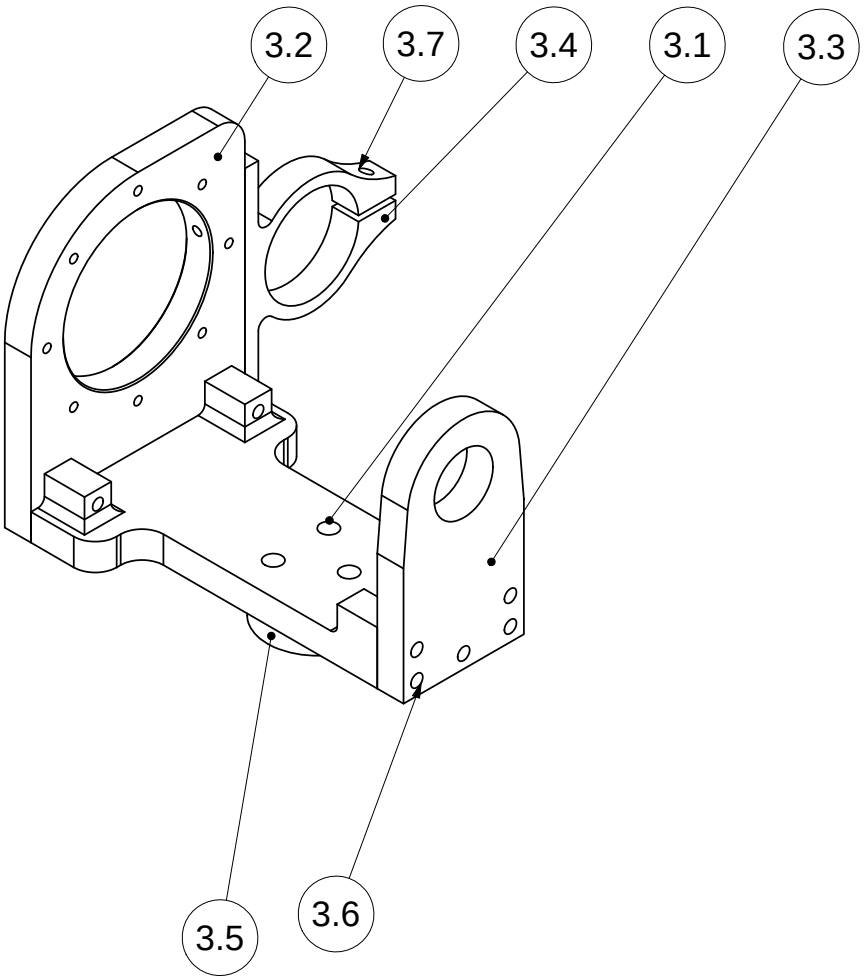
COMMENTS:

2.7 Foot

Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	2_7_Foot	
	WEIGHT:	SHEET 1 OF 1





Note:	DIMENSIONS ARE IN mm. TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±		NAME / DATE	3.A AnkleBracket
		DRAWN		Aalborg Universitet DMS10 Grp.43A, Pon 103
		CHECKED		
	MATERIAL	Forventet færdig:		
	Number 2 stk.	COMMENTS:		SIZE A · DWG. NO. 3_A_AnkleBracket · REV.
	SCALE:1:2			WEIGHT: · SHEET 1 OF 1

Nummer	Betegnelse (*)	Materiale	Antal
3.1	Plade: 20mm	EN AW-6082 T6	2
3.2	Plade:10mm	EN AW-6082 T6	2
3.3	Plade: 10 mm	EN AW-6082 T6	2
3.4	Plade: 10mm	EN AW-6082 T6	2
3.5	Aksel: Ø40	Al	2
3.6	Unbrako skrue M4x25mm		20
3.7	Unbrako skrue M3x25mm. m møtrik		16
3.8	Unbrako skrue M4x15mm.		2
3.9	Cylinderstift Ø4x20mm		4
3.10	Cylinderstift Ø10x25		2
3.11	Unbrako skrue M4x15mm.		8

Note:

* Forslag til dimension af råemne

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

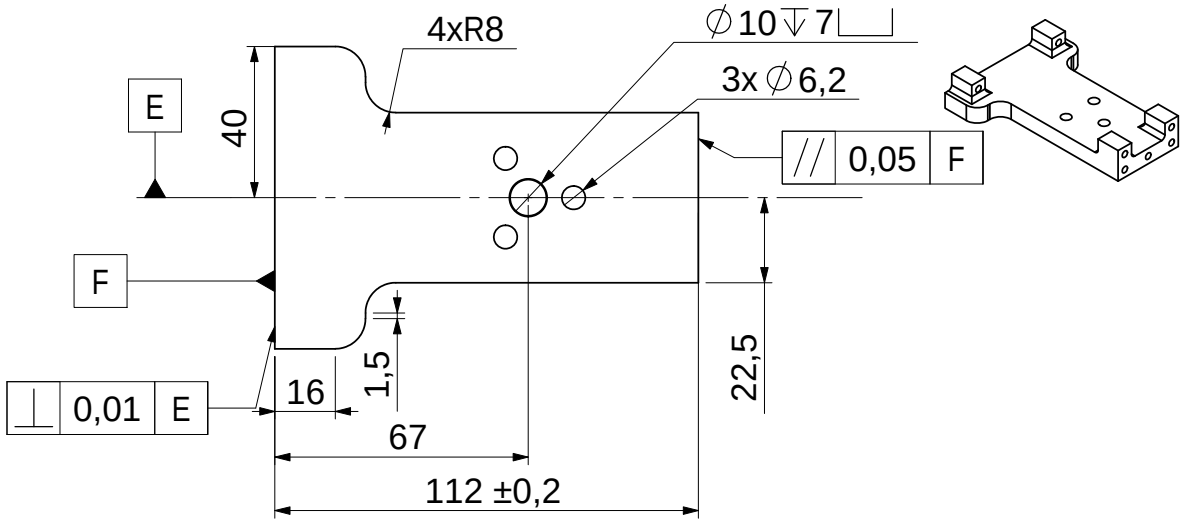
COMMENTS:

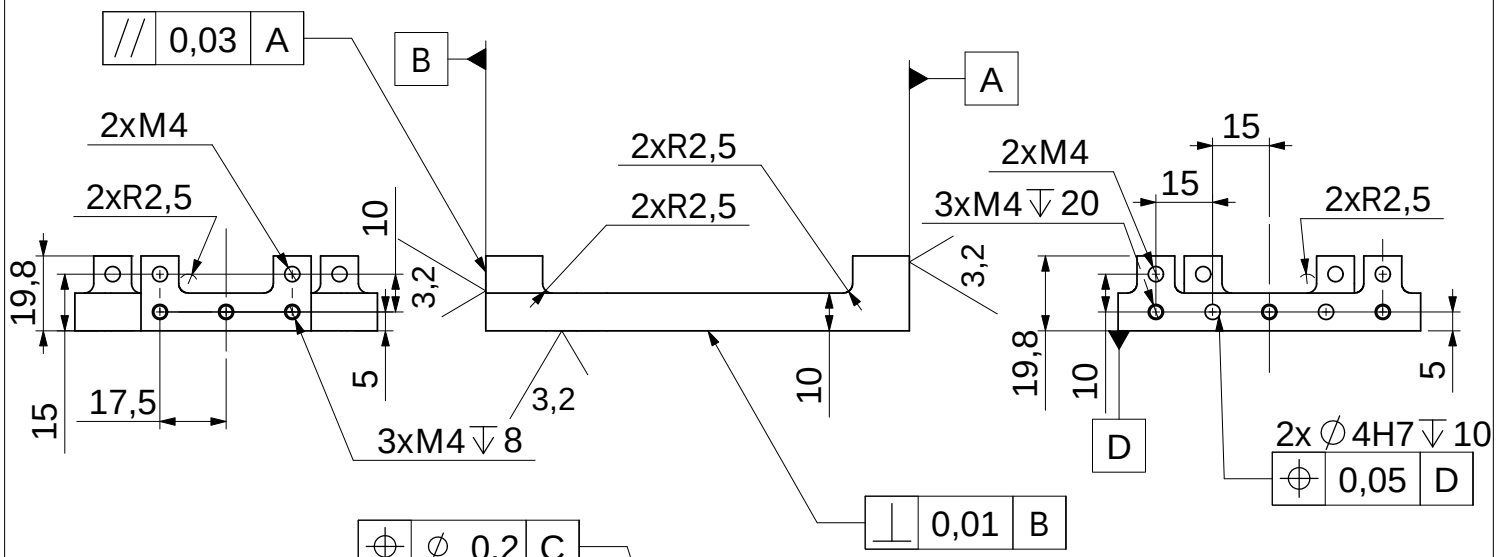
3.0 AnkleBracket

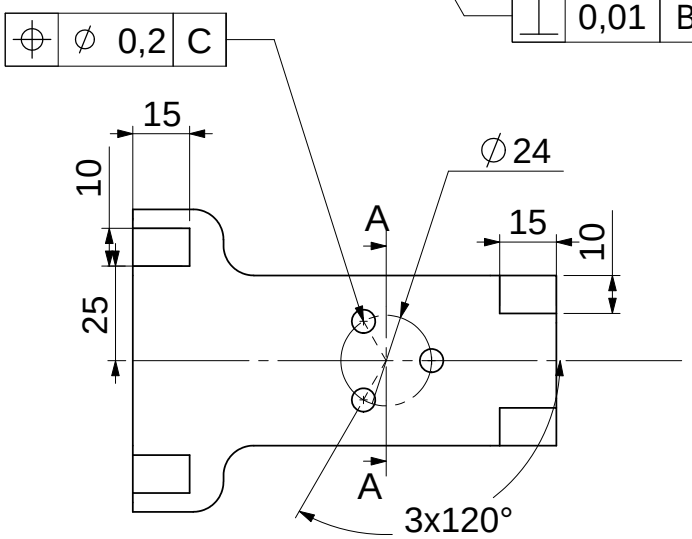
Aalborg Universitet
DMS10
Grp.43A, Pon 103

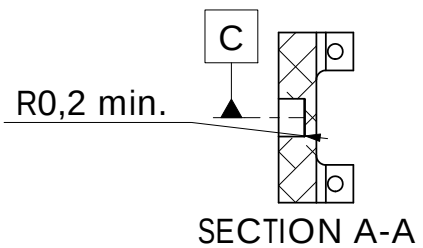
SIZE	DWG. NO.	REV.
A	3_0_AnkleBracket	

WEIGHT: SHEET 1 OF 1









Note:
For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 2 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

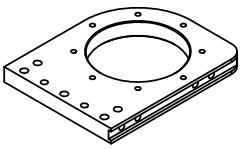
COMMENTS:

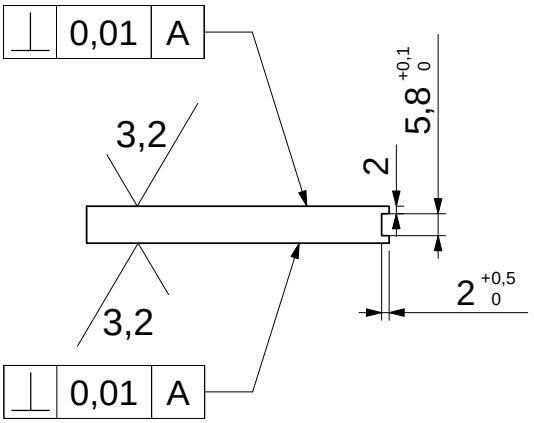
3.1 AnkleBracket

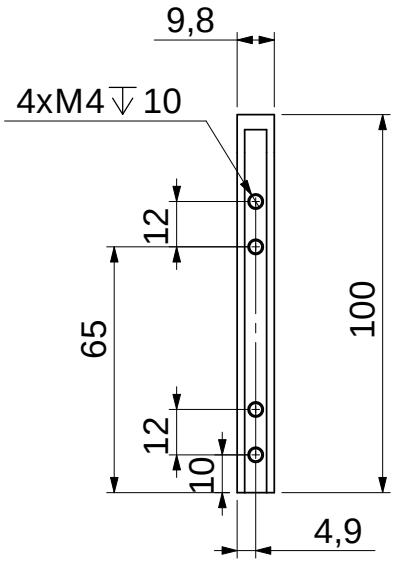
Aalborg Universitet
DMS10
Grp.43A, Pon 103

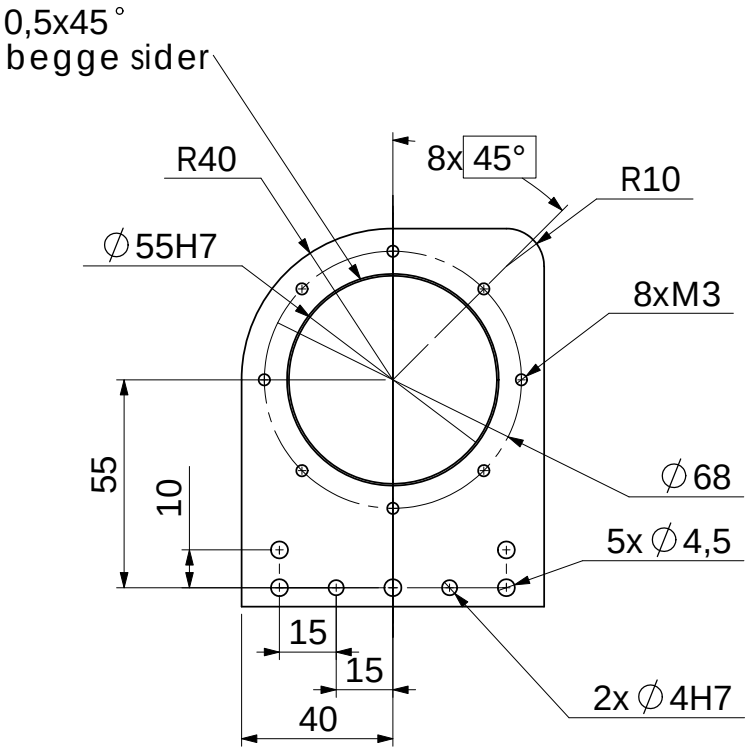
SIZE	DWG. NO.	REV.
A	3_1_AnkleBracket	

WEIGHT: SHEET 1 OF 1









Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 2 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

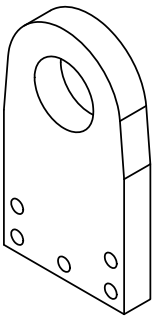
3.2 AnkleBracket

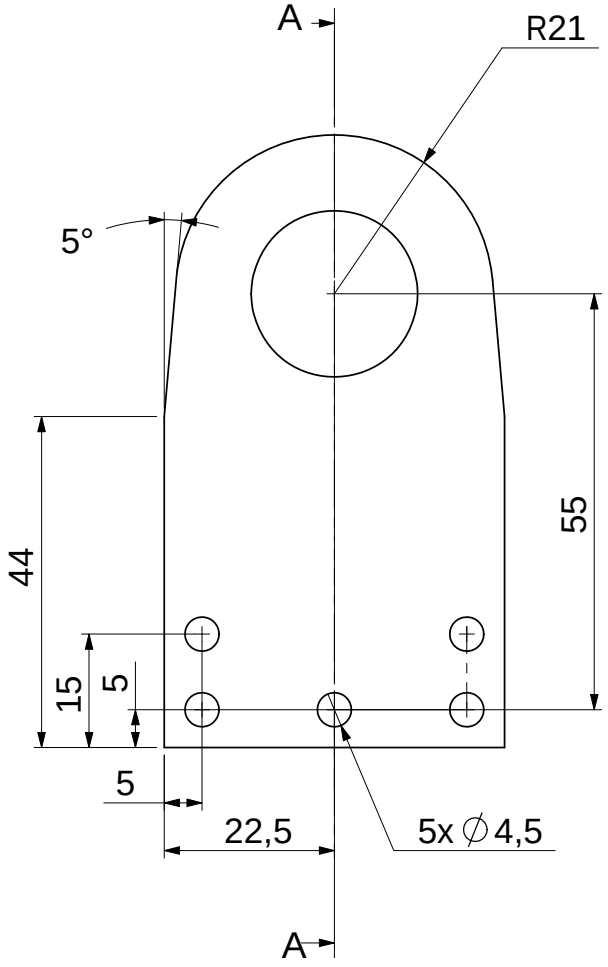
Aalborg Universitet
DMS10
Grp.43A, Pon 103

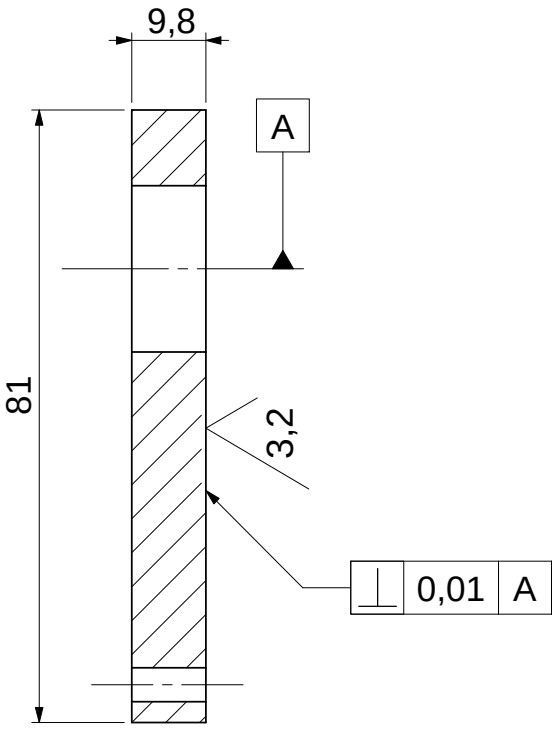
SIZE	DWG. NO.	REV.
A	3_2_AnkleBracket	

WEIGHT:

SHEET 1 OF 1







SECTION A-A

Note:
For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 2 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

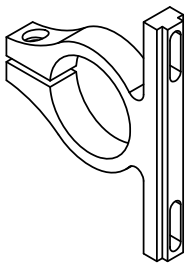
COMMENTS:

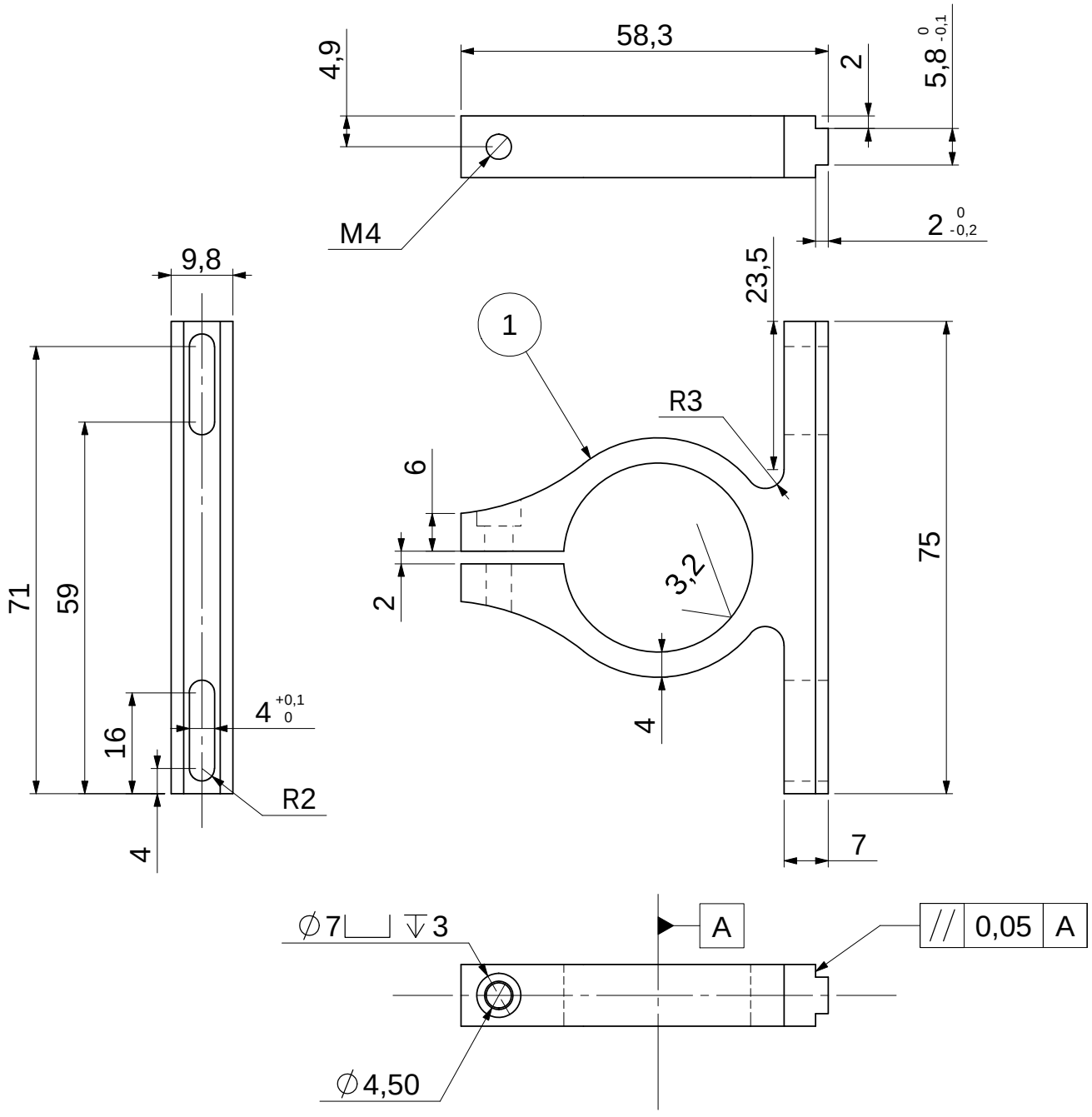
3.3 AnkleBracket

Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	3_3_AnkleBracket	

WEIGHT: | SHEET 1 OF 1





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

(1) Geometrifil:
4_2_CrossAkselAnkle.igs

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 2 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

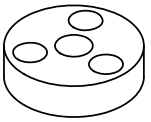
COMMENTS:

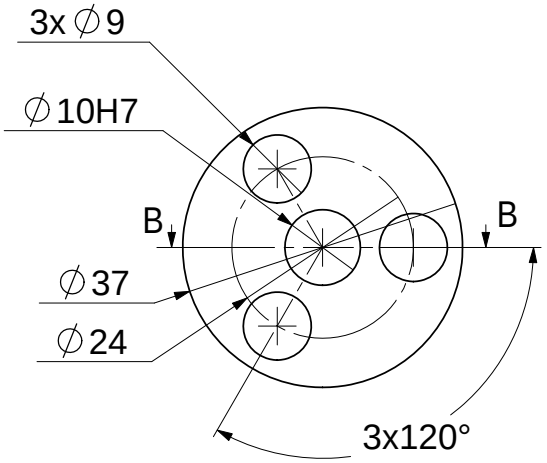
3.4 AnkleBracket

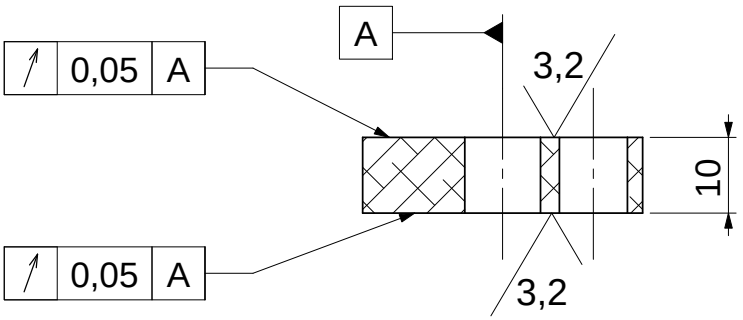
Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE A | DWG. NO. 3_4_AnkleBracket | REV.

WEIGHT: | SHEET 1 OF 1

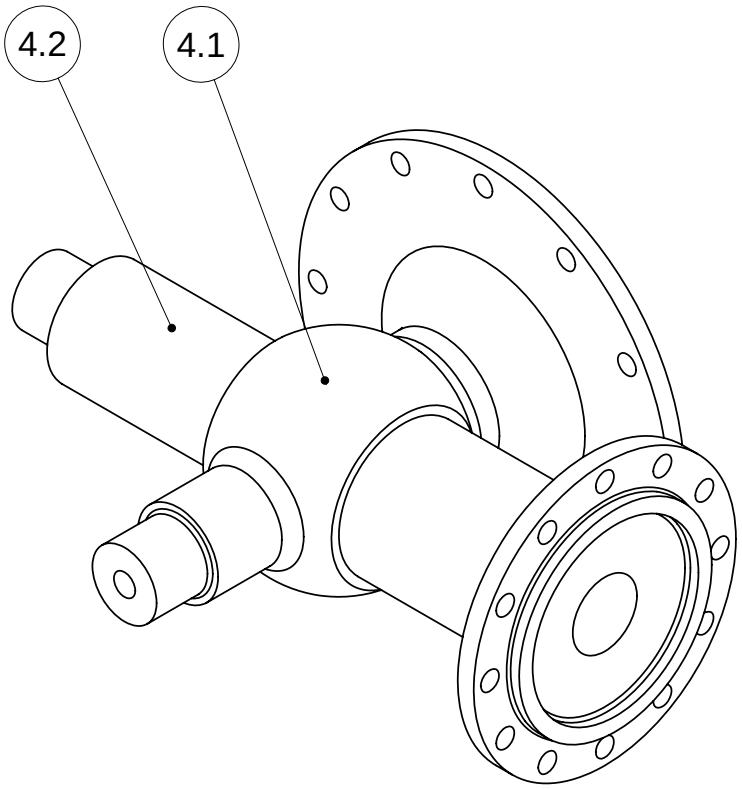






SECTION B-B

Note: For ikke tolerance specificeret mål benyttes DS/ISO 2768-m	DIMENSIONS ARE IN mm. TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	NAME / DATE DRAWN CHECKED Forventet færdig: COMMENTS:	3.5 AnkleBracket Aalborg Universitet DMS10 Grp.43A, Pon 103
	MATERIAL Al Number 2 stk. SCALE:1:1		SIZE A · DWG. NO. 3_5_AnkleBracket · REV. WEIGHT: · SHEET 1 OF 1

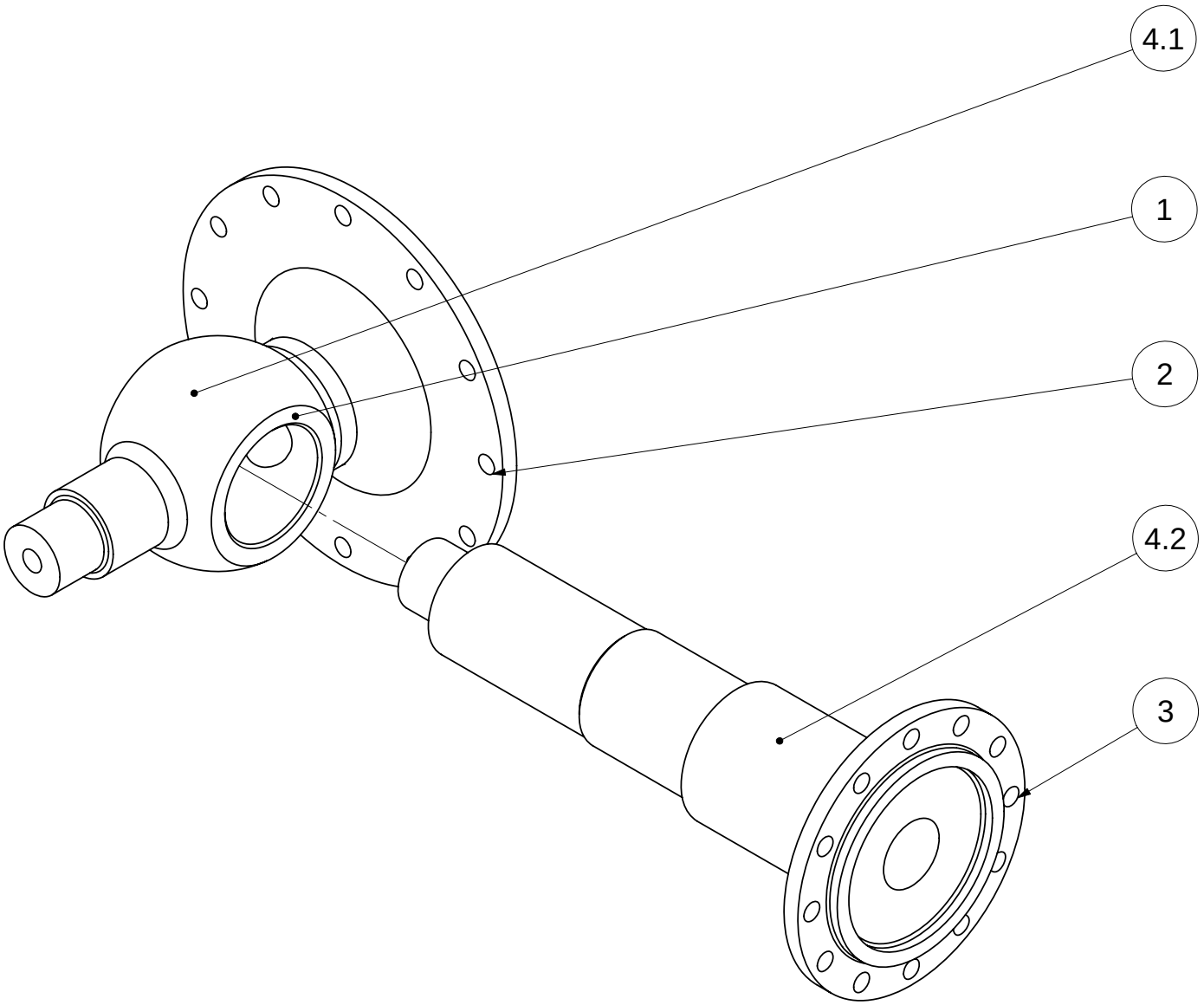


Note:			NAME	DATE
		DRAWN		
		CHECKED		
		Forventet færdig:		
	MATERIAL	COMMENTS:		
	Number 2 stk			
	SCALE:1:1			

4.A CrossAxleAnkle

Aalborg Universitet
DMS10
Grp.126

SIZE	DWG. NO.	REV.
A	4_A_CrossAxleAnkle	
	WEIGHT:	SHEET 1 OF 1



Note:

Krympning af CrossAxle Ankle.
(4.1) opvarmes til 270°C, hvorefter aksel (4.2) anlægges mod bryst(1). Hullerne (2) og (3) skal ikke passe over for hinanden.

NB:akslen skal først samles når gearene til robotten er modtaget.

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

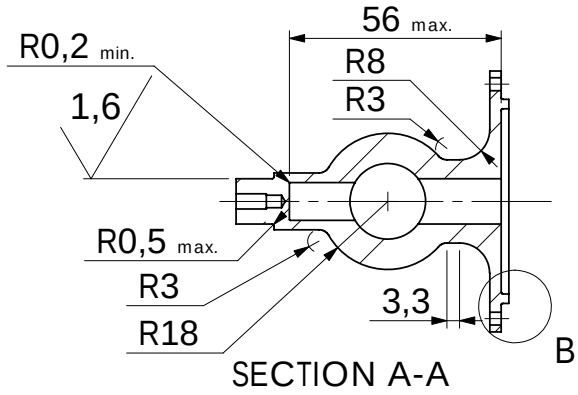
COMMENTS:

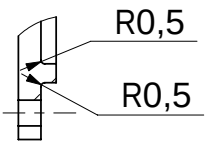
4.B CrossAxle Ankle

Aalborg Universitet
DMS10
Grp.43A, Pon 103

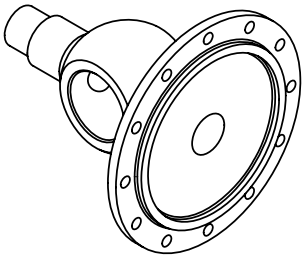
SIZE	DWG. NO.	REV.
A	4_B_CrossAxleAnkle	

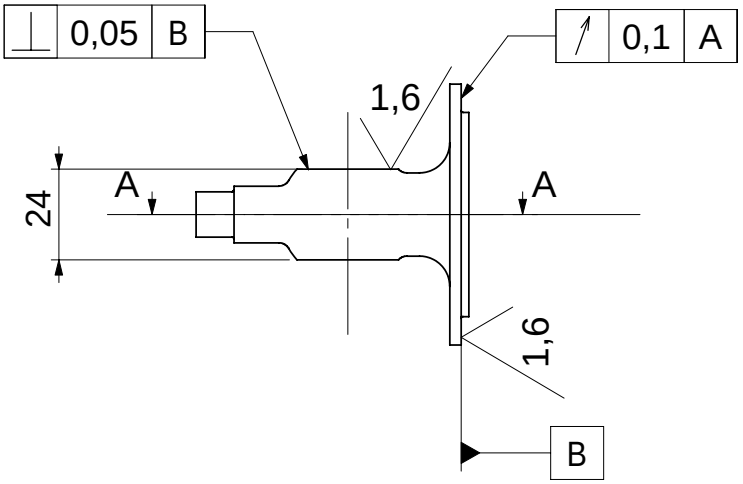
WEIGHT: SHEET 1 OF 1

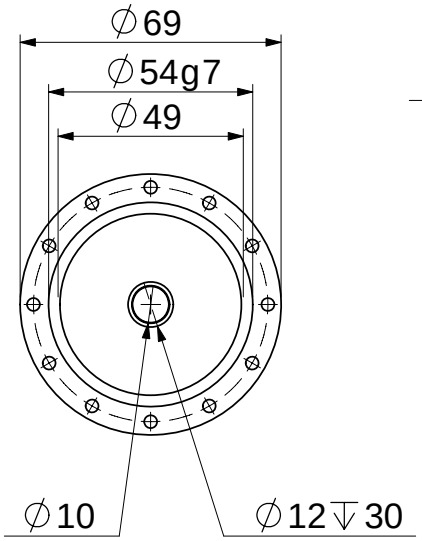


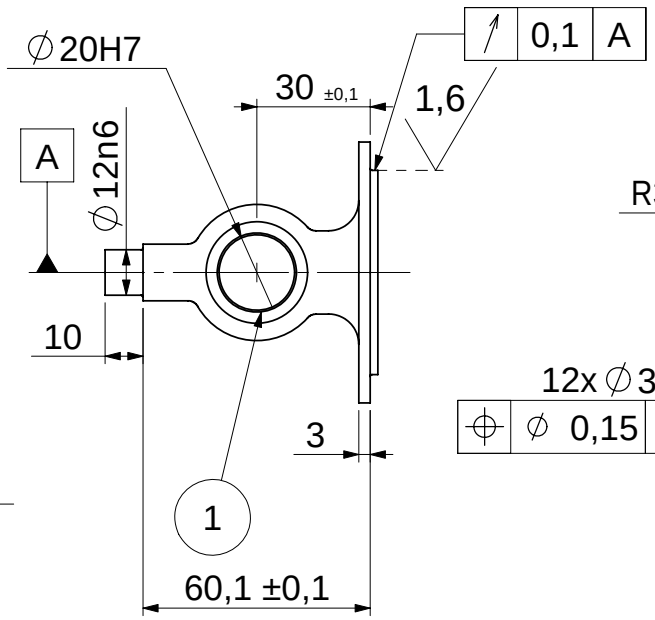


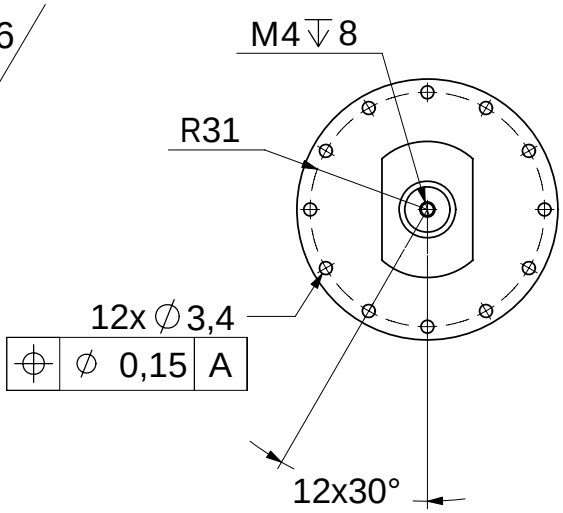
DETAIL B
SCALE 1 : 1











Note:

(1) skærpes min. 0,5x45°/max. 0,8x45° på begge sider

For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Stål:C45

Number: 2 stk

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

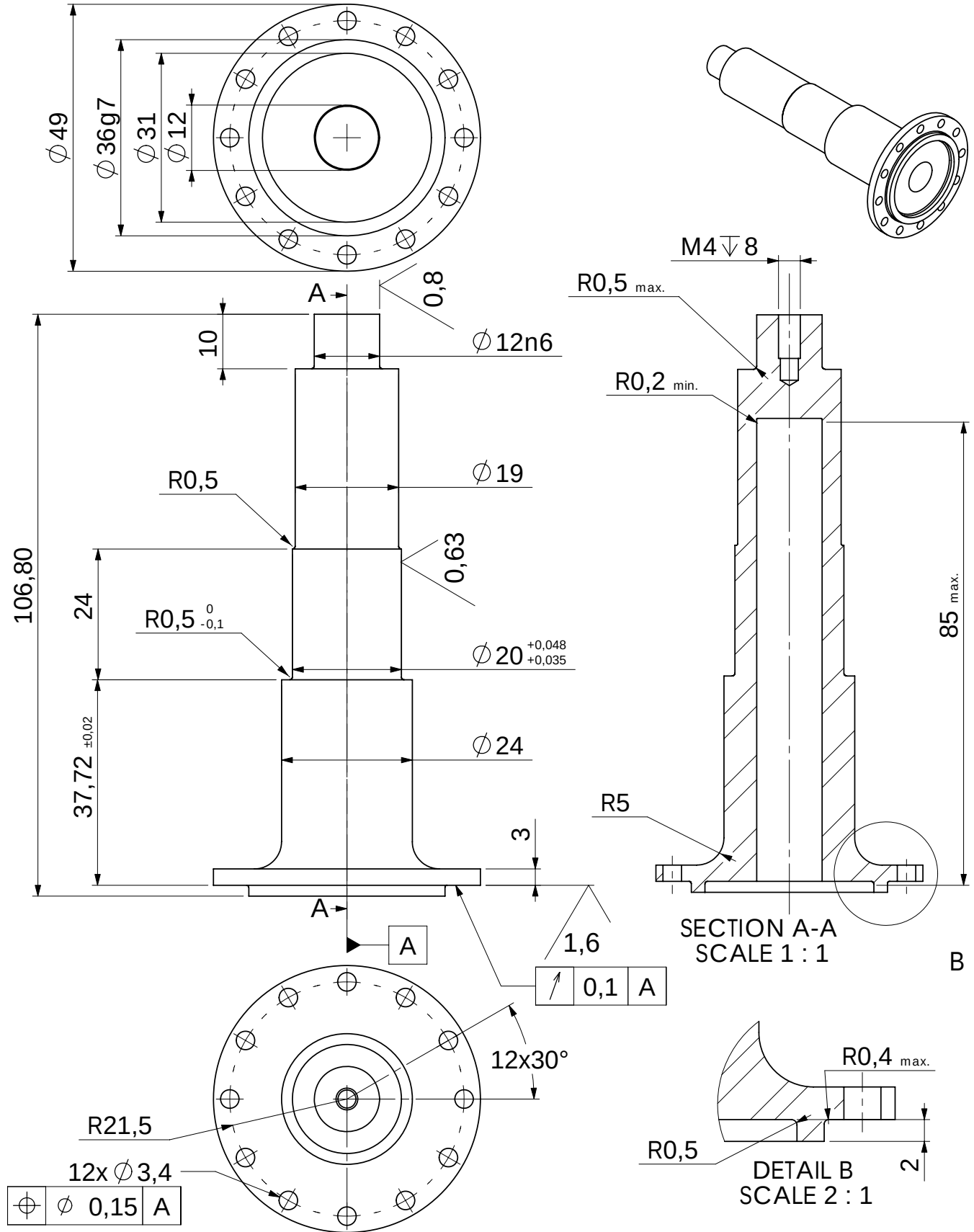
COMMENTS:

4.1 CrossAxle Ankle

Aalborg Universitet
DMS10
Grp.43A, Pon. 103

SIZE A | DWG. NO. 4_1_CrossAxleAnkle | REV.

WEIGHT: | SHEET 1 OF 1



Note:

For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL Stål:C45

Number 2 stk

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

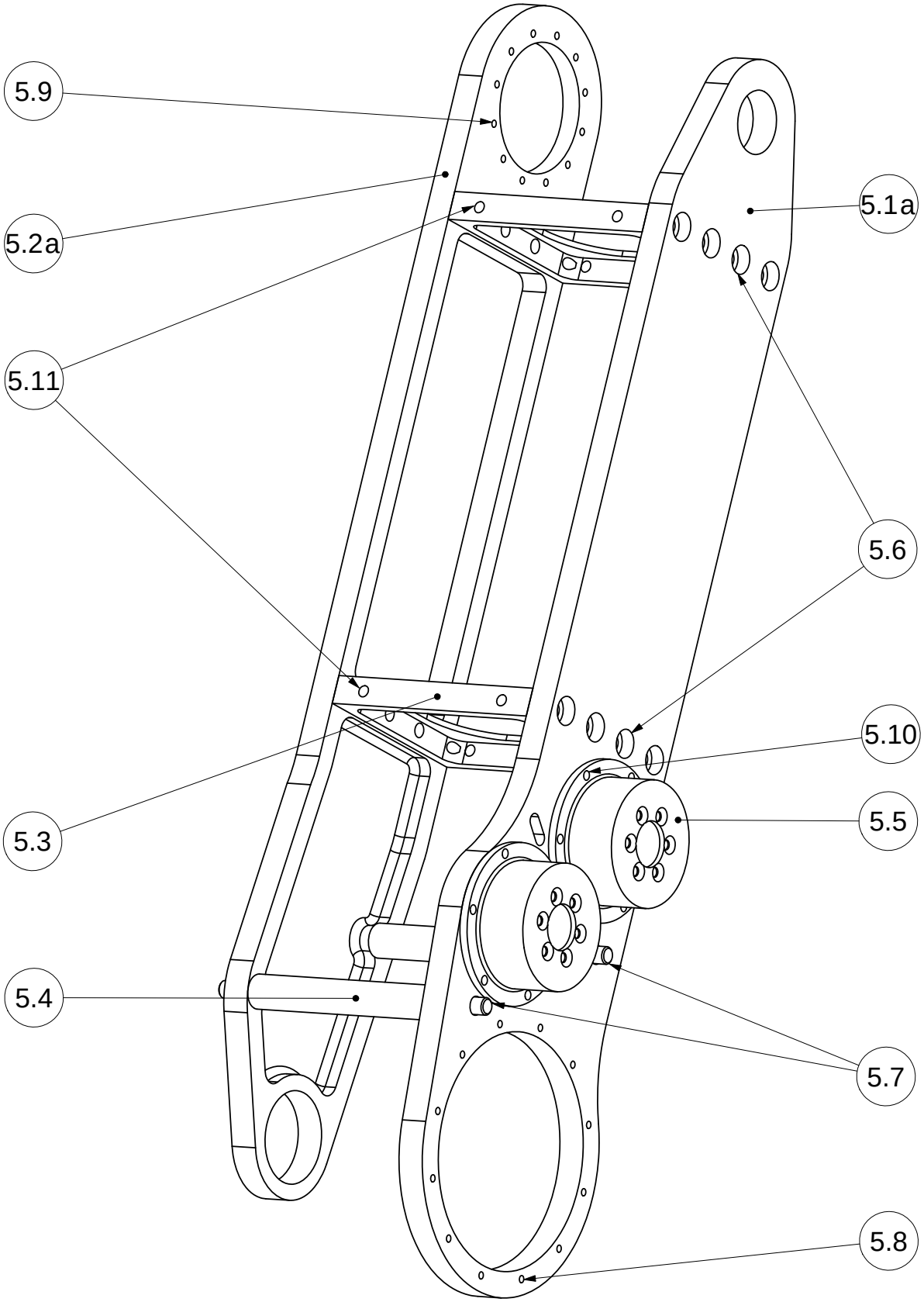
COMMENTS:

4.2 CrossAksle Ankle

Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	4_2_CrossAxleAnkle	

WEIGHT: SHEET 1 OF 1



Note:
Laves i højre/venstre udgaver
højre: 5.1a og 5.2a
venstre: 5.1b og 5.2b

	NAME	DATE
DRAWN		
CHECKED		

MATERIAL

Forventet færdig:

Number 1 højre + 1 venstre

COMMENTS:

SCALE: 1:2

5.A Shin

Aalborg Universitet
DMS10
Grp.43a Pon.103

SIZE	DWG. NO.	REV.
A	Shin_5_A	
	WEIGHT:	SHEET 1 OF 1

Nummer	Betegnelse (*)	Materiale	Antal
5.1a 5.1b	Plade: 10mm	EN AW-6082 T6	1 1
5.2a 5.2b	Plade: 10mm	EN AW-6082 T6	1 1
5.3	Plade: 10mm	EN AW-6082 T6	4
5.4	Aksel: 12mm	Stål	4
5.5	Aksel: 60mm	Al	4
5.6	Unbrako bolt	M5x10	32
5.7	Låsemøtrik	M6	8
5.8	Unbrako bolt	M3x25	24
5.9	Unbrako bolt	M3x15	24
5.10	Unbrako bolt	M3x10	24
5.11	Unbrako bolt	M5x10	24

Note:

* Forslag til dimensioner af råemne.

MATERIAL

Number

	NAME	DATE
DRAWN		
CHECKED		

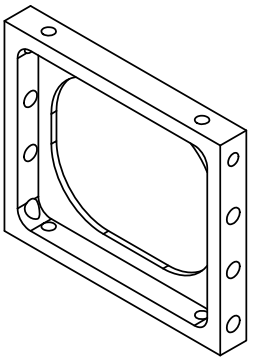
COMMENTS:

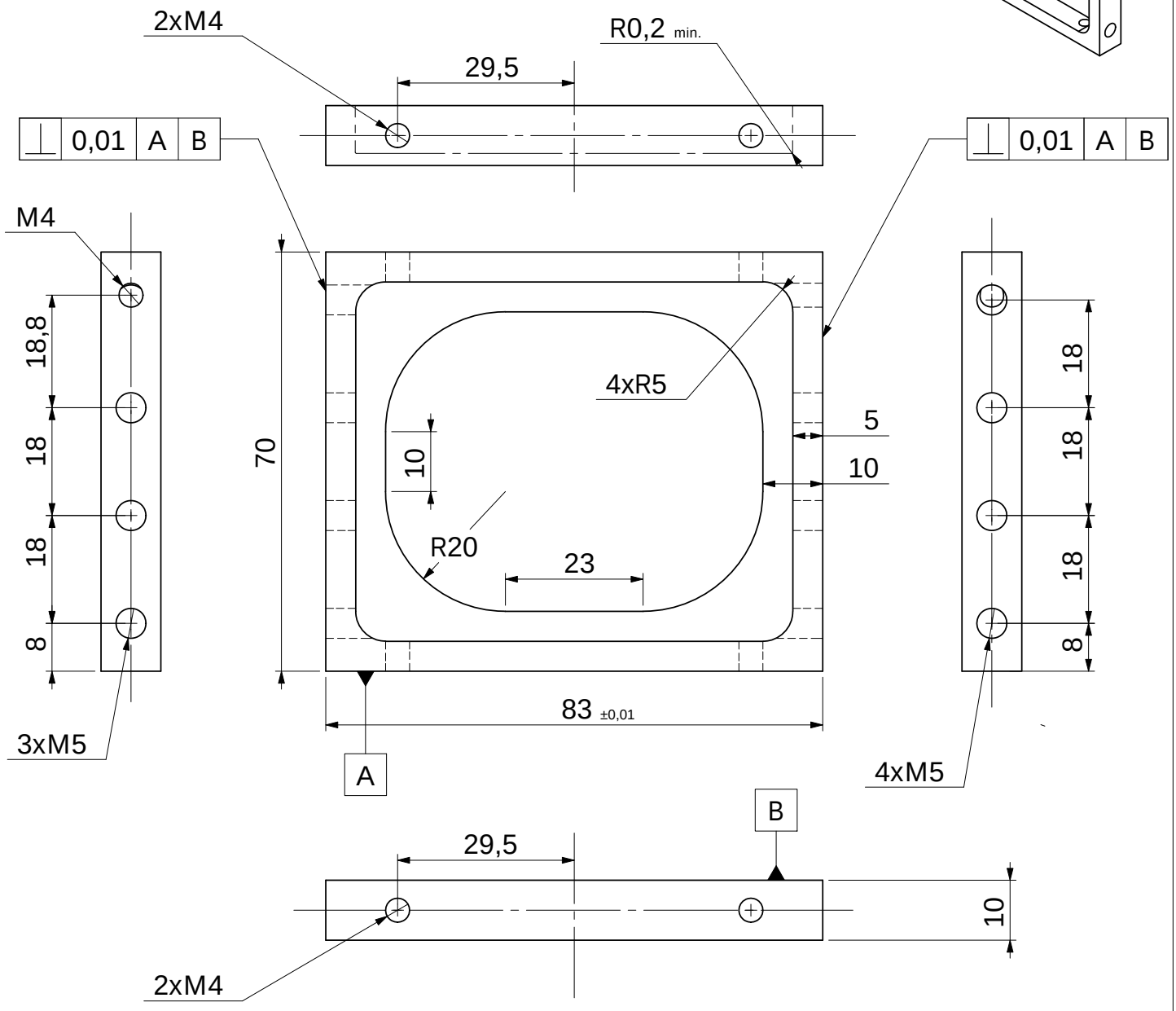
5.0 Shin

Aalborg Universitet
DMS10
Grp.43a Pon.103

SIZE A | DWG. NO. Shin_5_0 | REV.

WEIGHT: | SHEET 1 OF 1





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL
AW-6082 T6

Number
4 stk.

SCALE:1:1

	NAME	DATE
DRAWN	LFC	
CHECKED		

Forventet færdig:

COMMENTS:

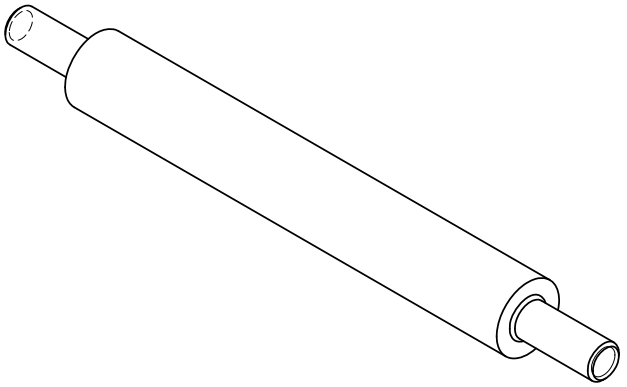
5.3 Shin

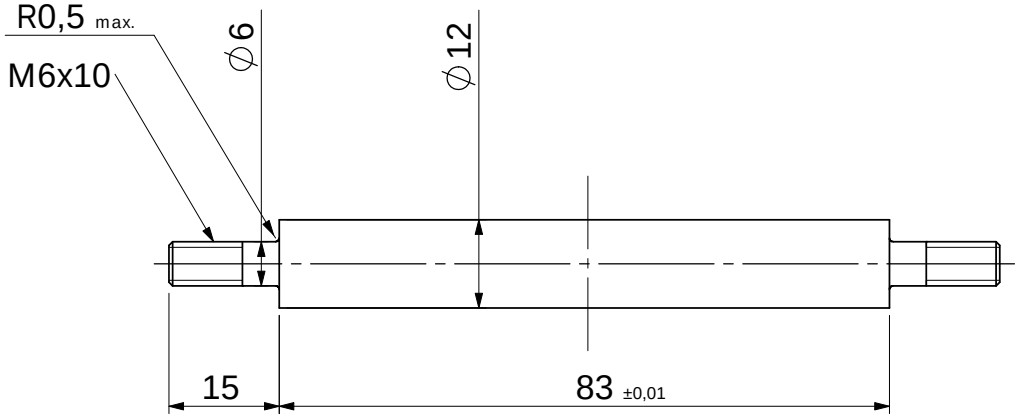
Aalborg Universitet
DMS10
Grp.126

SIZE	DWG. NO.	REV.
A	Shin	

WEIGHT:

SHEET 5 OF 6





Note:

For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Stål

Number: 4 stk.

SCALE:1:1

	NAME	DATE
DRAWN	LFC	
CHECKED		

Forventet færdig:

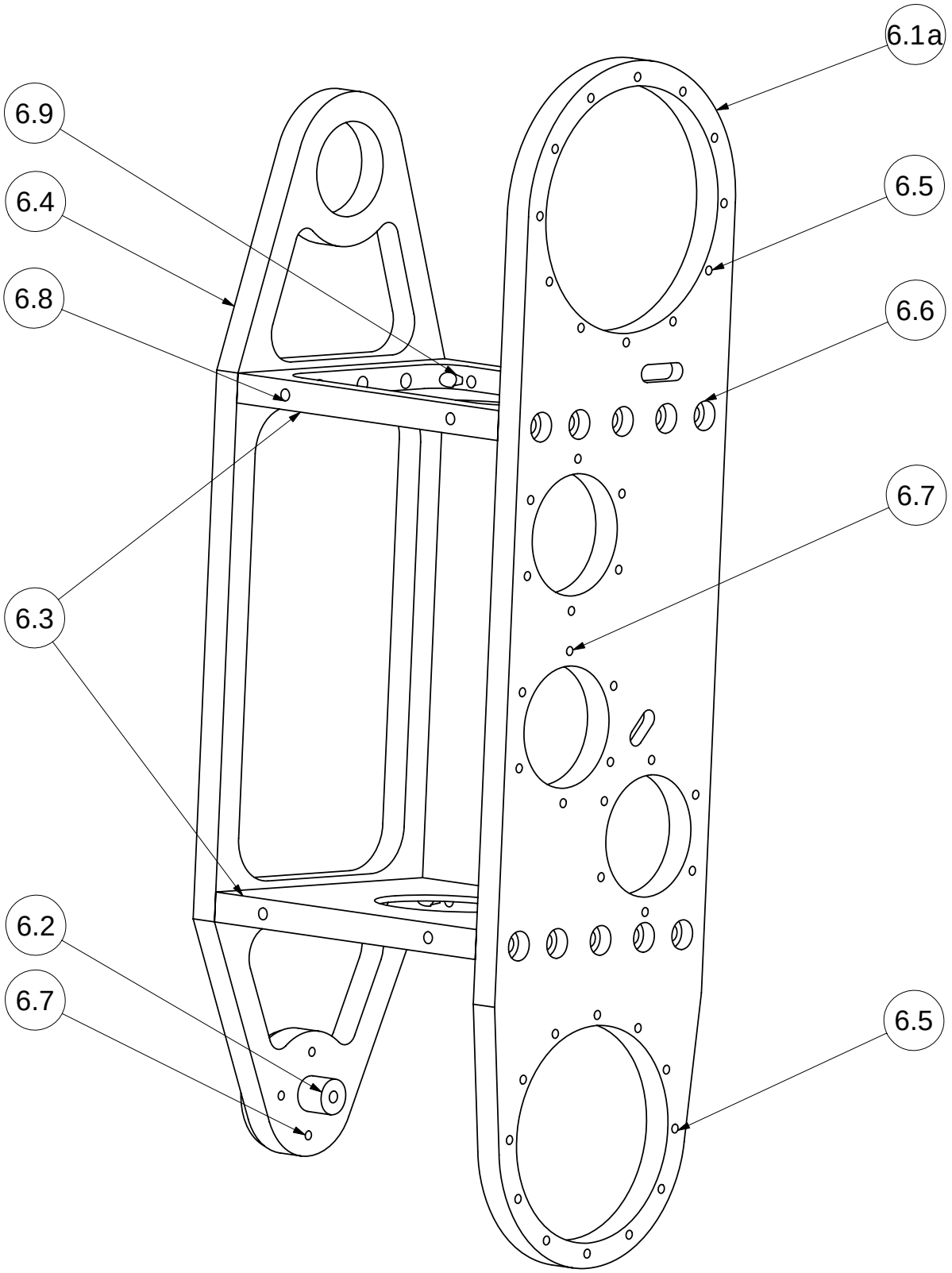
COMMENTS:

5.4 Shin

Aalborg Universitet
DMS10
Grp.126

SIZE	DWG. NO.	REV.
A	Shin	

WEIGHT: | SHEET 6 OF 6



Note:	DIMENSIONS ARE IN mm TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	NAME / DATE DRAWN CHECKED Forventet færdig:	6.A Thigh Aalborg Universitet DMS10 Grp.43a Pon. 103
	MATERIAL	COMMENTS: 1 højre og 1 venstre	
	Number 2 stk. SCALE:1:2		SIZE A · DWG. NO. Thigh · REV. WEIGHT: · SHEET 1 OF 1

Nummer	Betegnelse (*)	Materiale	Antal
6.1a 6.1b	Plade 10mm	EN AW-6082 T6	1 1
6.2	Aksel Ø40mm	Stål	2
6.3	Plade 10mm	EN AW-6082 T6	4
6.4	Plade 10mm	EN AW-6082 T6	2
6.5	Umbracho skrue	M3x25	48
6.6	Umbracho skrue	M5x15	32
6.7	Umbracho skrue	M3x12	8
6.8	Umbracho skrue	M4x12	16
6.9	Umbracho skrue	M5x20	8

Note:
* Forslag til materialevalg

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

Number

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

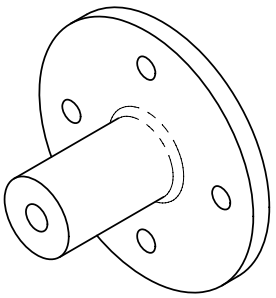
COMMENTS:

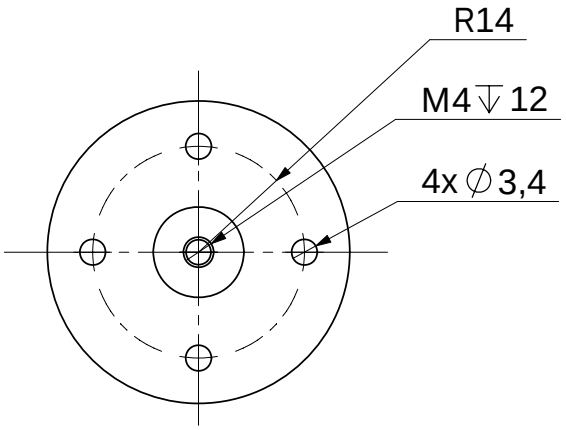
6.0 Thigh

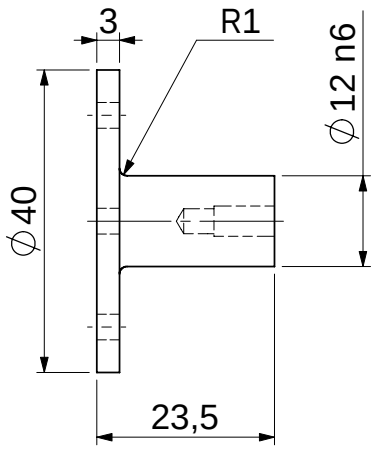
Aalborg Universitet
DMS10
Grp.43a Pon. 103

SIZE A | DWG. NO. 6_0_Thigh | REV.

WEIGHT: | SHEET 1 OF 1







Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN INmm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Stål

Number: 2 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

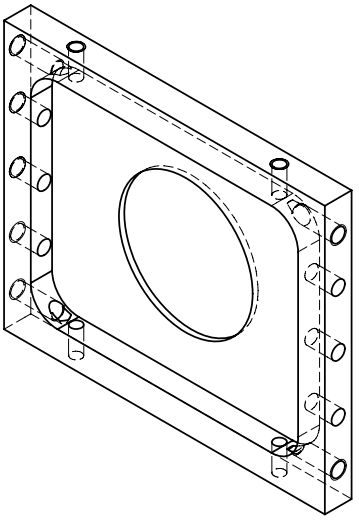
6.2 Thigh

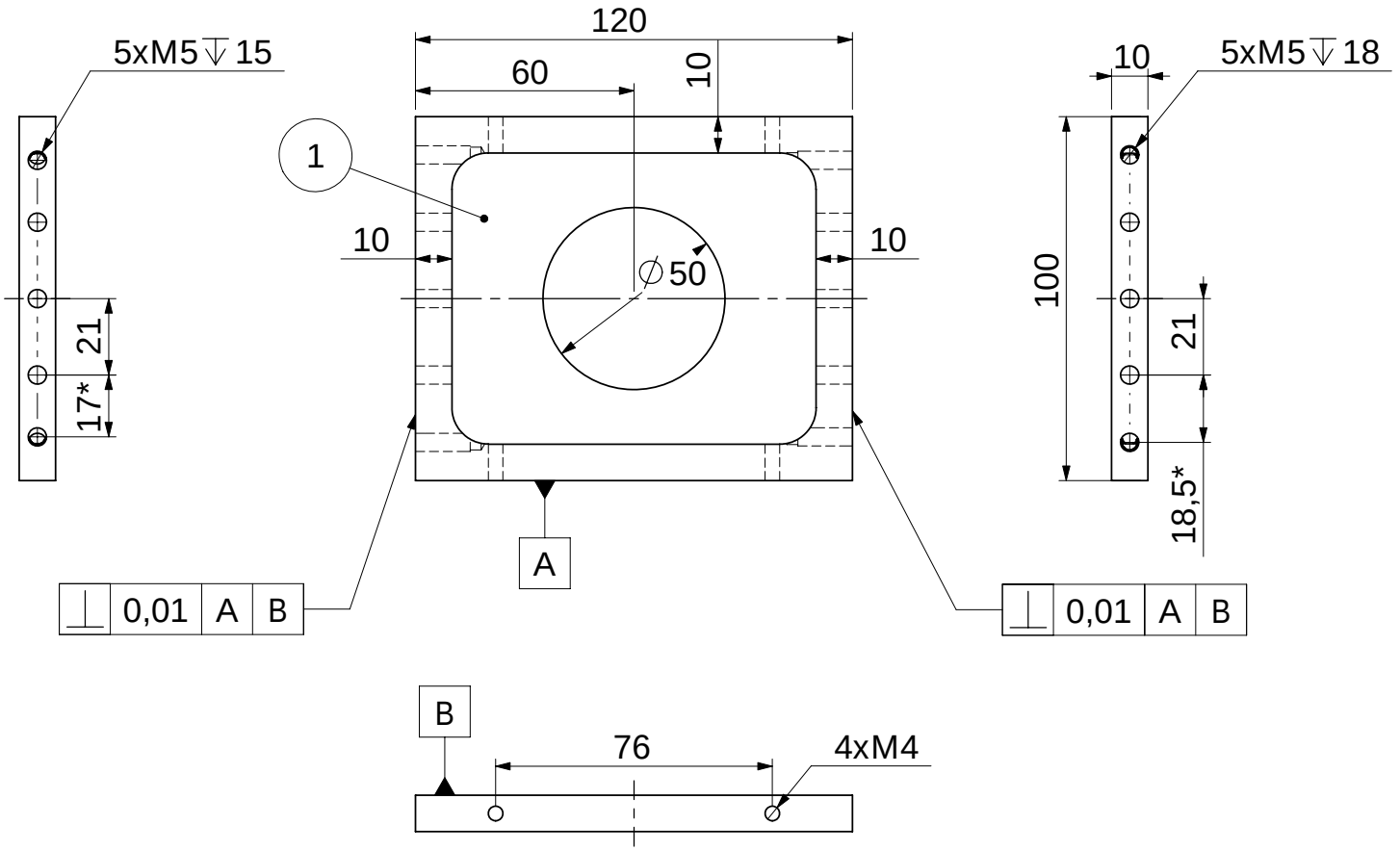
Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	6_2_FlangeAxle	

WEIGHT:

SHEET 1 OF 1





Note: For ikke tolerance specificeret mål. Benyttes DS/ISO 2768-m

(1) Udfræsning
dybde 8mm
indvendig radius 10mm
bund radius 2,5mm
Fræses efter TransverseLegPlate_6_3.iges

* Bemærk der er forskel på hul-afstanden fra side til side

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 4 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

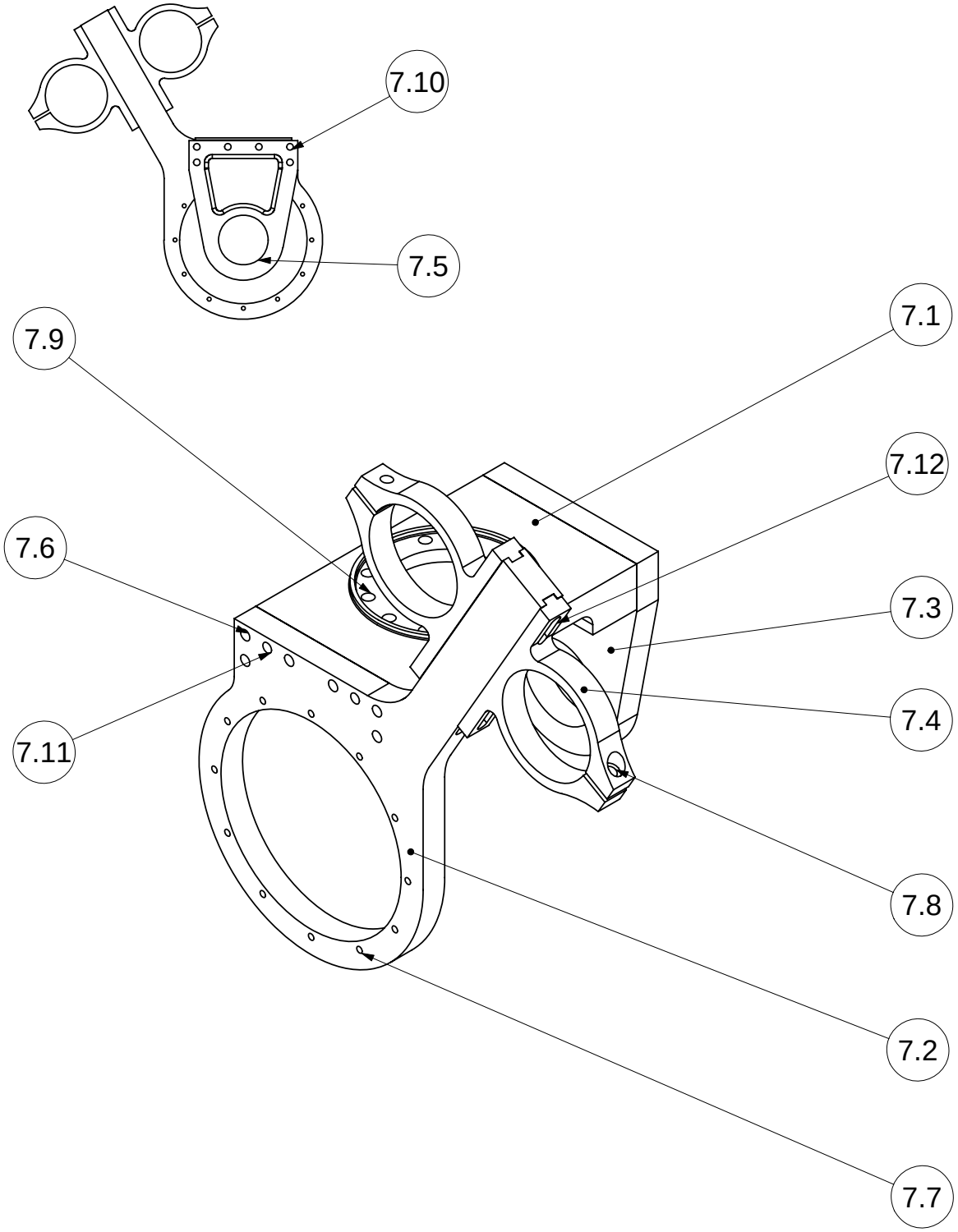
COMMENTS:

6.3 Thigh

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE A | DWG. NO. 6_3_TransverseLegPlate | REV.

WEIGHT: | SHEET 1 OF 1



Note:		NAME / DATE	7.A HipBracket
		DRAWN	Aalborg Universitet DMS10 Grp.43A, Pon. 103
		CHECKED	
		Forventet færdig:	
	MATERIAL	COMMENTS:	
	Number 2 stk		SIZE A | DWG. NO. 7_A_HipBracket | REV.
	SCALE:1:2		WEIGHT: | SHEET 1 OF 1

Nummer	Betegnelse	Materiale	Antal
7.1	Plade: 20mm*	EN AW-6082 T6	2
7.2	Plade:10mm*	EN AW-6082 T6	2
7.3	Plade:10mm*	EN AW-6082 T6	2
7.4	Plade:10mm*	EN AW-6082 T6	2
7.5	UnBracko skue	M4x12	16
7.6	Unbrako skrue	M5x20	12
7.7	Unbrako skrue	M3x35	24
7.8	Unbrako skrue	M4x15	4
7.9	Unbrako skrue	M4x15	24
7.10	Unbrako skrue	M5x20	12
7.11	Cylinderstift	Ø4x16	4

Note:

(*) Forslag til dimension på råemne

	NAME	DATE
DRAWN		
CHECKED		

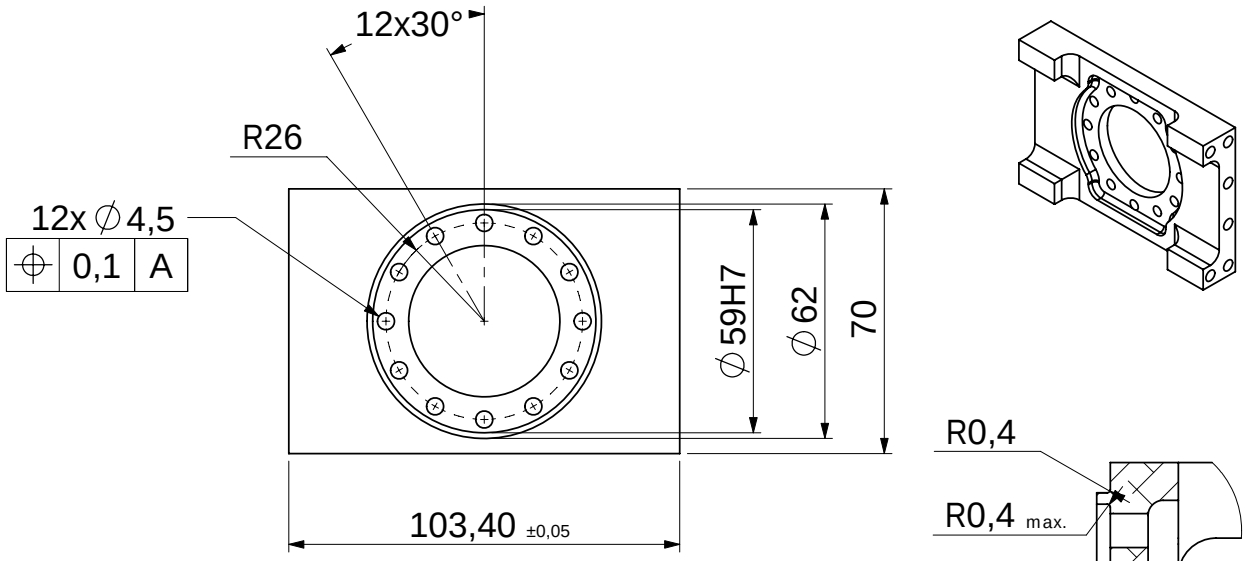
Forventet færdig:

COMMENTS:

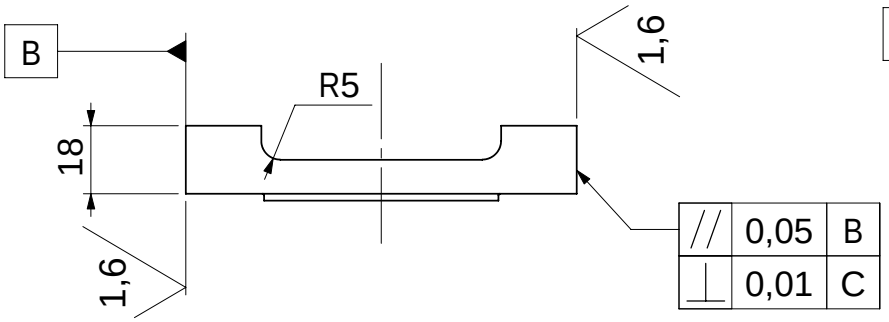
7.0 HipBracket

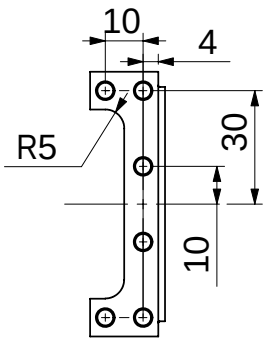
Aalborg Universitet
DMS10
Grp.43A, Pon 103

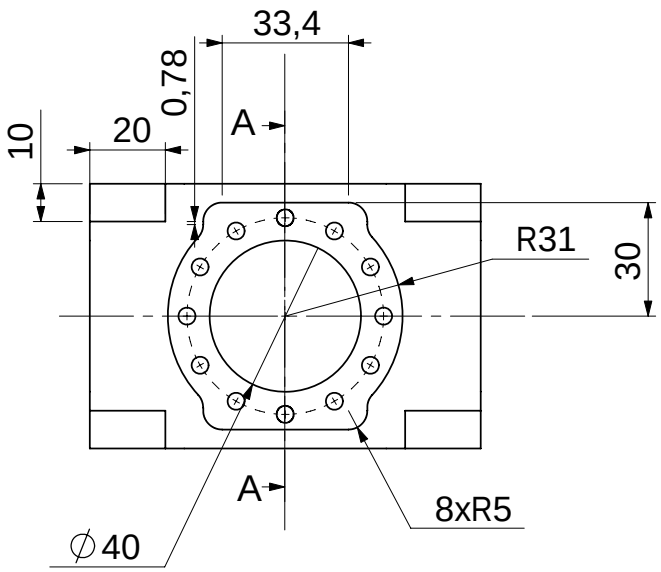
SIZE	DWG. NO.	REV.
A	7_0_HipBracket	
	WEIGHT:	SHEET 1 OF 1

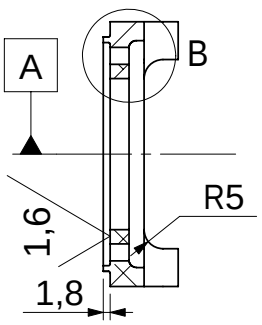


DETAIL B
SCALE 1 : 1









SECTION A-A

Note:

For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 2 stk

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

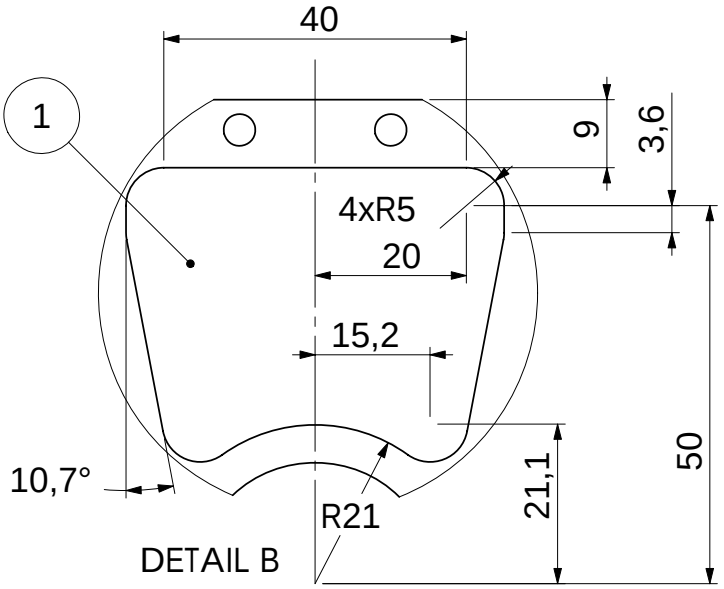
COMMENTS:

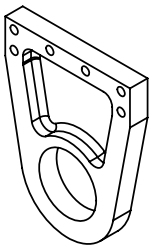
7.1 HipBracket

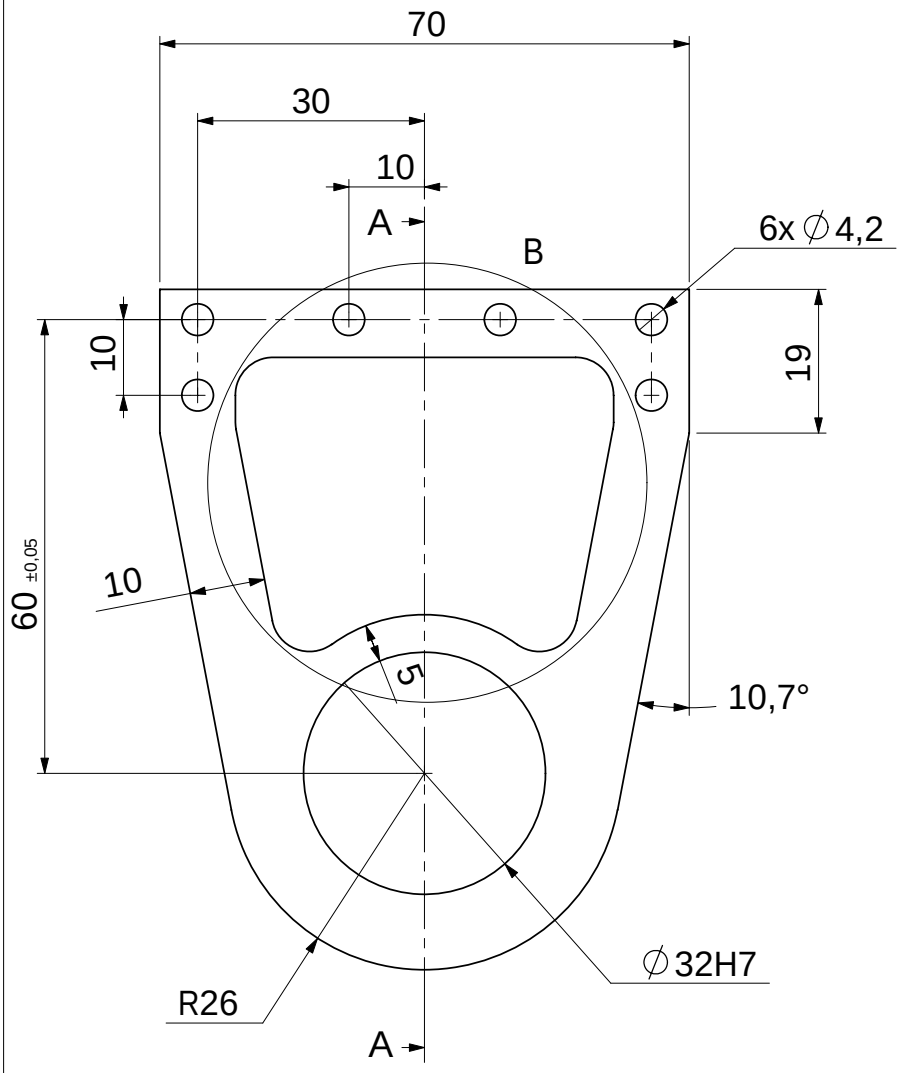
Aalborg Universitet
DMS10
Grp.43A, Pon. 103

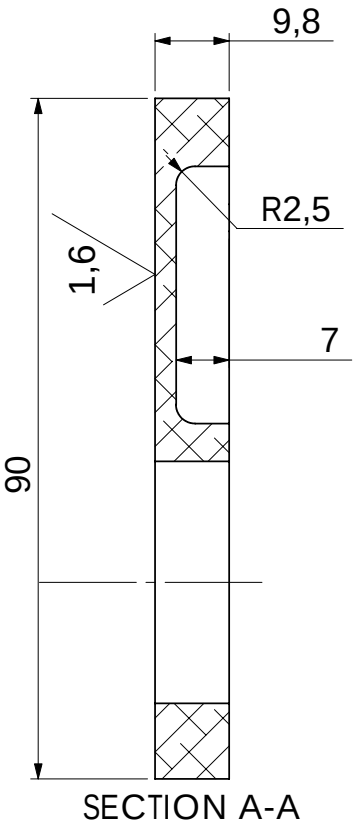
SIZE	DWG. NO.	REV.
A	7_1_HipBracket	

WEIGHT: SHEET 1 OF 1









Note:

For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

(1) Fræses efter igs-fil: 7_3_HipBracket.igs

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 2 stk

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

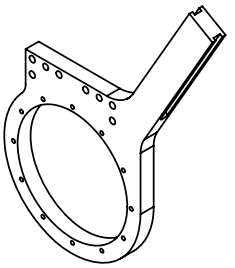
COMMENTS:

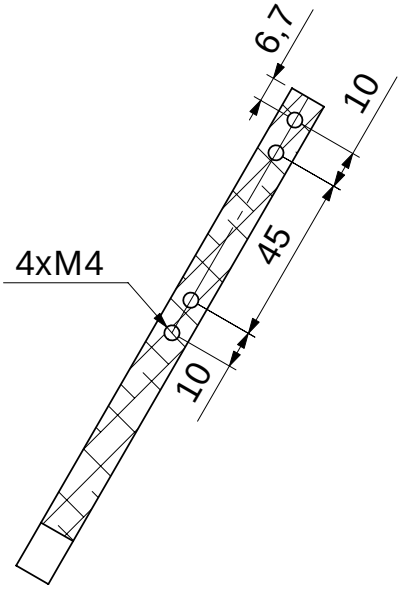
7.3 HipBracket

Aalborg Universitet
DMS10
Grp.126

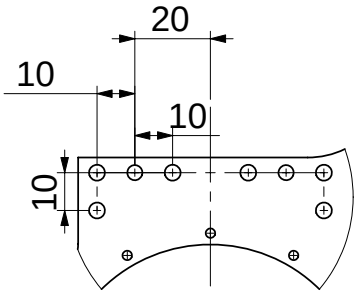
SIZE	DWG. NO.	REV.
A	7_3_HipBracket	

WEIGHT: SHEET 1 OF 1

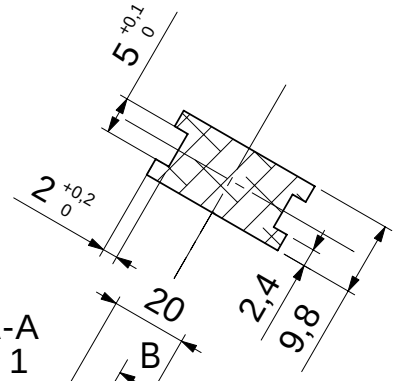




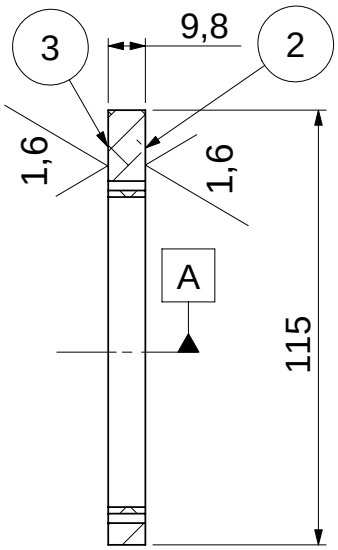
SECTION B-B



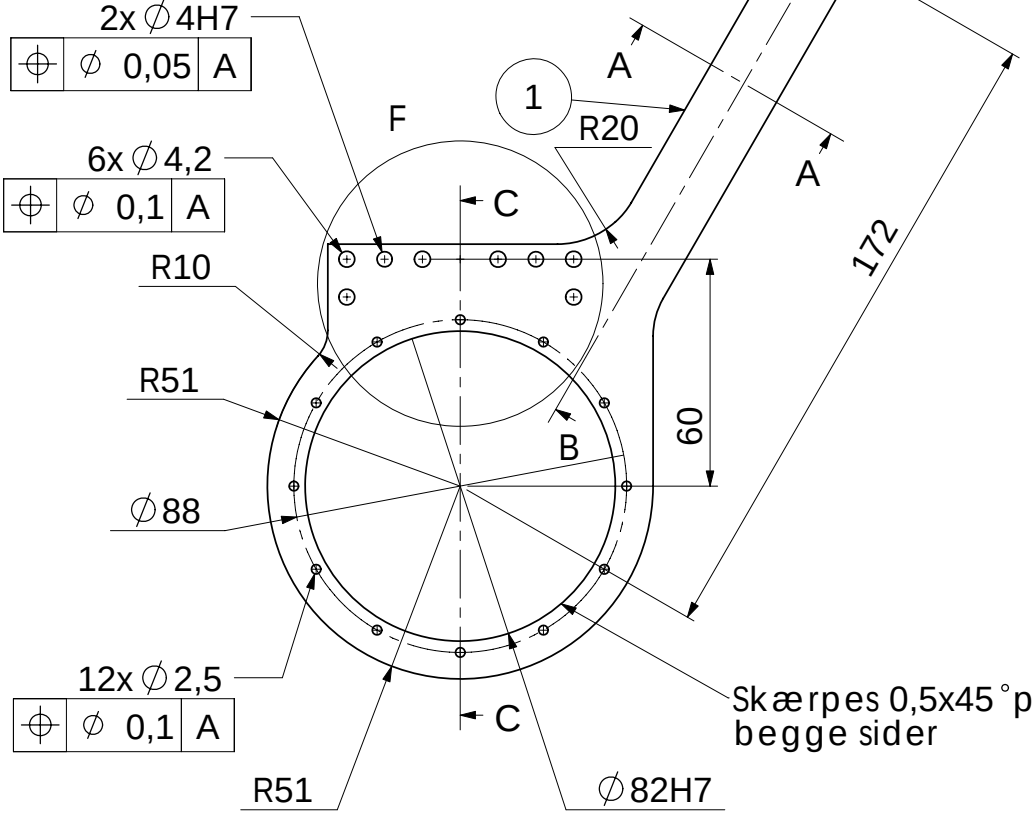
DETAIL F



SECTION A-A
SCALE 1 : 1



SECTION C-C



Note:

For ikke tolerance specificeret mål benyttes DS/ISO 2768-m
(1) bearbejdes efter geometri -fil: 7_2_HipBracket.igs
(2) Ruhed gælder kun for (a)
(3) Ruhed gælder kun for (b)

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 2 stk*

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

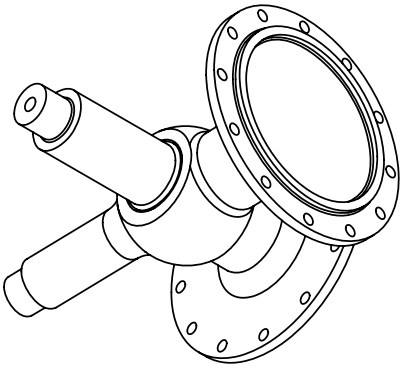
(*)
1 stk højre(a)
1 stk venstre(b)

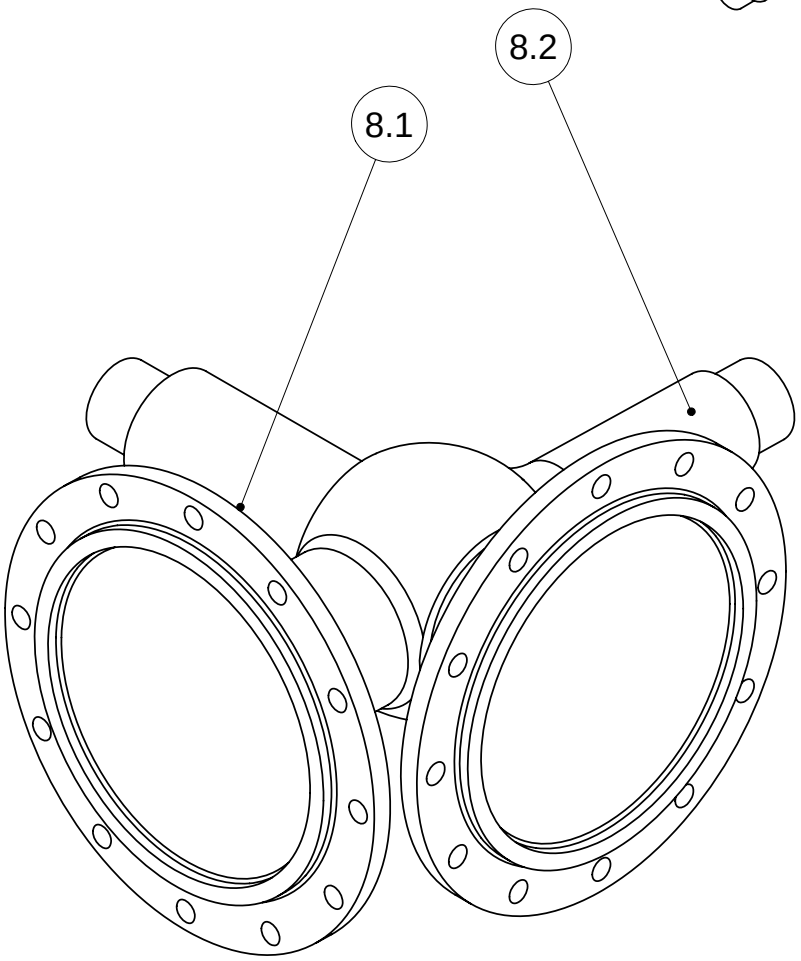
7.2 HipBracket

Aalborg Universitet
DMS10
Grp.43A, Pon 103

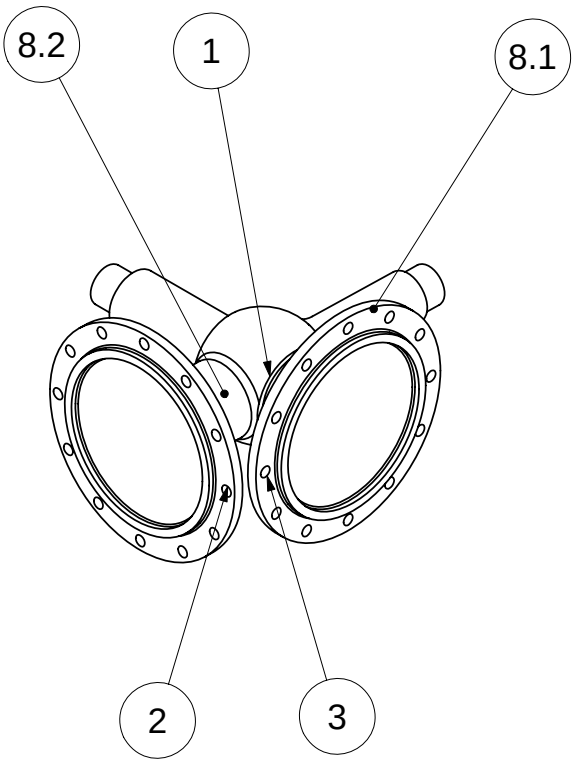
SIZE	DWG. NO.	REV.
A	7_2_HipBracket	

WEIGHT: SHEET 1 OF 1





Note:	DIMENSIONS ARE IN mm. TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±		NAME / DATE DRAWN CHECKED Forventet færdig: COMMENTS:
	MATERIAL Number 1 stk. SCALE:1:1		8.A CrossAxleHip Aalborg Universitet DMS10 Grp.43A, Pon 103 SIZE A · DWG. NO. 8_A_CrossAxleHip · REV. WEIGHT: · SHEET 1 OF 1



Note:

Krympning af CrossAxle Hip (8.1)opvarmes til 270 °C, hvorefter aksel (8.2) anlægges mod bryst(1). Hullerne (2) og (3) skal ikke passe over for hinanden.

NB. akslen skal først samles nå gearene til robotten er modtaget.

	NAME	DATE
DRAWN		
CHECKED		

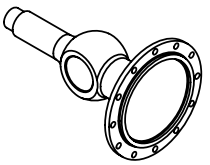
Forventet færdig:

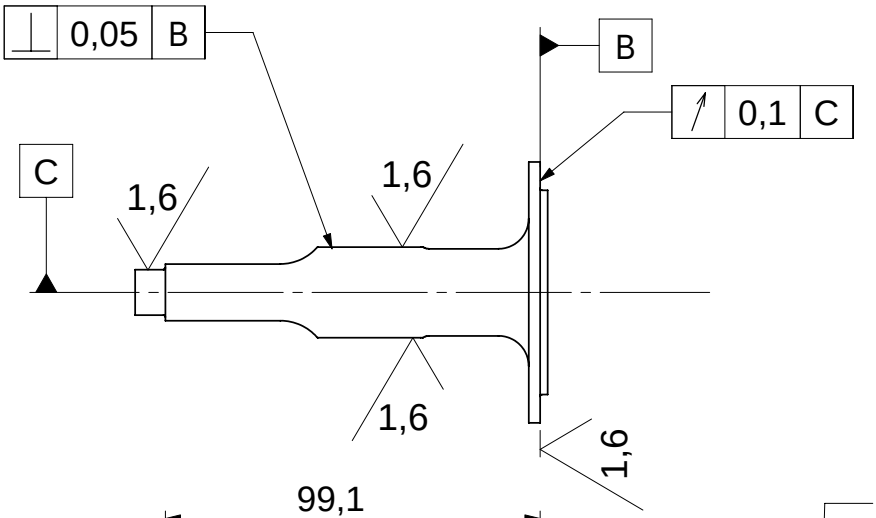
COMMENTS:

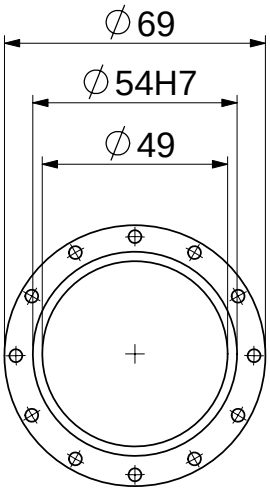
8.B CrossAxle Hip

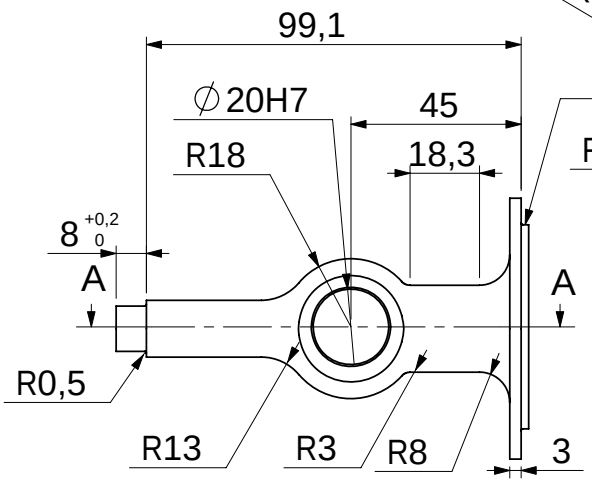
Aalborg Universitet
DMS10
Grp.43A, Pon 103

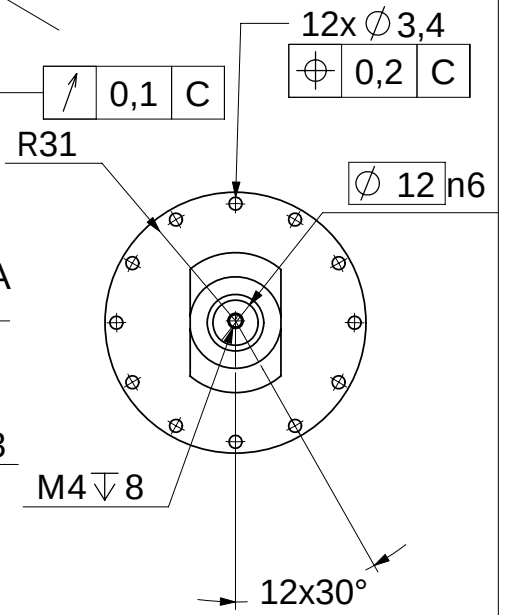
SIZE	DWG. NO.	REV.
A	8_B-CrossAxleHip	
	WEIGHT:	SHEET 1 OF 1

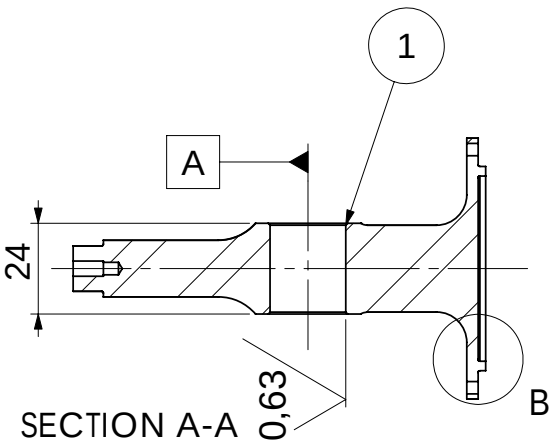




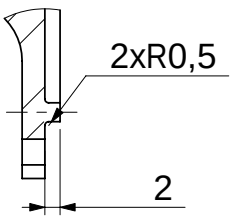








SECTION A-A



DETAIL B
SCALE 1 : 1

Note:
For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

(1) Skærpes min. 0,5x45° / max. 0,8x45° på begge sider

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: stål: C45

Number: 2 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

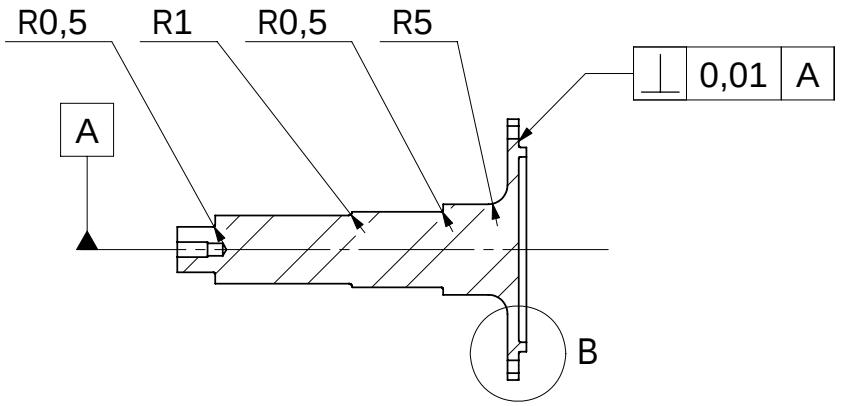
COMMENTS:

8.1 CrossAxleHip

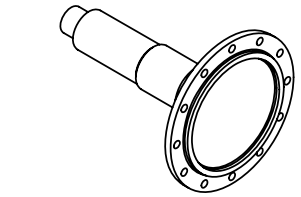
Aalborg Universitet
DMS10
Grp.43A, Pon 103

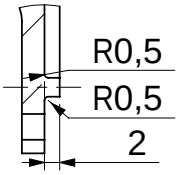
SIZE	DWG. NO.	REV.
A	8_1_CrossAxleHip	

WEIGHT: SHEET 1 OF 1

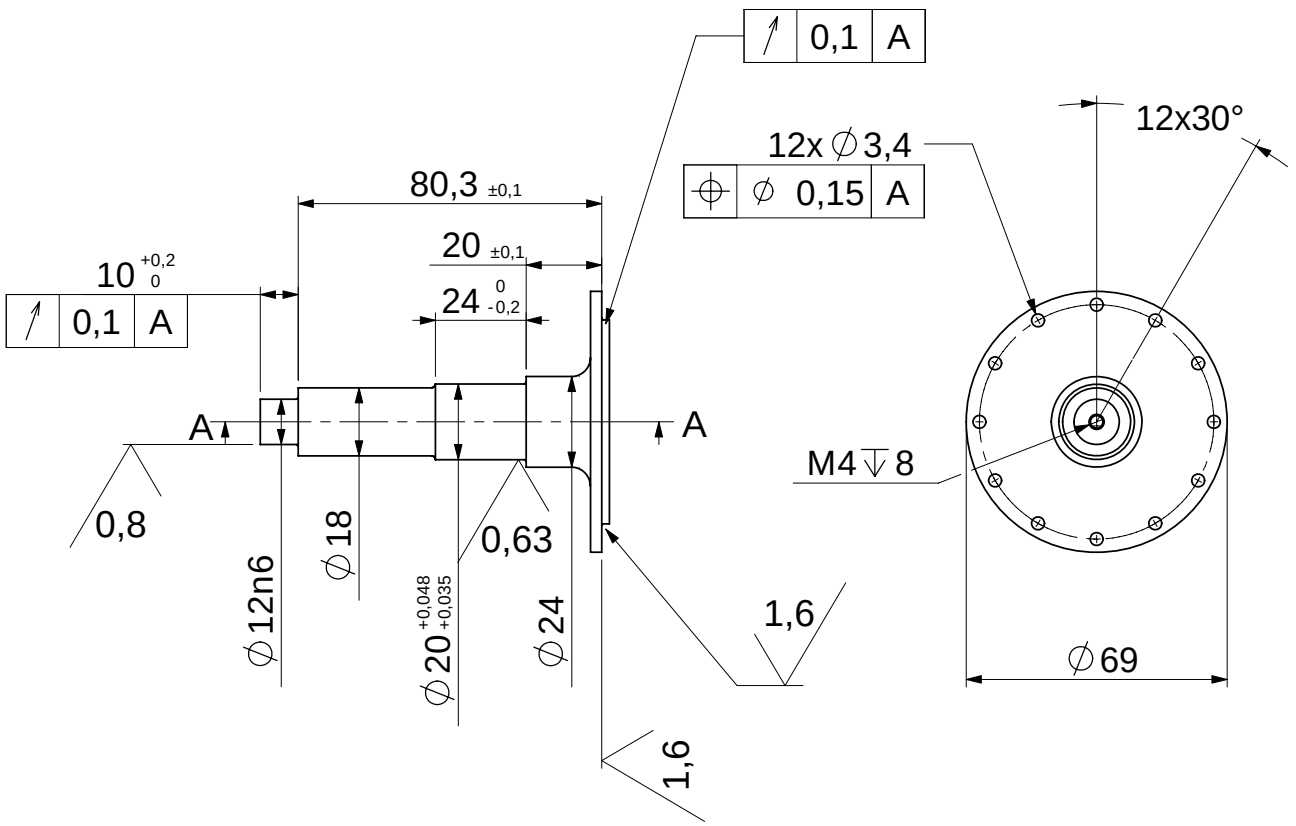


SECTION A-A





DETAIL B
SCALE 1 : 1



Note:
For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm.
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Stål: C45

Number: 2 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

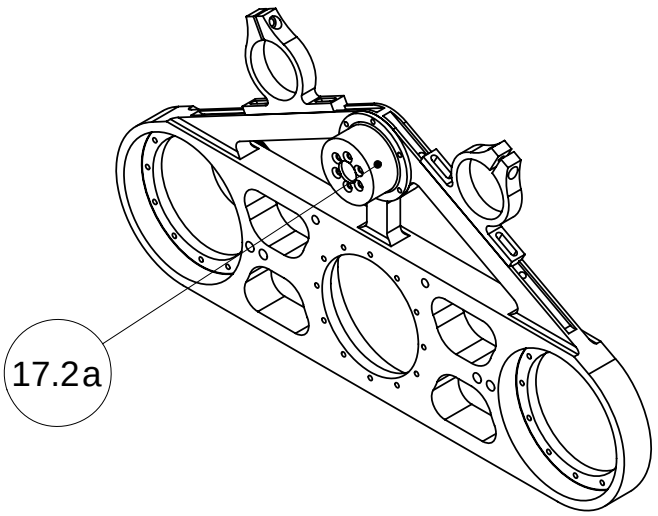
8.2 CrossAxleHip

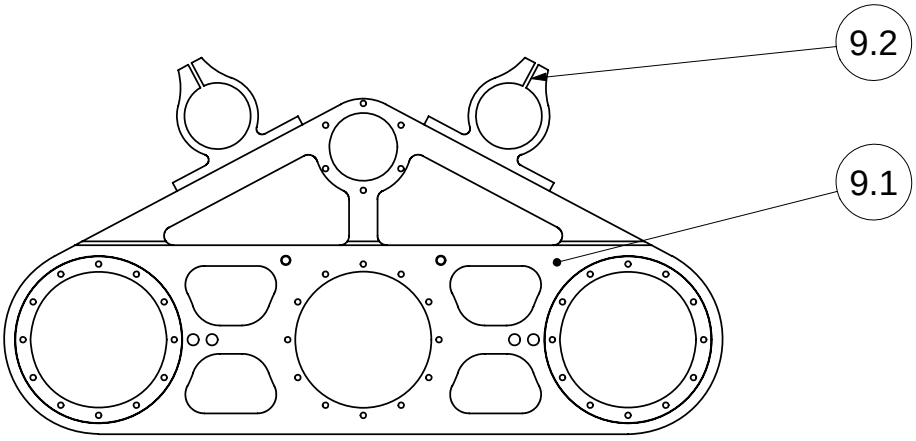
Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	8_2_CrossAxleHip	

WEIGHT:

SHEET 1 OF 1



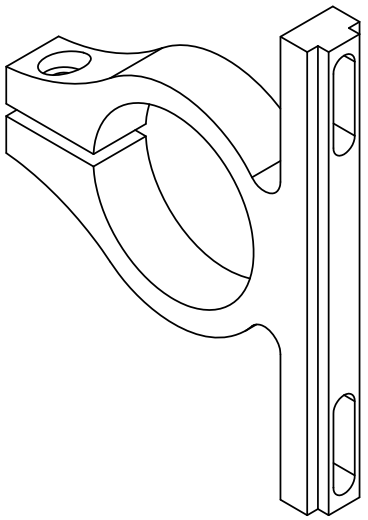


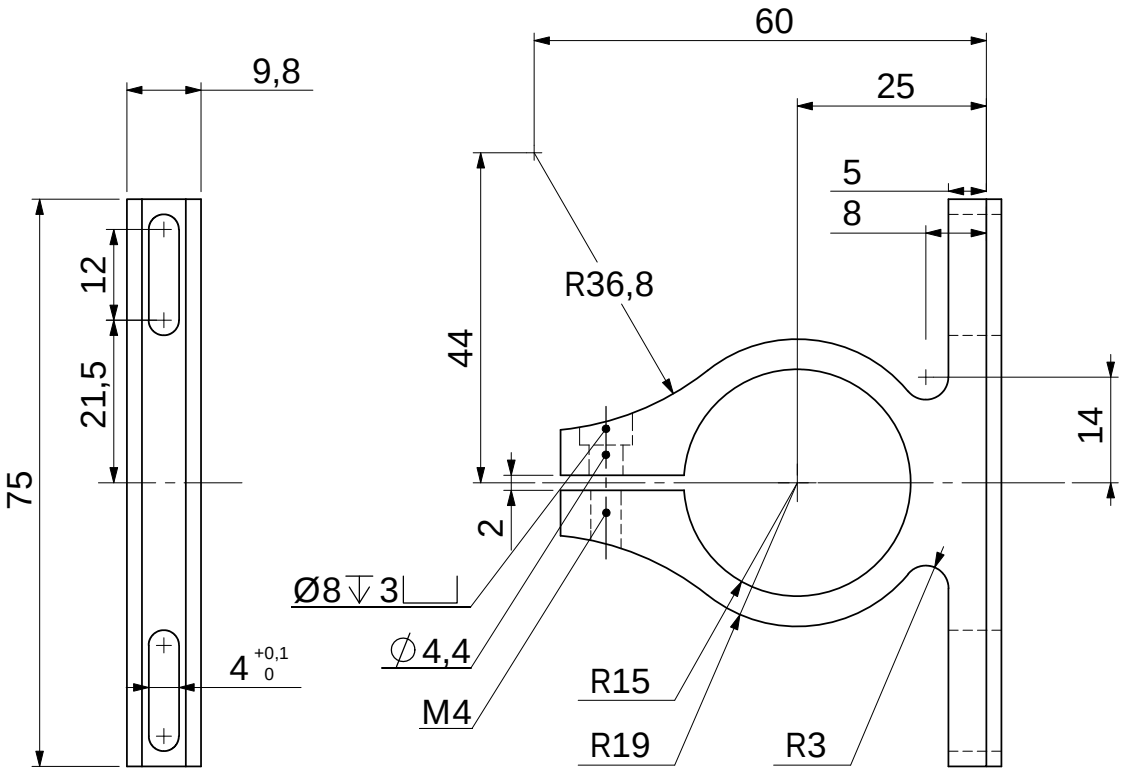
Note:			NAME	DATE
		DRAWN		
		CHECKED		
		Forventet færdig:		
	MATERIAL	COMMENTS:		
	Number 1 stk.			
	SCALE:1:4			

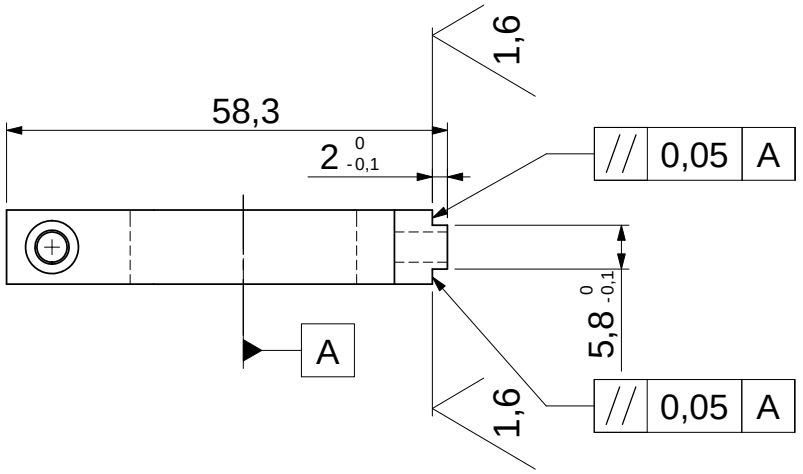
9.A Pelvis

Aalborg Universitet
DMS10
Grp.43A, Pon 103

SIZE	DWG. NO.	REV.
A	9_A_Pelvis	
	WEIGHT:	SHEET 1 OF 1







Note:
For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

	NAME	DATE
DRAWN	LFC	
CHECKED		

Forventet færdig:

COMMENTS:

MATERIAL Al 6082-T6

Number 2 stk.

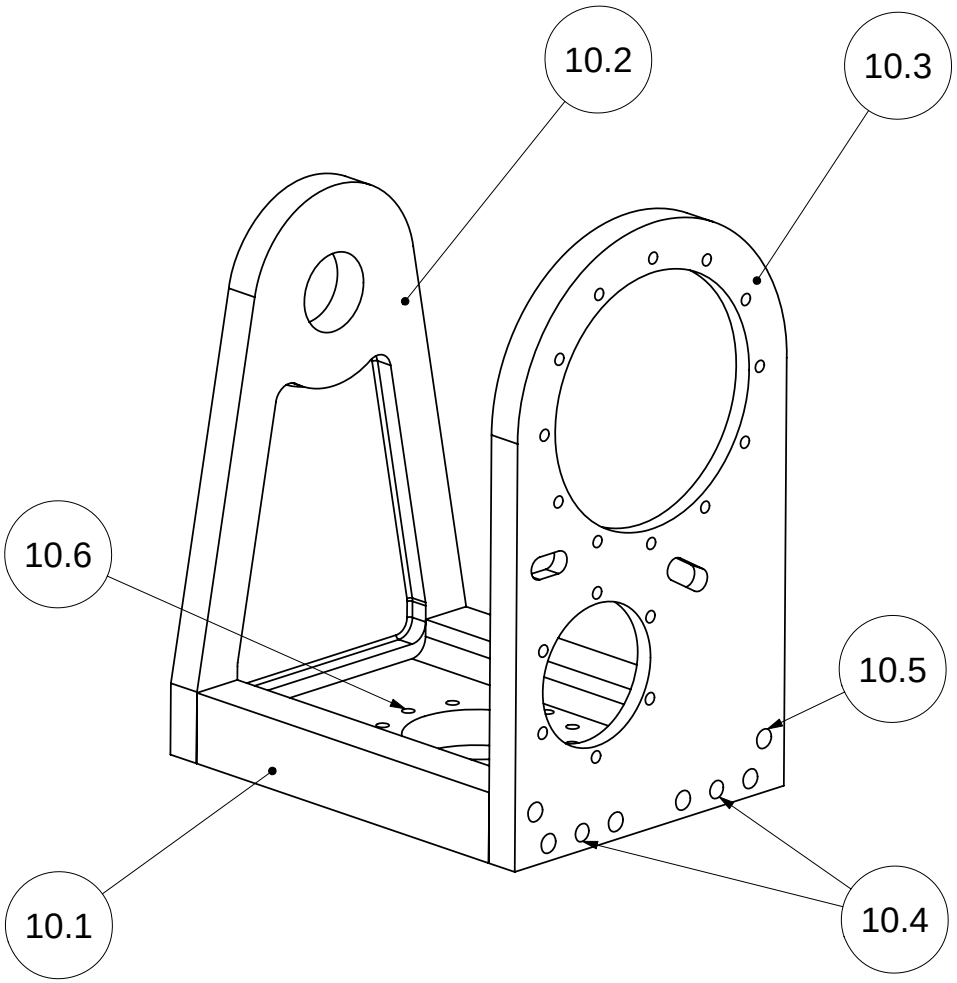
SCALE: 1:1

9.2 Pelvis

Aalborg Universitet
DMS10
Grp.43a Pon.103

SIZE	DWG. NO.	REV.
A	9_2_Pelvis	

WEIGHT: SHEET 1 OF 1



Note:	DIMENSIONS ARE IN mm TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± ±		NAME	DATE	10.A Waist Pitch Joint
		DRAWN			Aalborg Universitet DMS10
		CHECKED			Grp.43a, Pon103.
	MATERIAL	COMMENTS:			A WaistPitchJoint REV.
	Number 1 stk				WEIGHT: SHEET 1 OF 1

Nummer	Betegnelse (*)	Materiale	Antal
10.1	Plade 20mm	EN AW-6082 T6	1
10.2	Plade 10mm	EN AW-6082 T6	1
10.3	Plade 10mm	EN AW-6082 T6	1
10.4	Stift	Ø5x20	2
10.5	Unbrako skrue	M5x20	12
10.6	Unbrako skrue	M3x20	12

Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

Number

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

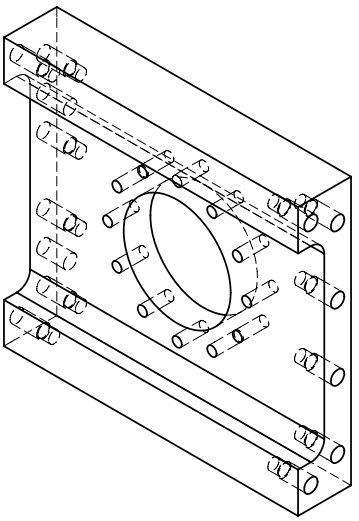
COMMENTS:

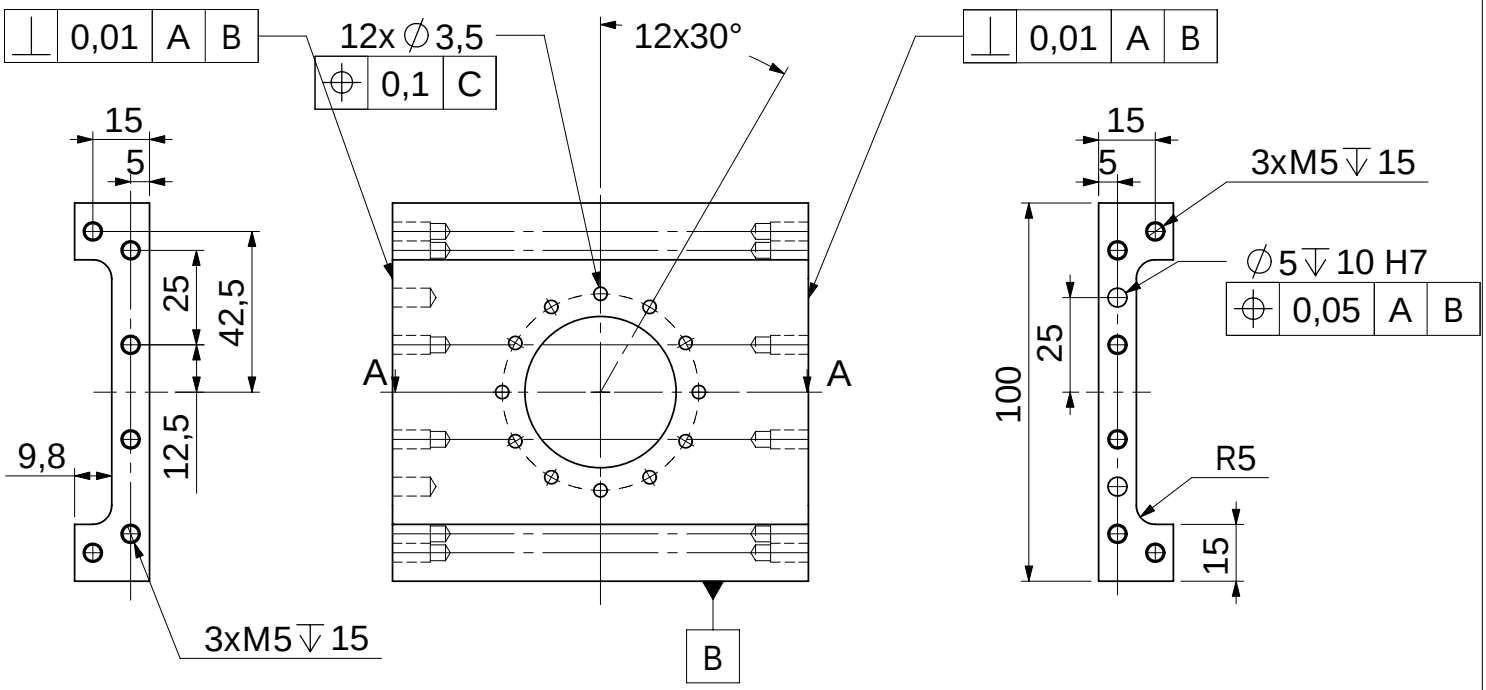
10.0 Waist Pitch

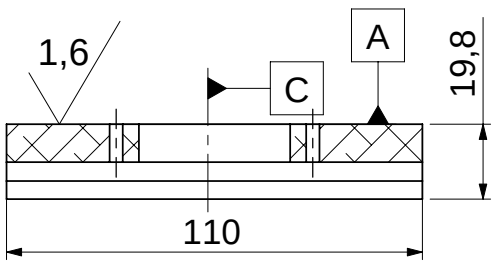
Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE A | DWG. NO. 10_0_WaistPitch | REV.

WEIGHT: | SHEET 1 OF 1







SECTION A-A

Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number 1 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

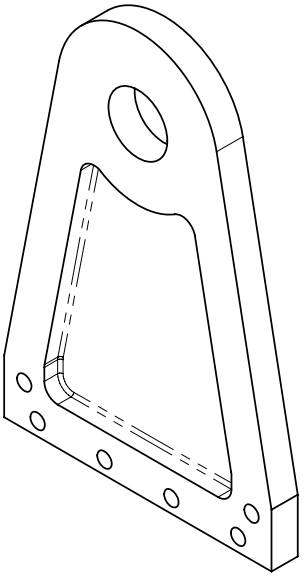
10.1 Waist Pitch

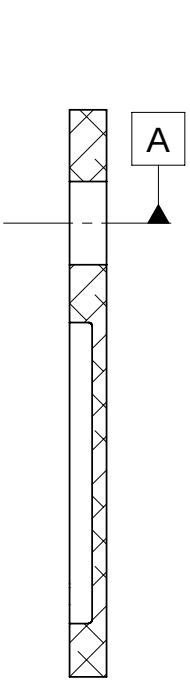
Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE A | DWG. NO. 10_1_Waistpitchbottom | REV.

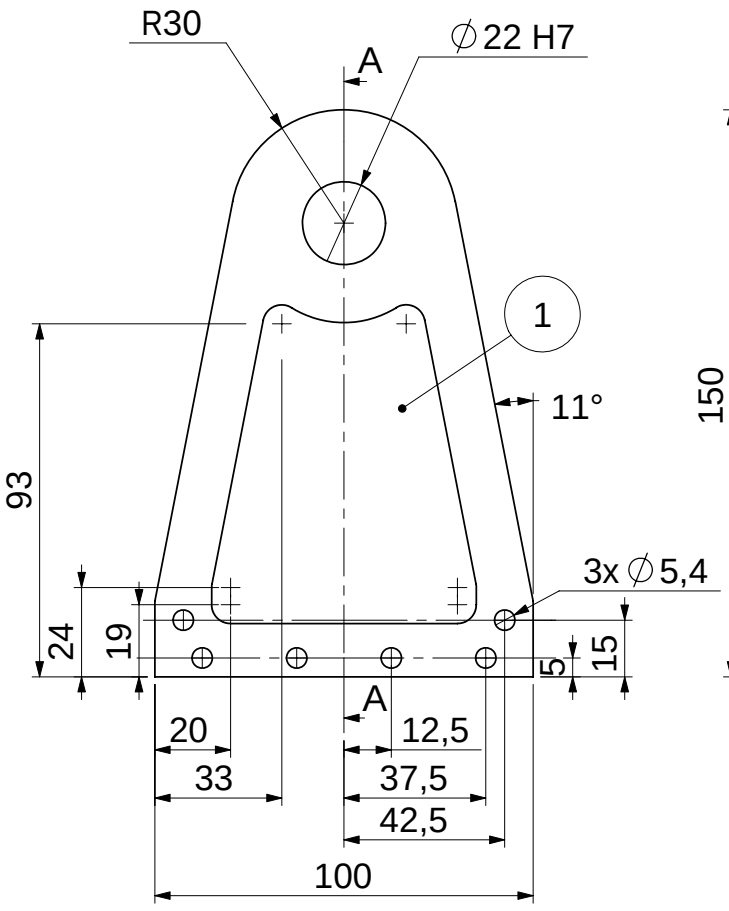
WEIGHT:

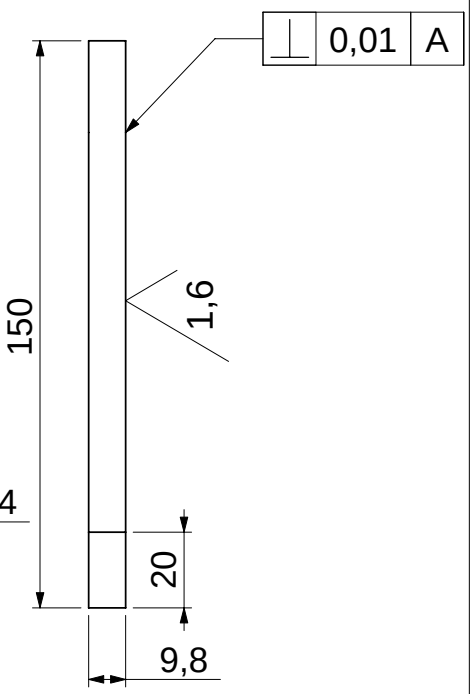
SHEET 1 OF 1





SECTION A-A





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

(1) Udfræsning
dybde 6mm
Indvendig radius 5mm
bund radius 2,5mm
Fræses efter WaistPitch_10_2.iges

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number 1 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

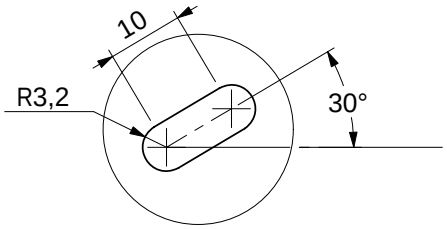
COMMENTS:

10.2 Waist Pitch

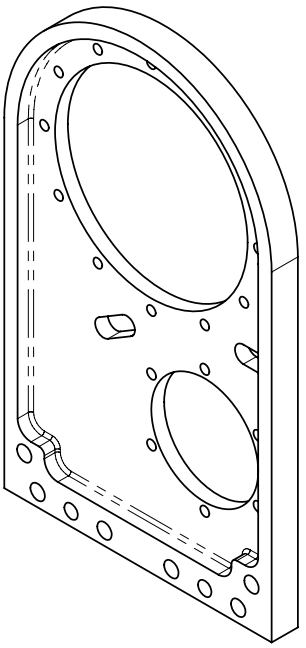
Aalborg Universitet
DMS10
Grp.43a, Pon. 103

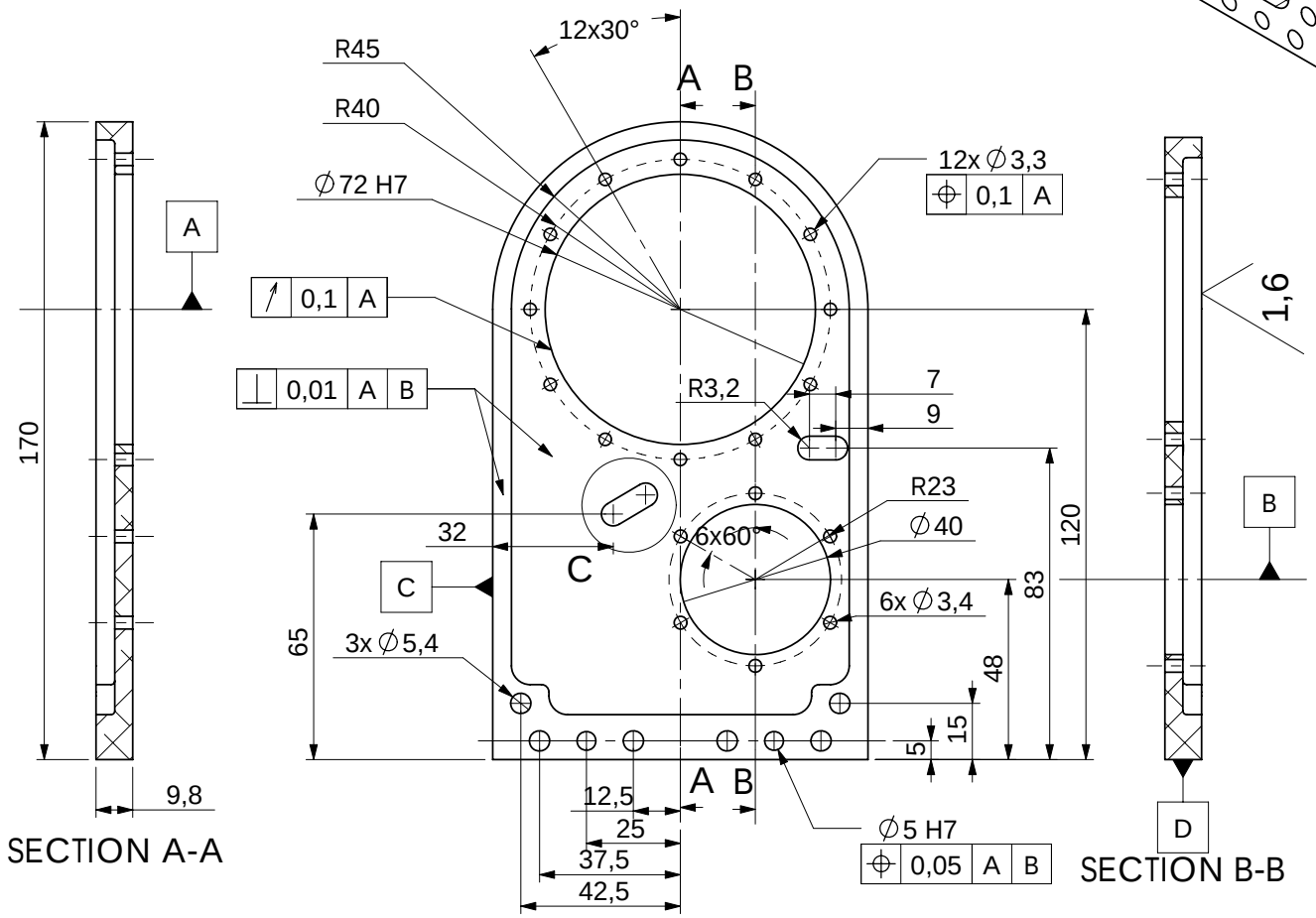
SIZE	DWG. NO.	REV.
A	10_2_WaistPitch	

WEIGHT: SHEET 1 OF 1



DETAIL C
SCALE 1 : 1





SECTION A-A

SECTION B-B

Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

Fræses efter WaistPitch_10_3.iges

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 1 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

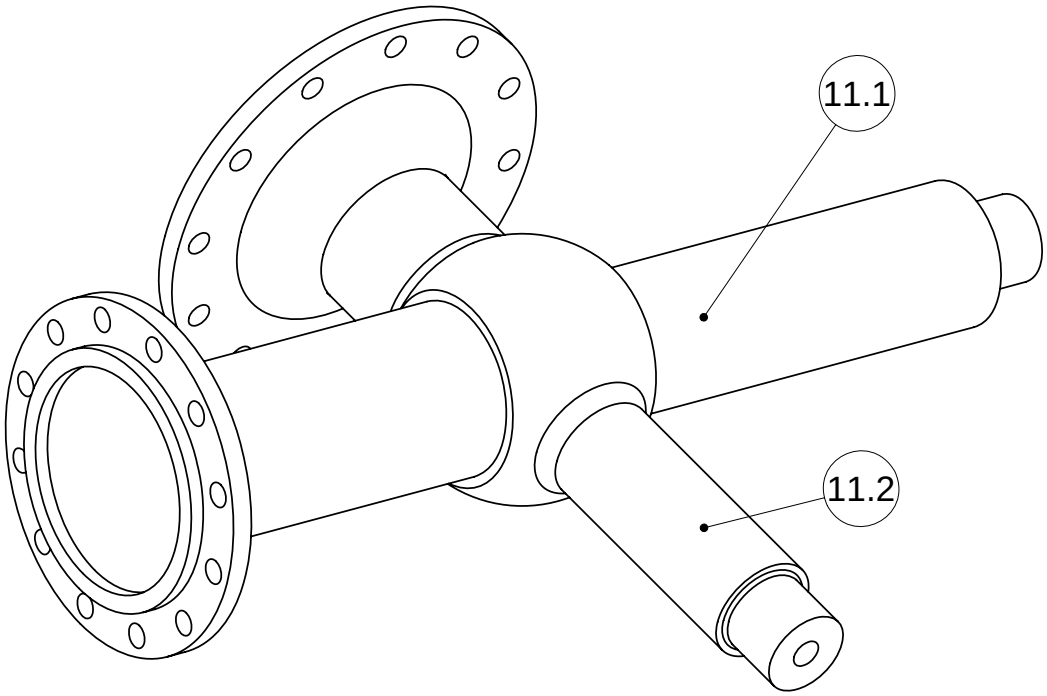
Forventet færdig:

COMMENTS:

10.3 Waist Pitch

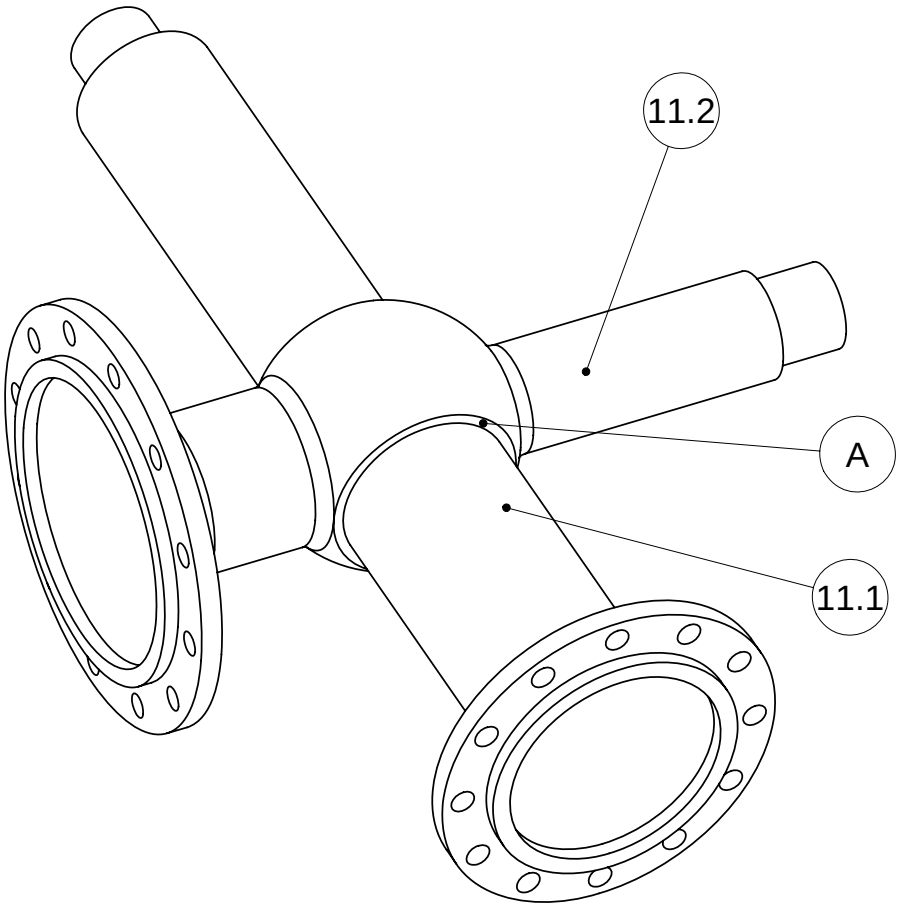
Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	10_3_Waist pitch	
	WEIGHT:	SHEET 1 OF 1



Nummer	Betegnelse*	Materiale	Antal
11.1	CrossAxle Waist (roll)	Stål: C45	1
11.2	CrossAxle Waist (pitch)	Stål: C45	1

Note:		NAME / DATE	11.A CrossAxle Waist
		DRAWN	Aalborg Universitet DMS10 Grp.43a Pon.103
		CHECKED	
		Forventet færdig:	
	MATERIAL Stål: C45	COMMENTS:	
	Number 1 stk.		SIZE A / DWG. NO. CrossAxleWaist / REV.
	SCALE:1:2		WEIGHT: / SHEET 1 OF 4



Note:
Krympes sammen.
11.2 opvarmes til 270° og anlægges mod brystet A på 11.1.

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Stål: C45

Number: 1 stk.

SCALE: 1:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

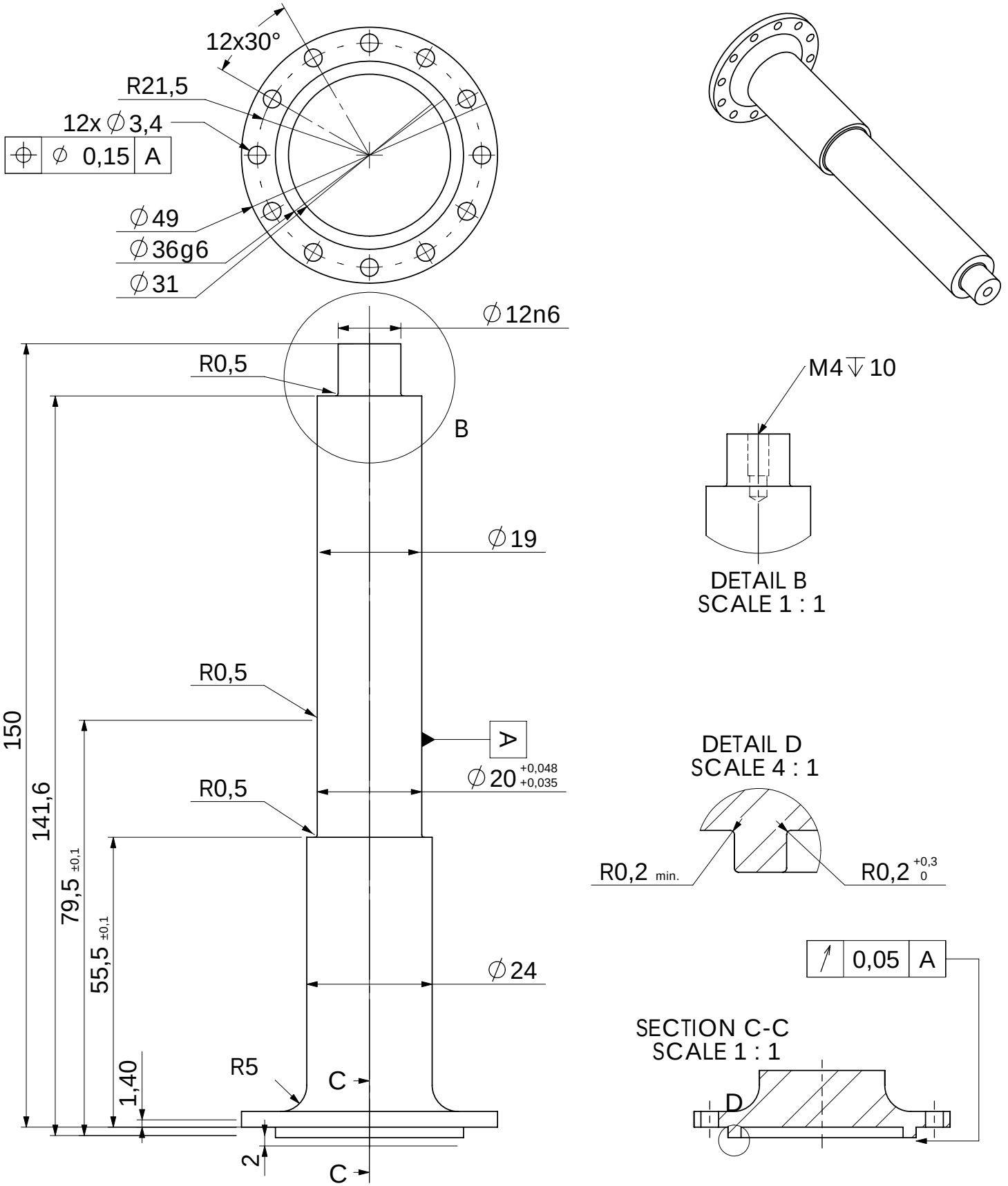
11.B CrossAxle Waist

Aalborg Universitet
DMS10
Grp.126

SIZE	DWG. NO.	REV.
A	CrossAxleWaist	

WEIGHT:

SHEET 2 OF 4



Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Stål: C45

Number: 1 stk.

SCALE: 1:1

	NAME	DATE
DRAWN	LFC	
CHECKED		

Forventet færdig:

COMMENTS:

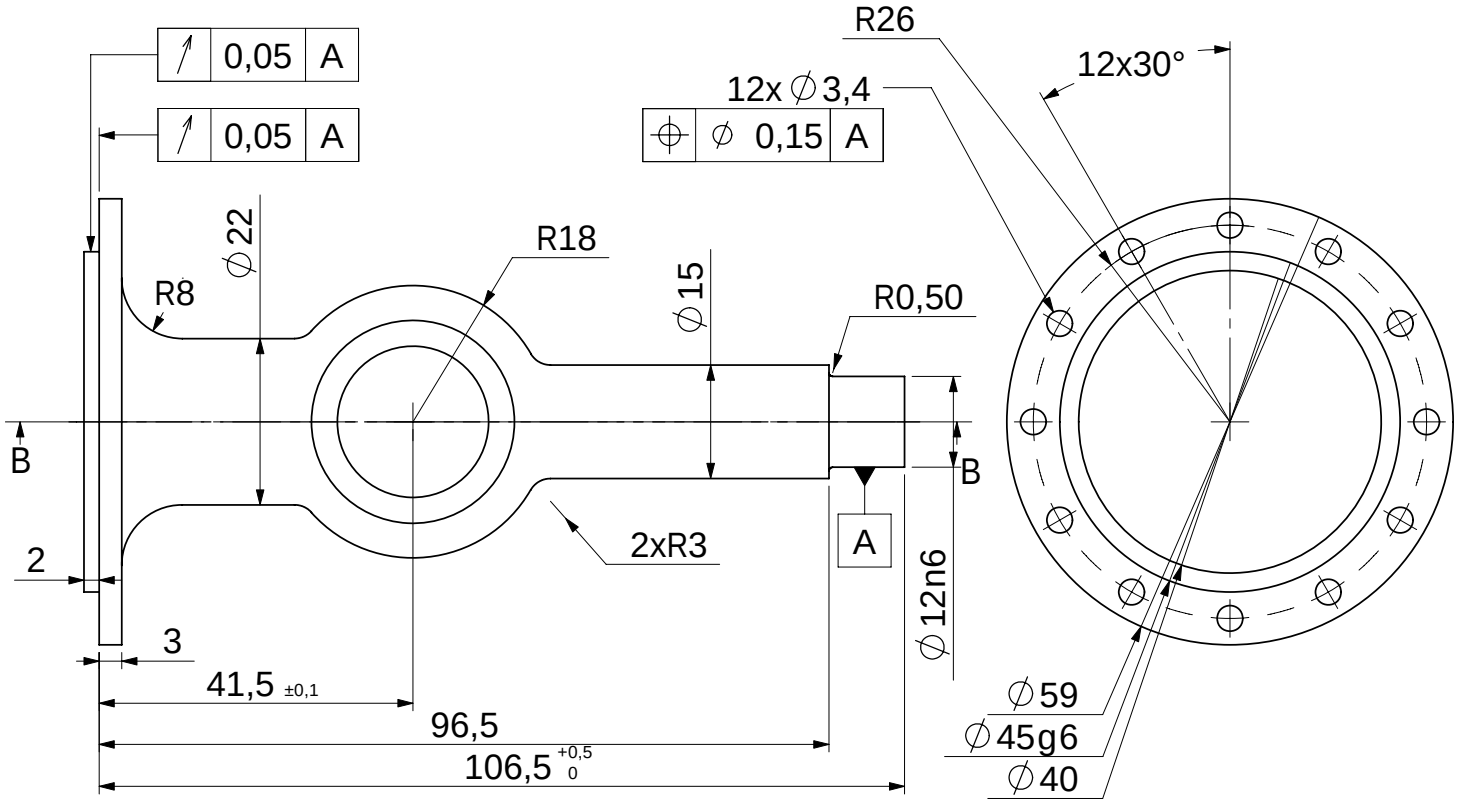
11.1 CrossAxle Waist

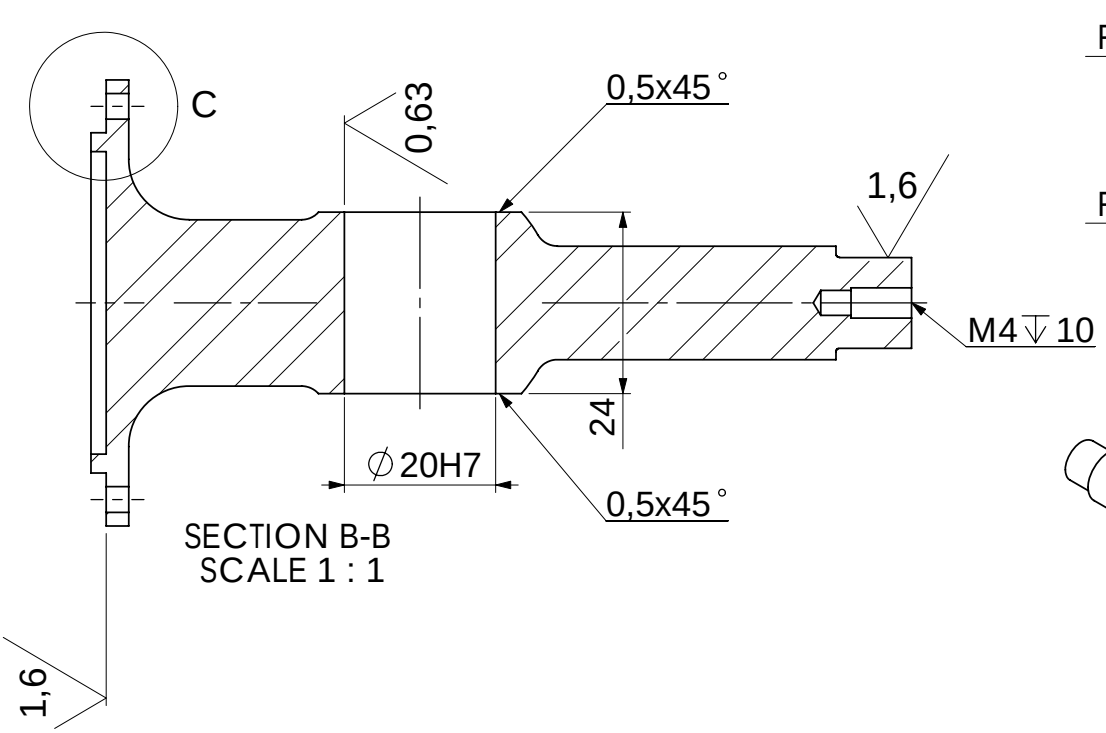
Aalborg Universitet
DMS10
Grp.126

SIZE A | DWG. NO. CrossAxleWaist | REV.

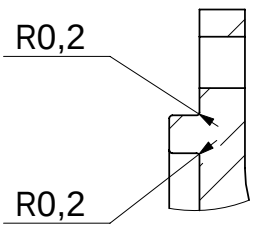
WEIGHT:

SHEET 3 OF 4

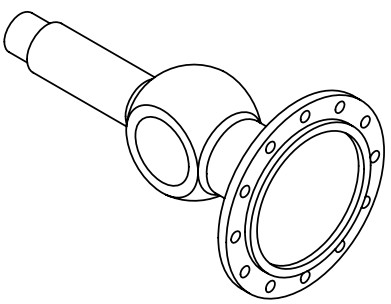




SECTION B-B
SCALE 1 : 1



DETAIL C
SCALE 2 : 1



Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Stål: C45

Number: 1 stk.

SCALE: 1:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

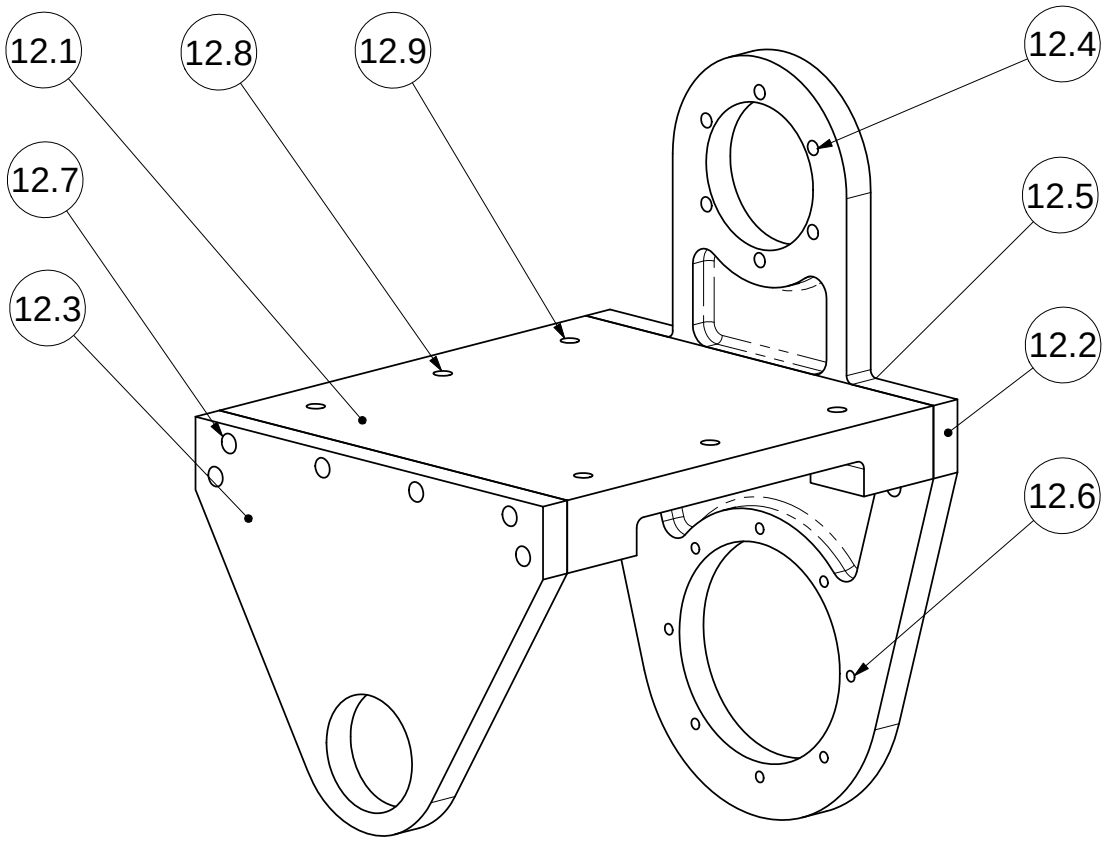
11.2 CrossAxle Waist

Aalborg Universitet
DMS10
Grp.126

SIZE	DWG. NO.	REV.
A	CrossAxleWaist	

WEIGHT:

SHEET 4 OF 4



Note:	DIMENSIONS ARE IN mm TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	NAME / DATE DRAWN CHECKED Forventet færdig: COMMENTS:	12.A Waist Roll Joint Aalborg Universitet DMS10 Grp.43a, Pon. 103
	MATERIAL: AlMg 6082-T6 Number: 1 stk. SCALE:1:2		SIZE A · DWG. NO. WaistRollJoint · REV. WEIGHT: · SHEET 1 OF 1

Nummer	Betegnelse (*)	Materiale	Antal
12.1	Plade 20mm	EN AW-6082 T6	1
12.2	Plade 10mm	EN AW-6082 T6	1
12.3	Plade 10mm	EN AW-6082 T6	1
12.4	Unbrako skrue	M3x12	12
12.5	Stift	Ø4x20	2
12.6	Unbrako skrue	M3x20	8
12.7	Unbrako skrue	M5x25	12
12.8	Unbrako skrue + møt.	M5x50	2
12.9	Unbrako skrue + møt.	M5x60	4

Note:

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

Number

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

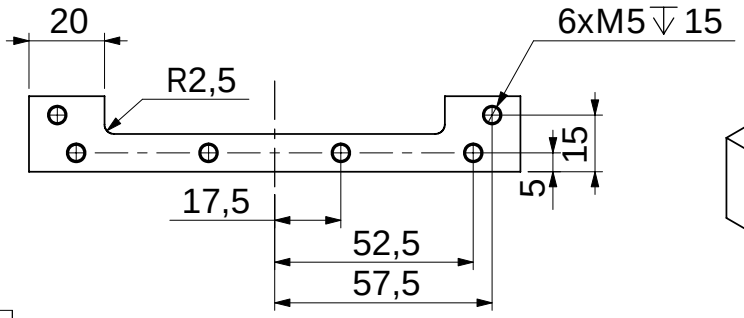
COMMENTS:

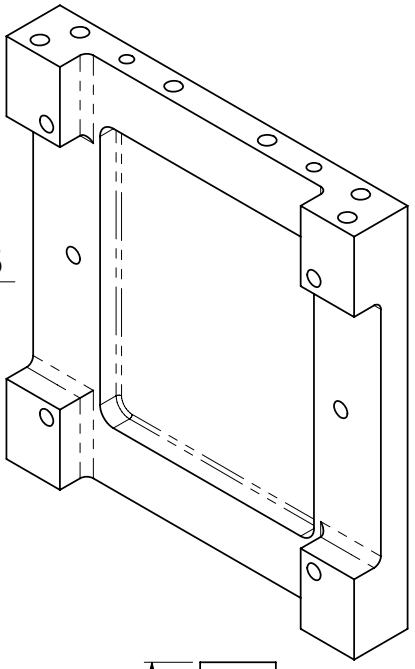
12.0 Waist Roll

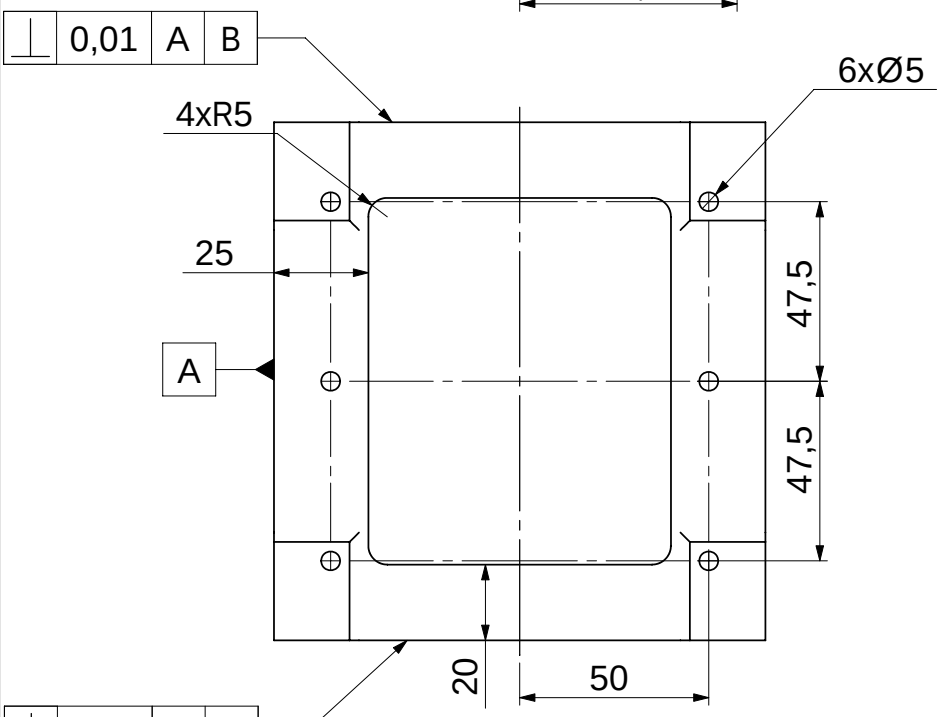
Aalborg Universitet
DMS10
Grp.43a, Pon. 103

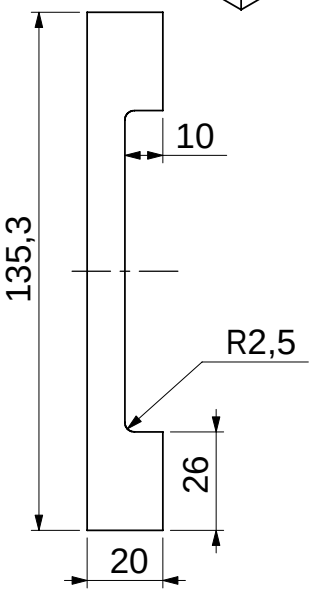
SIZE	DWG. NO.	REV.
A	12_0_WaistRoll	

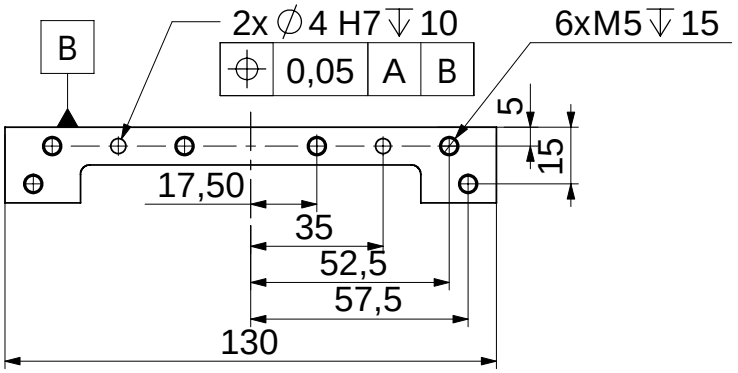
WEIGHT: SHEET 1 OF 1











Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

Fræses efter WaistRoll_12_1.iges

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 1 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

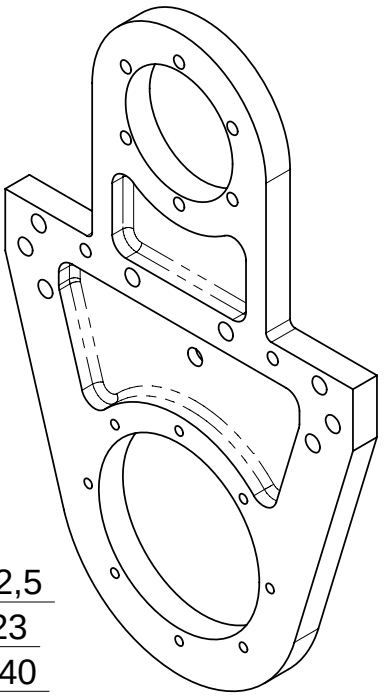
12.1 Waist Roll

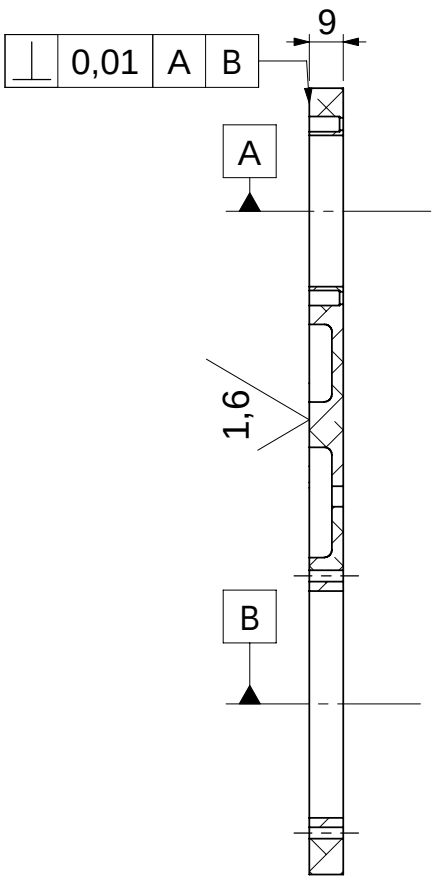
Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	12_1_WaistRoll	

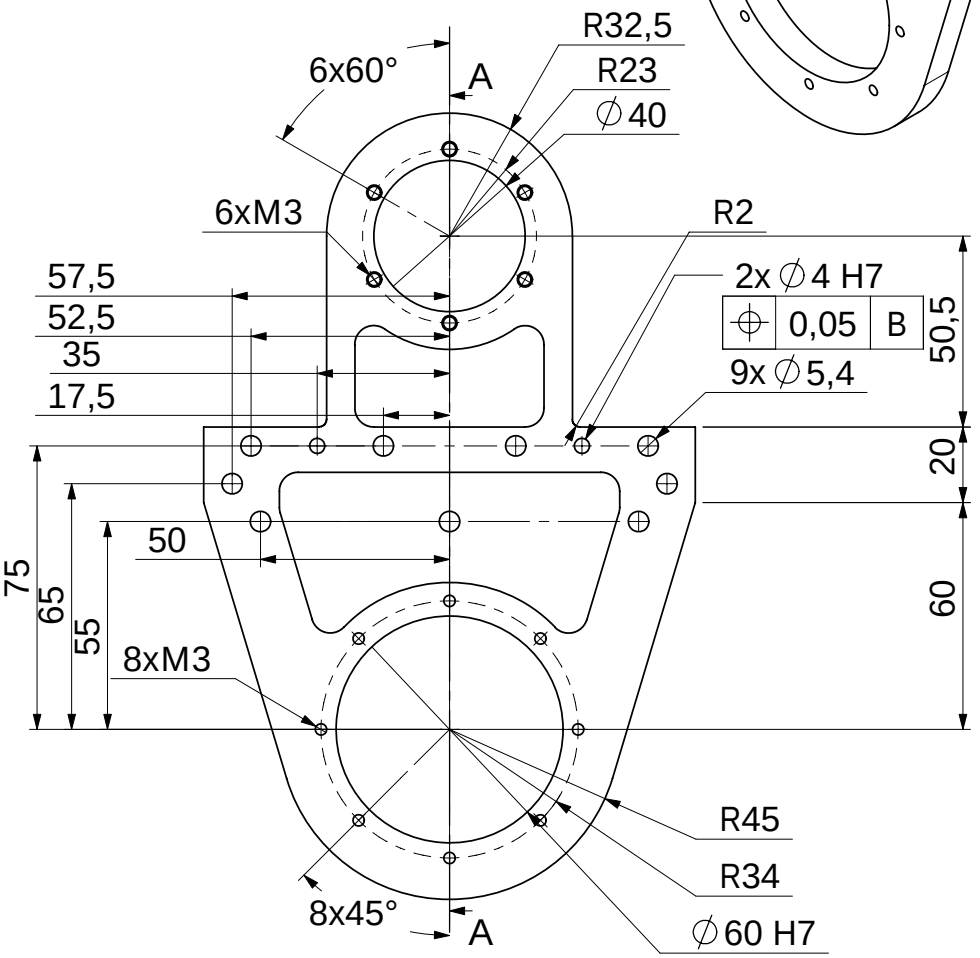
WEIGHT:

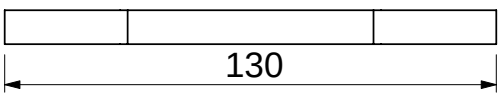
SHEET 1 OF 1





SECTION A-A
SCALE 1 : 2





Note:

For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

Fræses efter:
WaistRoll_12_2.iges

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 1 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

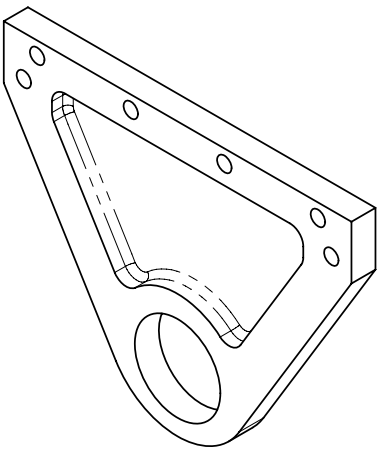
COMMENTS:

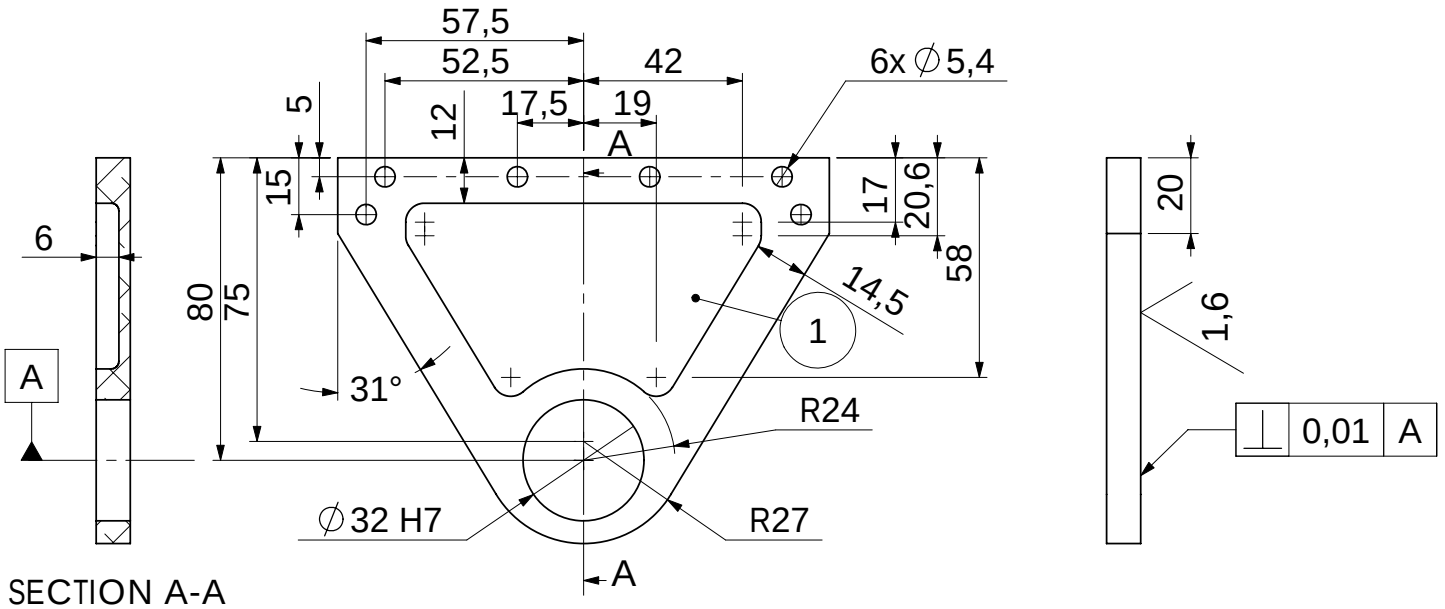
12.2 Waist Roll

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

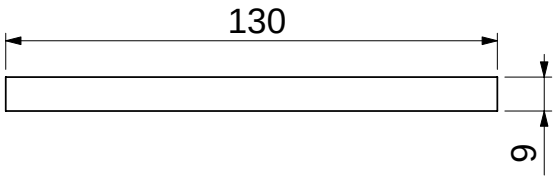
SIZE	DWG. NO.	REV.
A	12_2_WaistRoll	

WEIGHT: SHEET 1 OF 1





SECTION A-A



Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

(1) Udfræsning
Indv. radier 5mm
bund radius 2,5mm

Fræses efter WaistRoll_12_3.iges

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 1 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

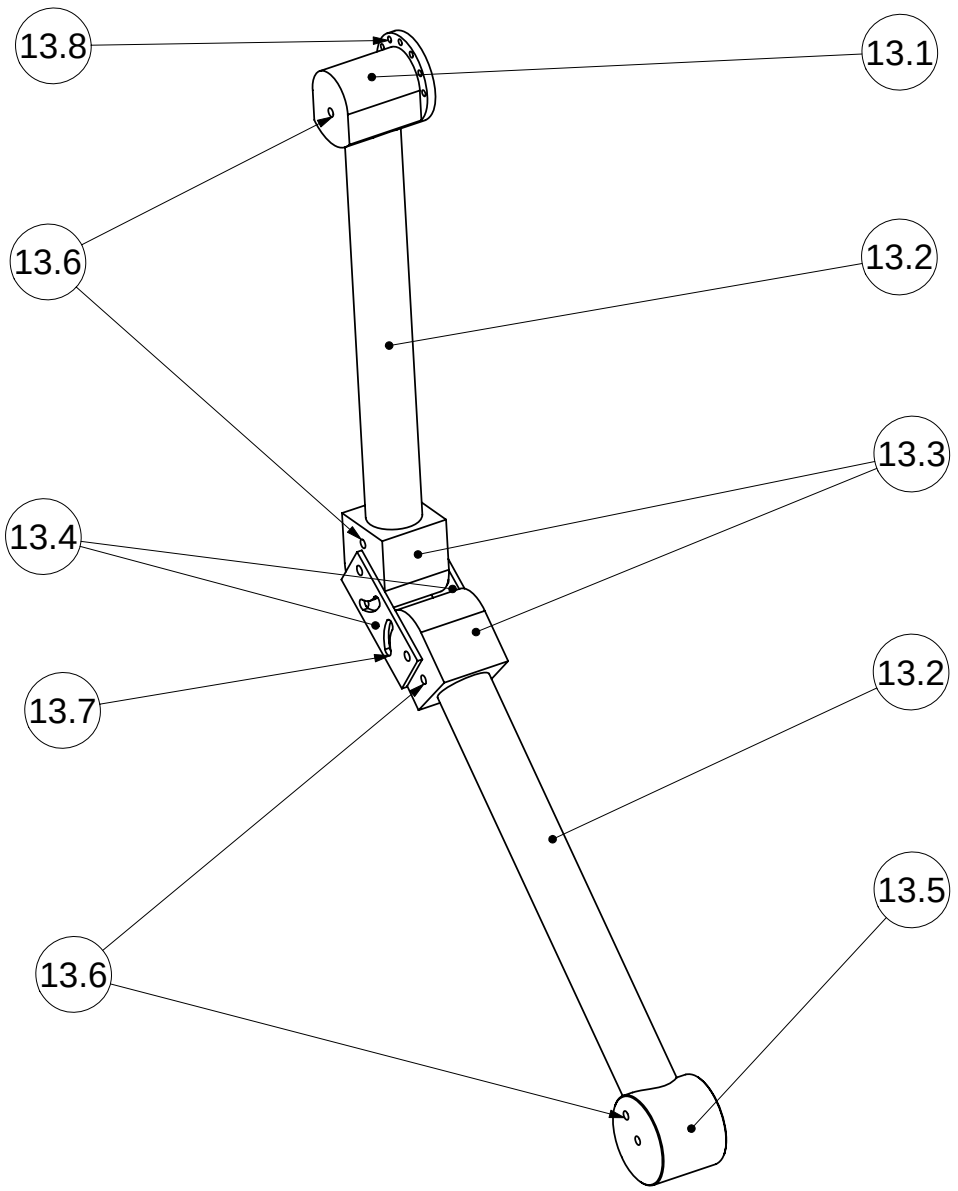
COMMENTS:

12.3 Waist Roll

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE A | DWG. NO. 12_3_WaistRoll | REV.

WEIGHT: | SHEET 1 OF 1



Note:

Højre/venstre parter identiske.
Dog vendes 13.4 på en af de to.

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

MATERIAL Al

Number 2 stk.

SCALE: 1:4

13.A Arm

Aalborg Universitet
DMS10
Grp.43a Pon.103

SIZE	DWG. NO.	REV.
A	Arms	

WEIGHT:

SHEET 1 OF 7

Nummer	Betegnelse*	Materiale	Antal
13.1	Aksel, Ø50mm	Al	2
13.2	Rør Ø30x26mm	Al	4
13.3	Skinne 40x40mm	Al	4
13.4	Plade 3mm	Al	4
13.5	Aksel Ø50mm	Al	2
13.6	Unbrako skrue	M5x40	8
13.7	Unbrako skrue	M5x12	16
13.8	Unbrako skrue	M3x15	14

Note:
* Forslag til råemne dimensioner.

MATERIAL

Number

	NAME	DATE
DRAWN		
CHECKED		

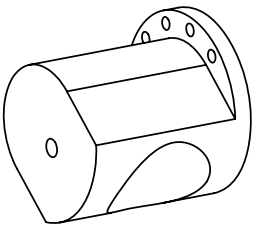
COMMENTS:

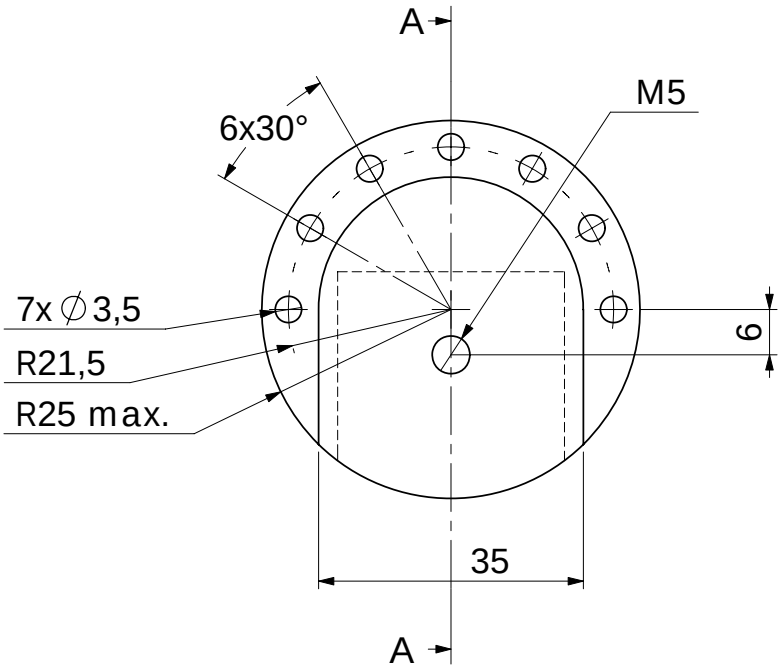
13.0 Arm

Aalborg Universitet
DMS10
Grp.43A, Pon103

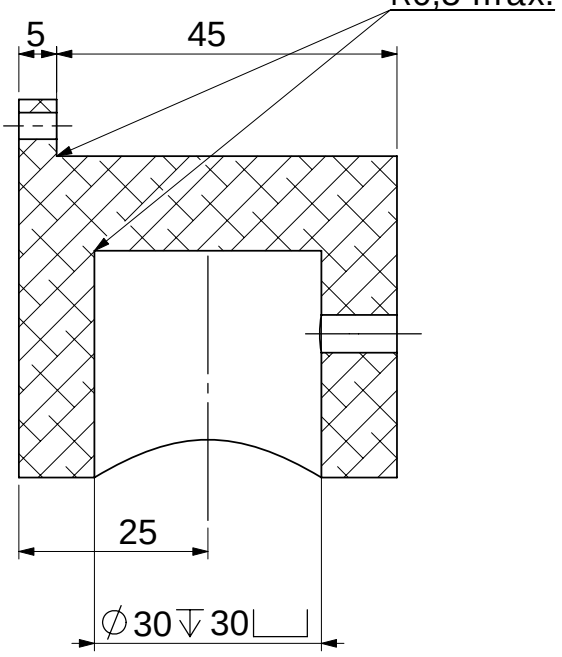
SIZE A | DWG. NO. Arms | REV.

WEIGHT: | SHEET 2 OF 7





SECTION A-A



Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

MATERIAL Al

Number 2 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

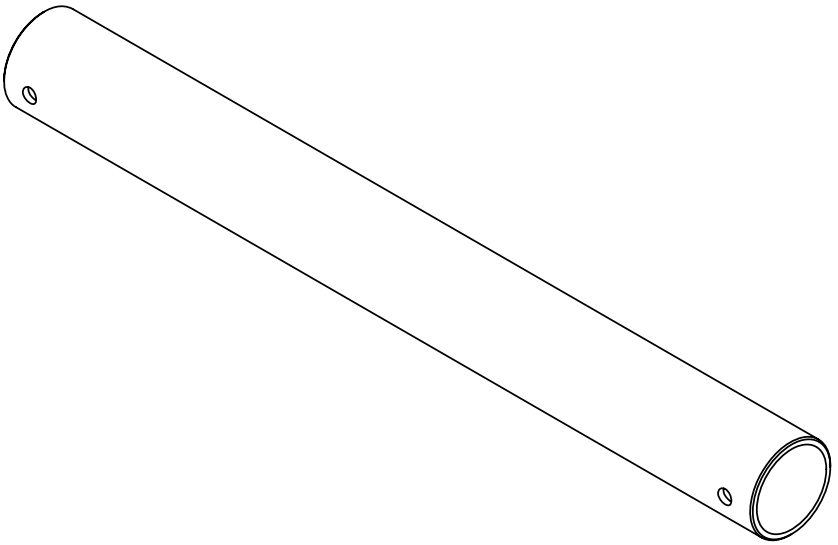
COMMENTS:

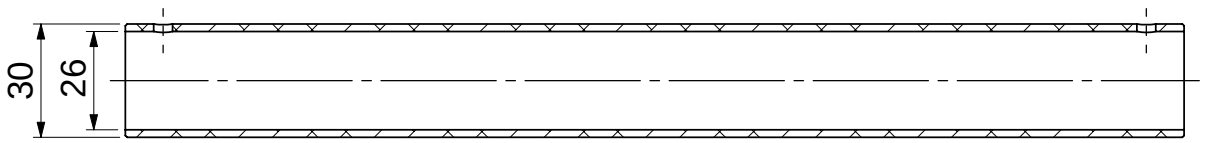
13.1 Arm

Aalborg Universitet
DMS10
Grp.43A, Pon103

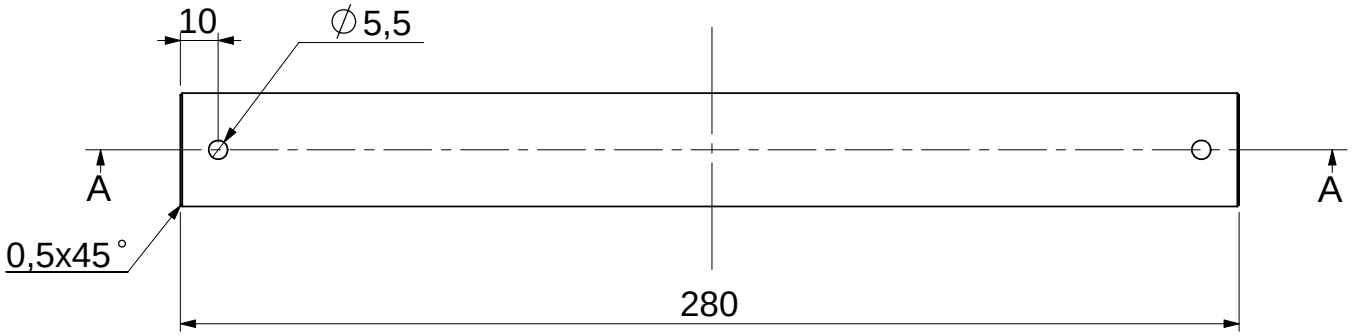
SIZE	DWG. NO.	REV.
A	Arms	

WEIGHT: SHEET 3 OF 7





SECTION A-A
SCALE 1 : 2



Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

MATERIAL: Al

Number: 4 stk.

SCALE: 1:2

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

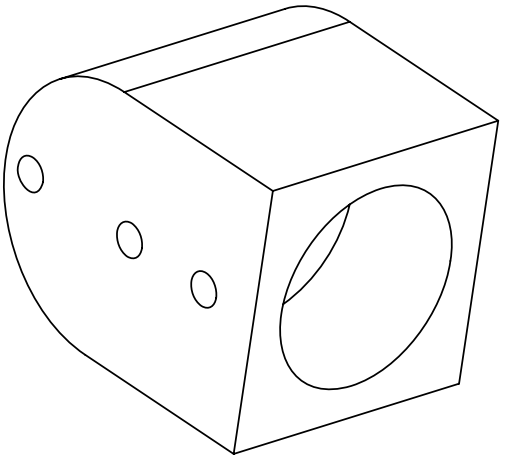
COMMENTS:

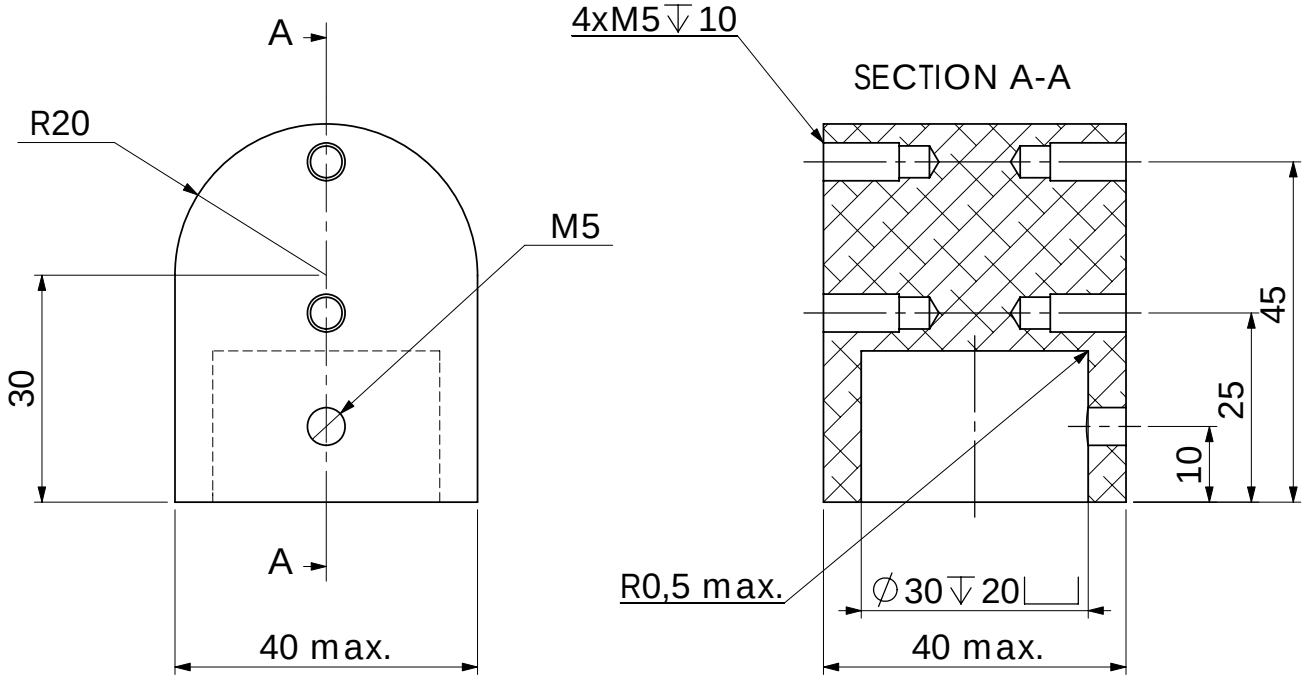
13.2 Arm

Aalborg Universitet
DMS10
Grp.43A, Pon103

SIZE A | DWG. NO. Arms | REV.

WEIGHT: | SHEET 4 OF 7





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

MATERIAL Al

Number 4 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

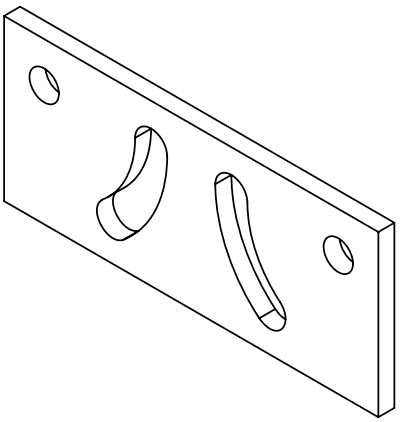
Forventet færdig:

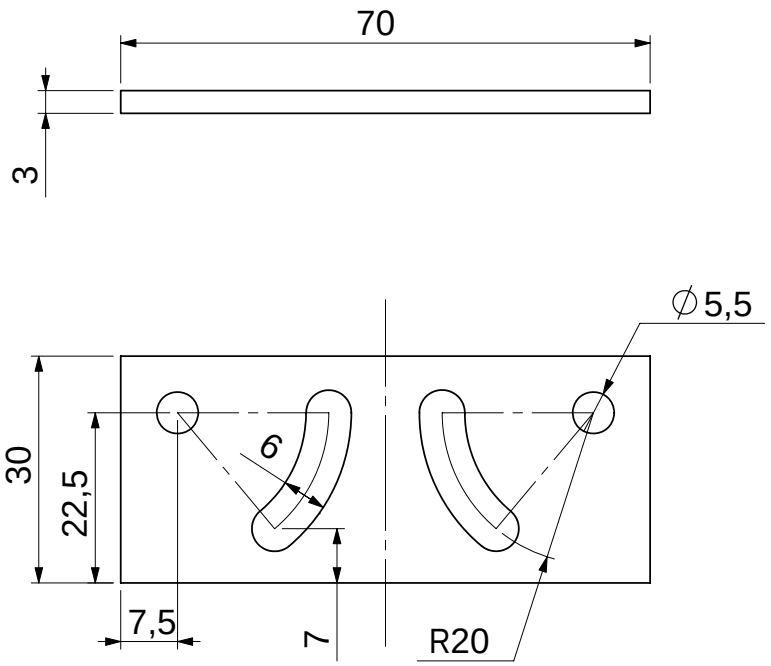
COMMENTS:

13.3 Arm

Aalborg Universitet
DMS10
Grp.43A, Pon103

SIZE	DWG. NO.	REV.
A	Arms	
	WEIGHT:	SHEET 5 OF 7





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

MATERIAL: Al

Number: 4 stk.

SCALE:1:1

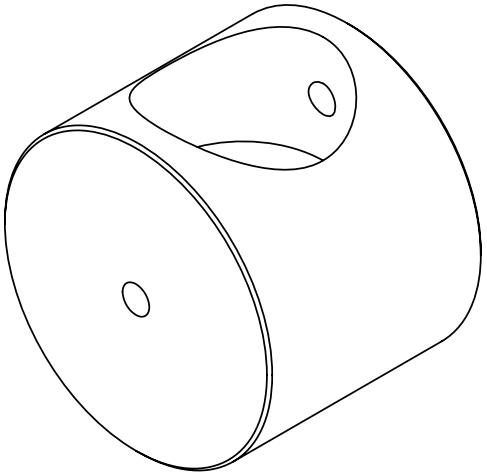
13.4 Arm

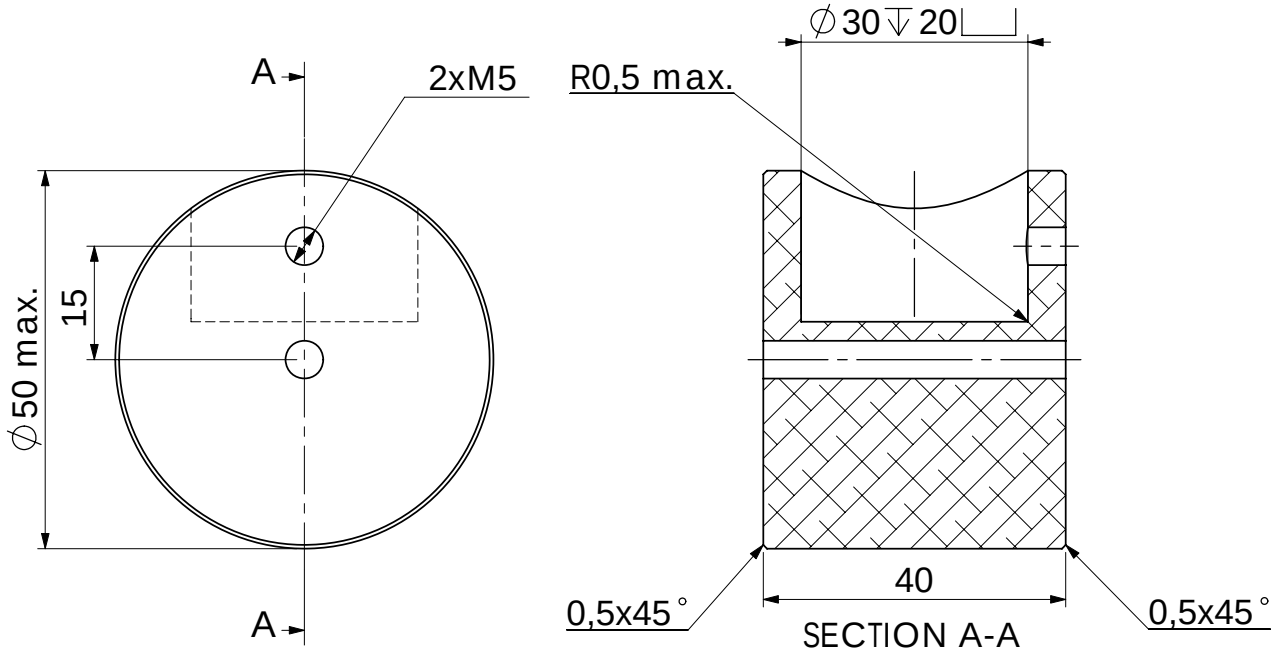
Aalborg Universitet
DMS10
Grp.43, Pon103

SIZE	DWG. NO.	REV.
A	Arms	

WEIGHT:

SHEET 6 OF 7





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

MATERIAL: Al

Number: 2 stk.

SCALE:1:1

13.5 Arm

Aalborg Universitet
DMS10
Grp.43, Pon103

SIZE	DWG. NO.	REV.
A	Arms	
	WEIGHT:	SHEET 7 OF 7

Nummer	Betegnelse	Materiale	Antal
14.1	30x30x2	Al (Svejsbart)	1
14.2	Plade: 3mm	Al	1
14.3	30x30x2	Al (Svejsbart)	1
14.4	Plade: 3mm	Al	2
14.5	30x30x2	Al (Svejsbart)	1
14.6	Rør: 10x2	Al (Svejsbart)	34
14.7	Skinne: 30x5	Al (Svejsbart)	4
14.8	Øjemøtrik	6mm	4
14.9	Unbrako skrue + møt.	5x50	4
14.10	Unbrako skrue + møt.	5x45	12
14.11	Unbrako skrue	3x12	16
14.12	Unbrako skrue + møt.	5x45	4
14.13	Unbrako skrue + møt.	5x45	4
14.14	Plade: 3mm	EN AW-6082 T6	2

Note:

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

Number

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

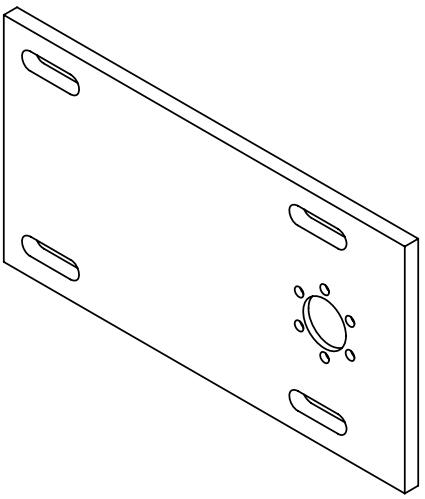
COMMENTS:

14.0 Torso

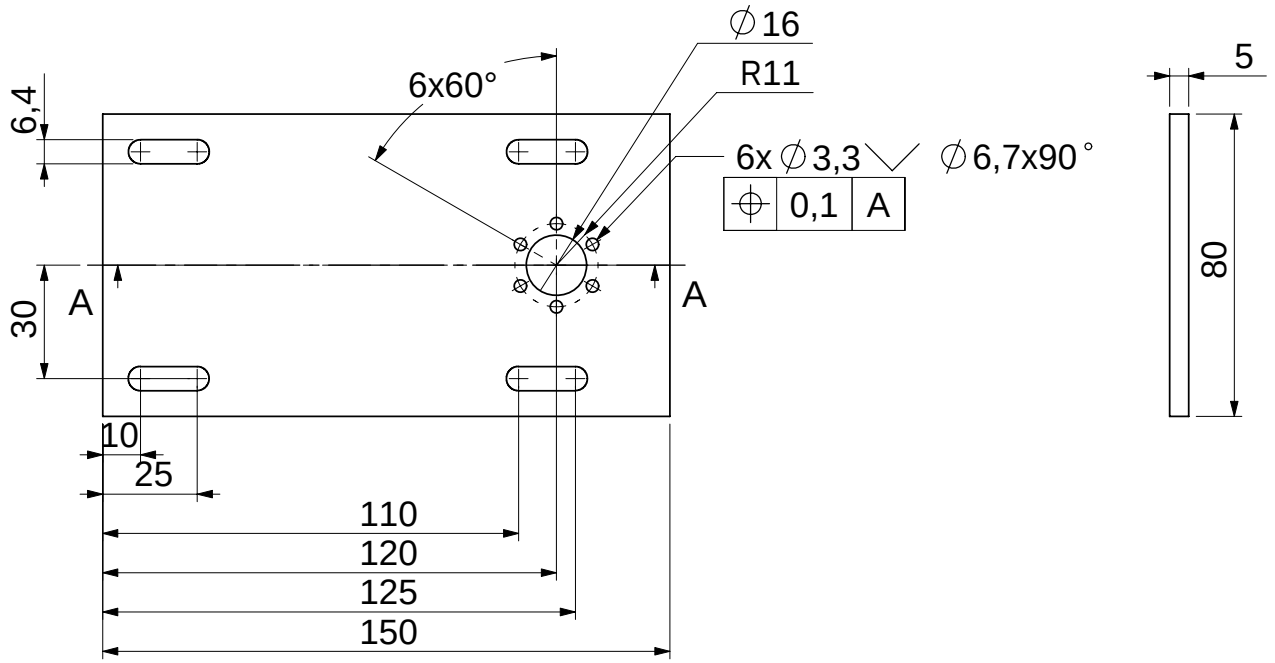
Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE A | DWG. NO. 14_0_Torso | REV.

WEIGHT: | SHEET 1 OF 1



SECTION A-A



Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number 2 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

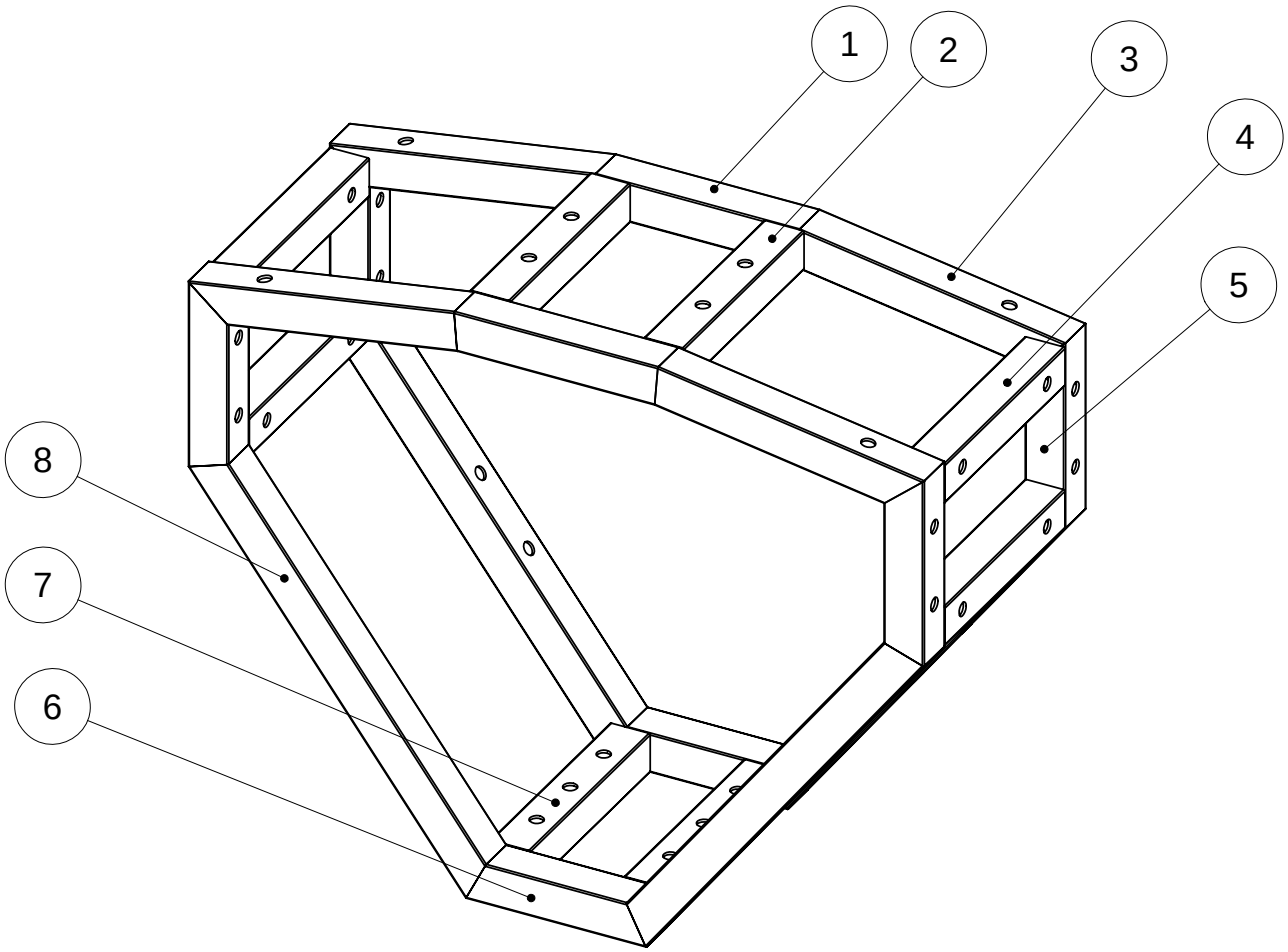
Forventet færdig:

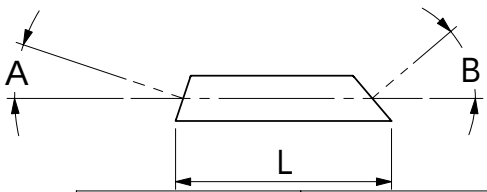
COMMENTS:

14.14 Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	14_14_Torso	
	WEIGHT:	SHEET 1 OF 1





Nummer	Længde [L]	Vinkel 1 [A]	Vinkel 2 [B]	Antal
1	152,5	4,5	4,5	2
2	170	0	0	2
3	201,5	40,5	4,5	4
4	170	0	0	4
5	143	40,5	18,5	4
6	135	26,5	26,5	2
7	170	0	0	2
8	345	26,5	18,5	4

Note:

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Al (Svejsbart)

Number: 1 stk.

SCALE:1:5

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

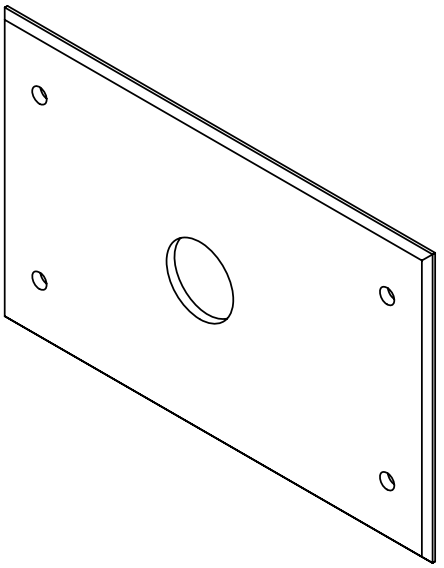
COMMENTS:

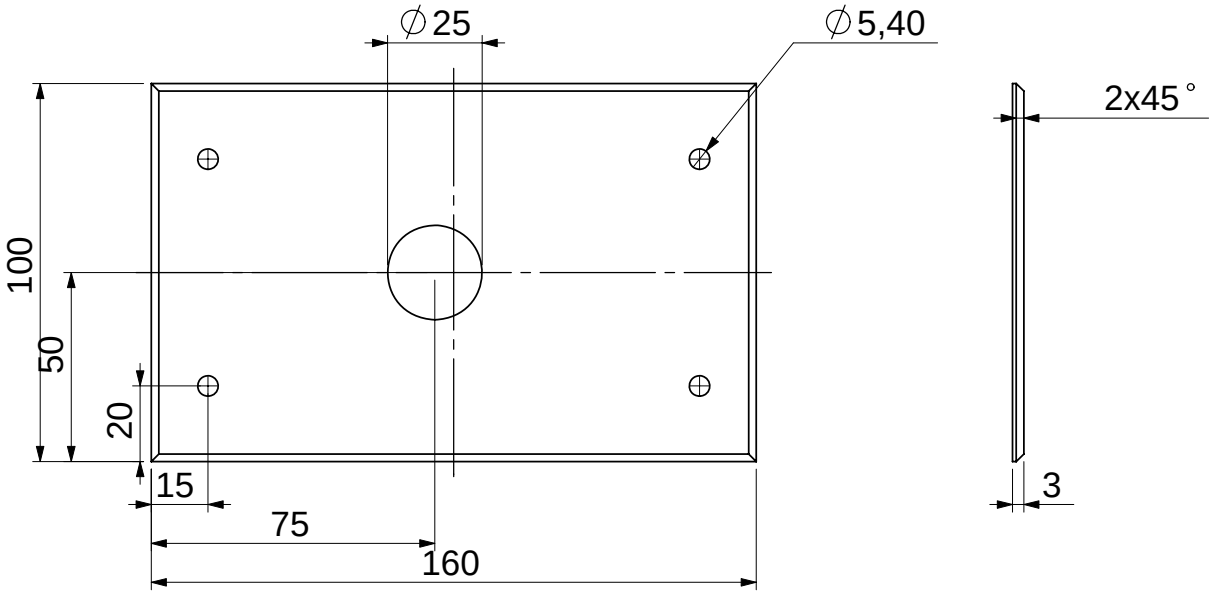
14.1A Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE A | DWG. NO. 14_1A_Torso | REV.

WEIGHT: | SHEET 1 OF 1





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 1 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

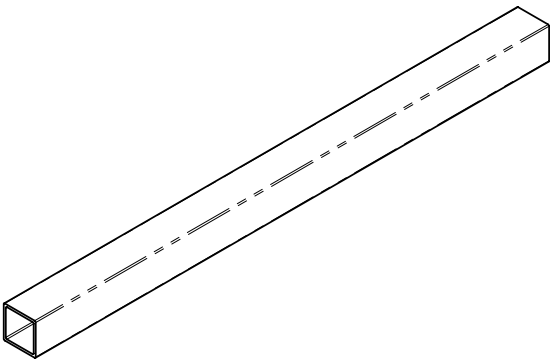
Forventet færdig:

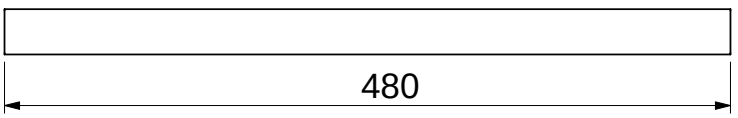
COMMENTS:

14.2 Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	14_2_Torso	
	WEIGHT:	SHEET 1 OF 1





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

Kvadratisk rør: 30x30x2

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Al (Svejsbart)

Number 1 stk.

SCALE:1:5

	NAME	DATE
DRAWN		
CHECKED		

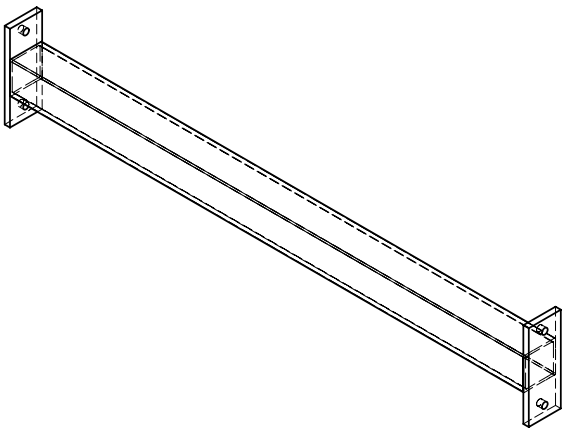
Forventet færdig:

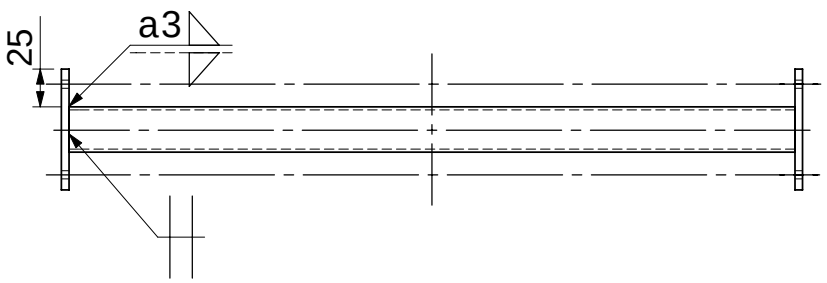
COMMENTS:

14.3 Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	14_3_Torso	
	WEIGHT:	SHEET 1 OF 1





Note:
For ikke tolerance specificeret mål.
Benyttes DS/EN ISO 13920 A

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

Number 1 stk.

SCALE:1:5

	NAME	DATE
DRAWN		
CHECKED		

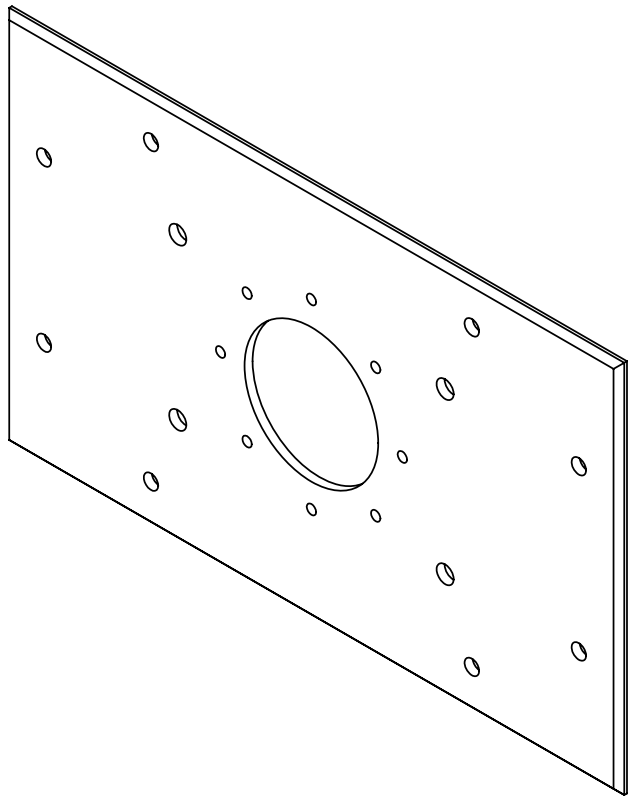
Forventet færdig:

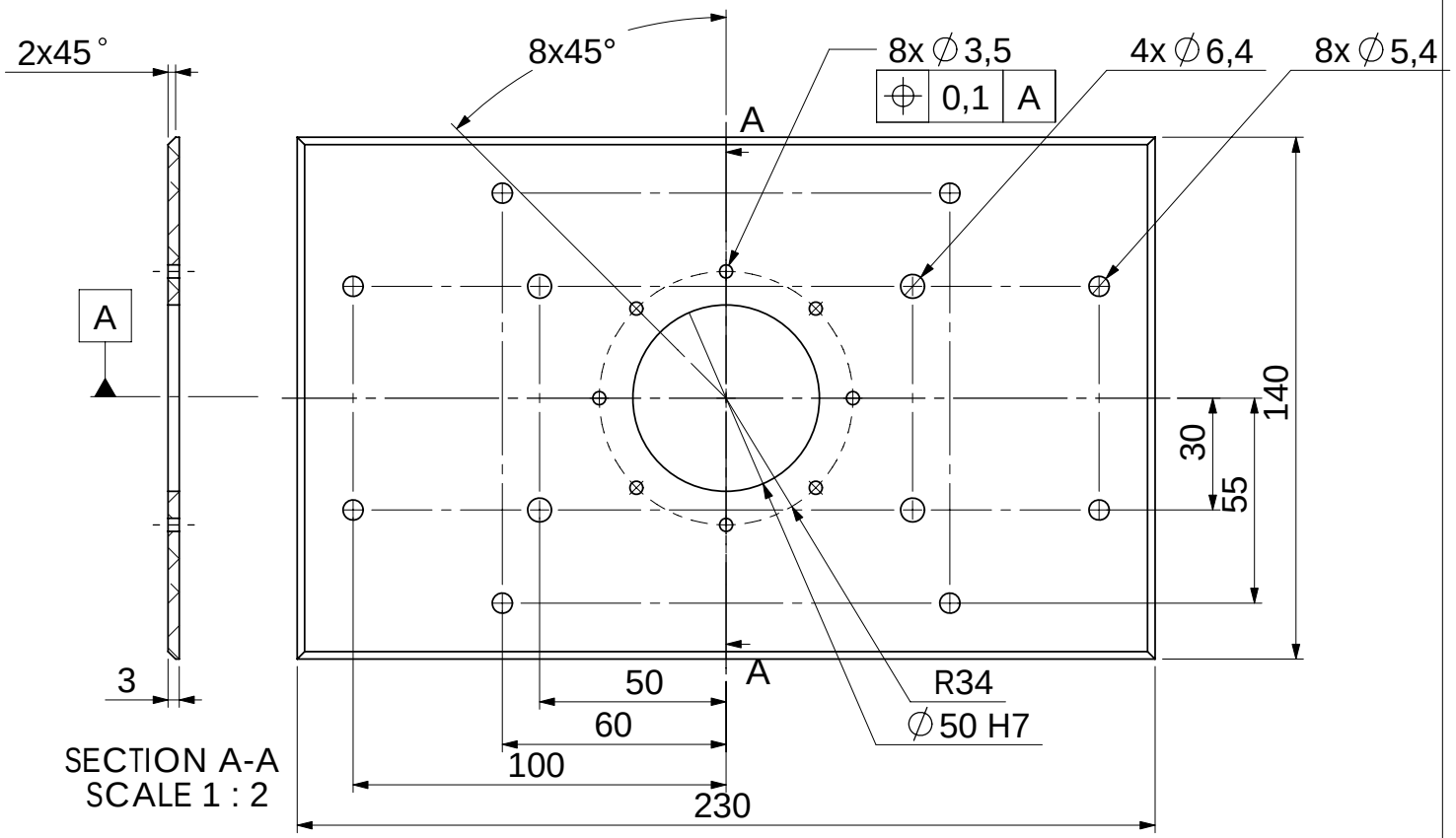
COMMENTS:

14.3A Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	14_3A_Torso	
	WEIGHT:	SHEET 1 OF 1





SECTION A-A
SCALE 1 : 2

Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: AW-6082 T6

Number: 2 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

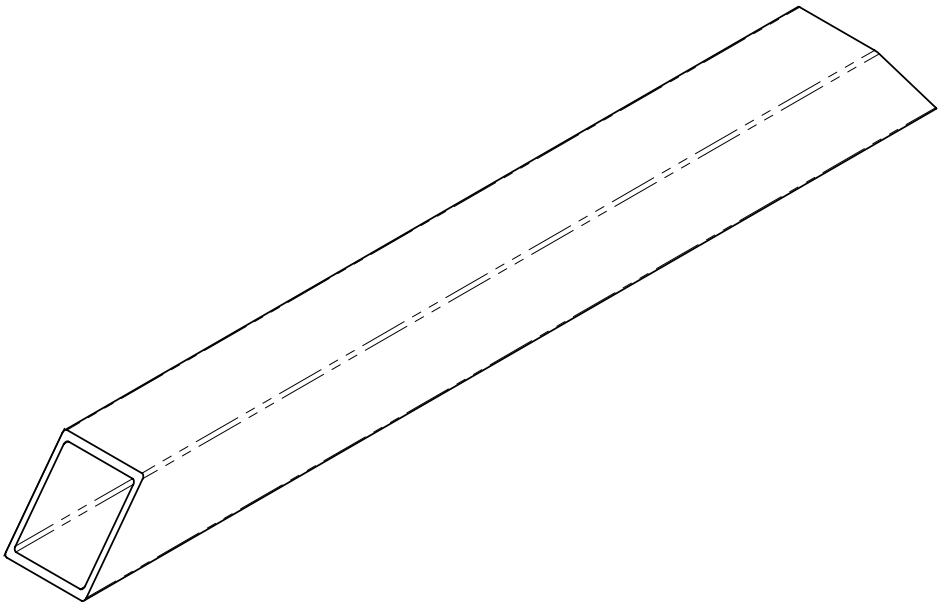
Forventet færdig:

COMMENTS:

14.4 Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	14_4_Torso	
	WEIGHT:	SHEET 1 OF 1



Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-G

Kvadratisk rør: 30x30x2

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Al (Svejsbart)

Number: 1 stk.

SCALE:1:2

	NAME	DATE
DRAWN		
CHECKED		

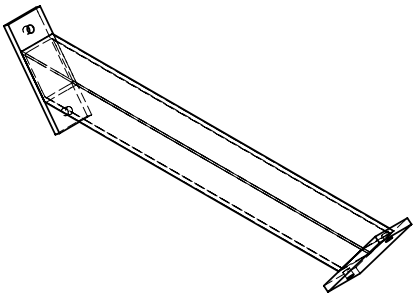
Forventet færdig:

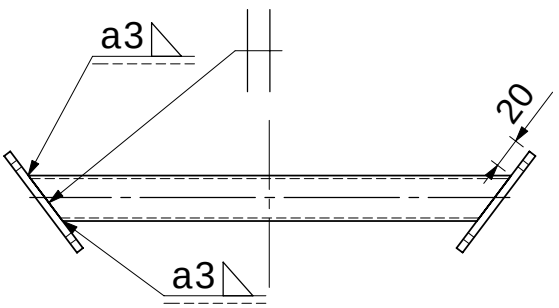
COMMENTS:

14.5 Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	14_5_Torso	
	WEIGHT:	SHEET 1 OF 1





Note:
For ikke tolerance specificeret mål.
Benyttes DS/EN ISO 13920 A

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

Number 1 stk.

SCALE:1:5

	NAME	DATE
DRAWN		
CHECKED		

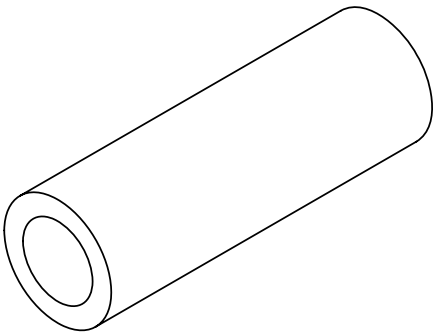
Forventet færdig:

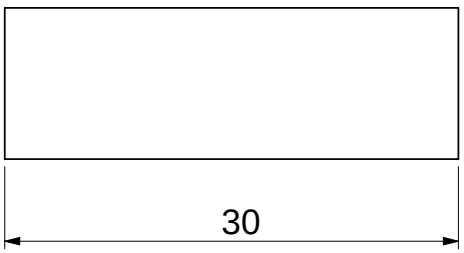
COMMENTS:

14.5A Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	14_5A_Torso	
	WEIGHT:	SHEET 1 OF 1





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

Dimension: 10x2

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL: Al (Svejsbart)

Number: 34 stk.

SCALE:2:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

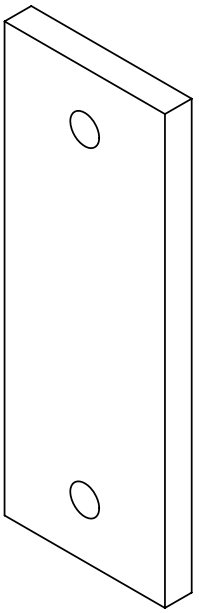
COMMENTS:

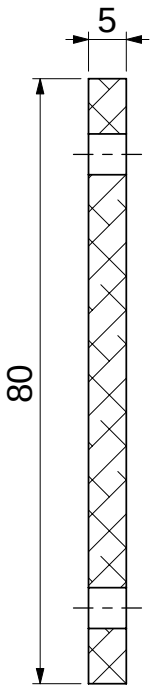
14.6 Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

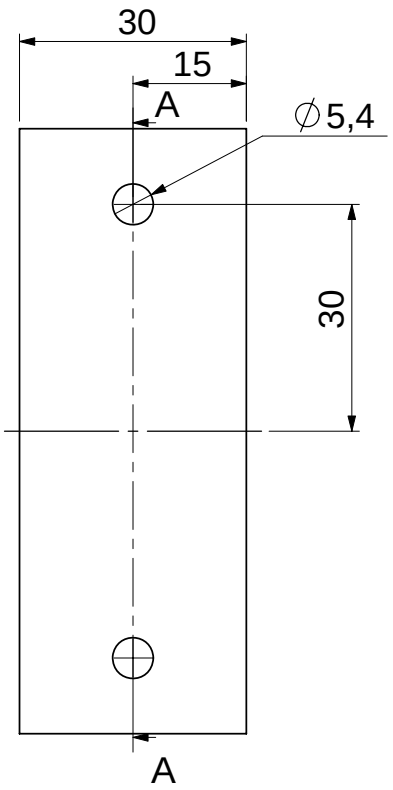
SIZE A | DWG. NO. 14_6_Torso | REV.

WEIGHT: | SHEET 1 OF 1





SECTION A-A

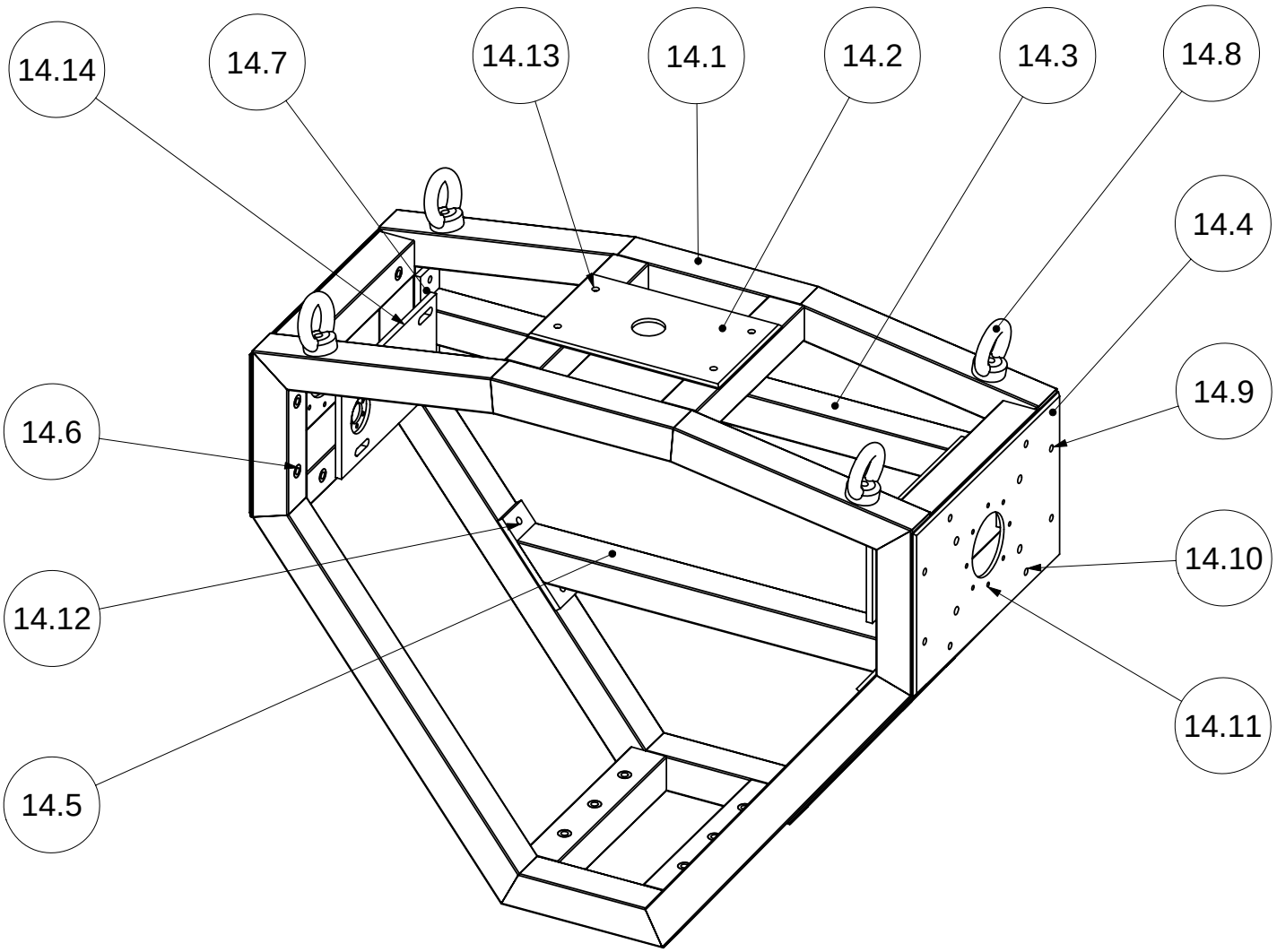


Note: For ikke tolerance specificeret mål. Benyttes DS/ISO 2768-m	DIMENSIONS ARE IN mm TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	NAME / DATE DRAWN CHECKED Forventet færdig: COMMENTS:
	MATERIAL: Al (Svejsbart) Number: 4 stk. SCALE:1:1	

14.7 Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	14_7_Torso	
	WEIGHT:	SHEET 1 OF 1



Note:

DIMENSIONS ARE IN mm
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

MATERIAL

Number 1 stk.

SCALE:1:5

	NAME	DATE
DRAWN		
CHECKED		

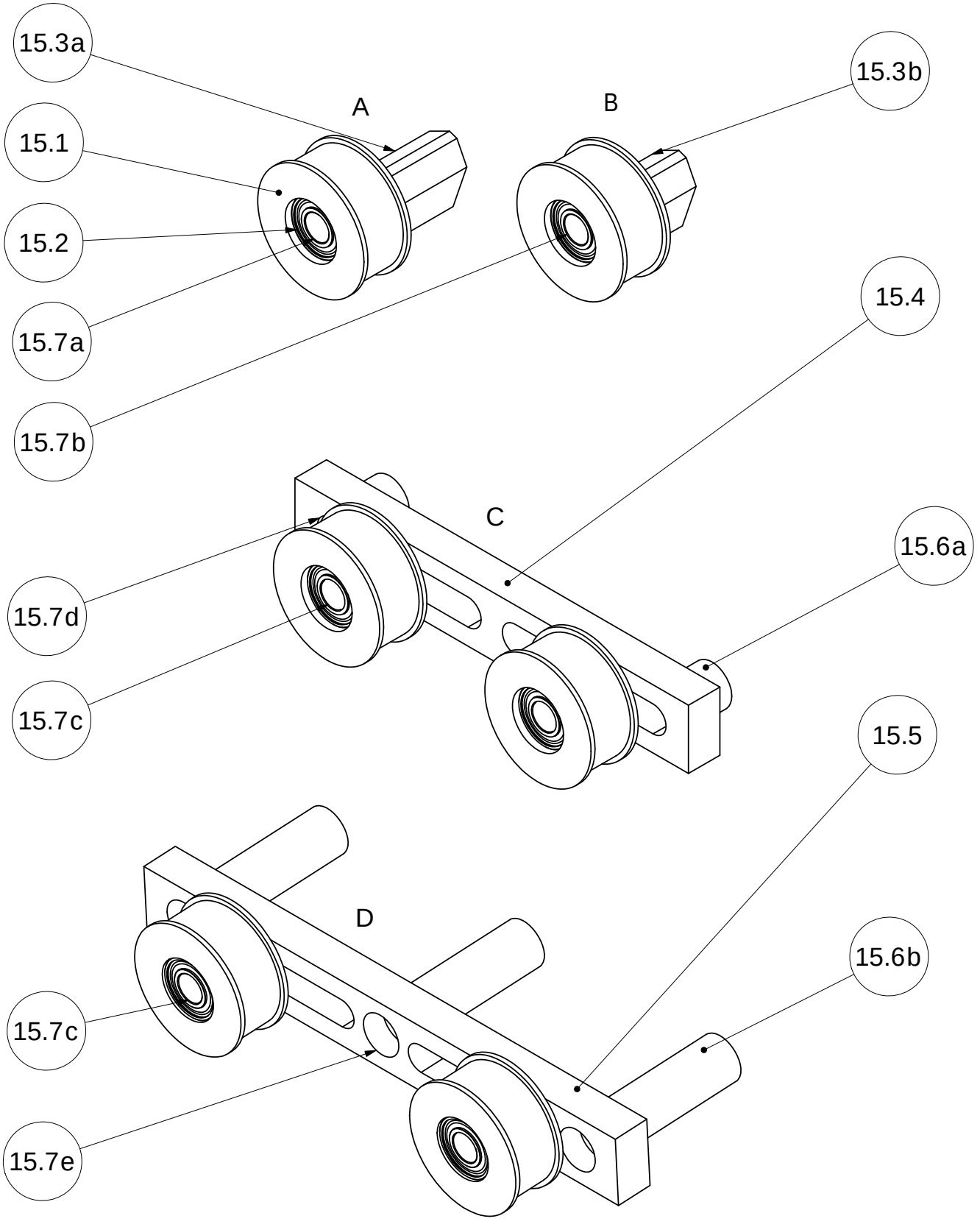
Forventet færdig:

COMMENTS:

14.A Torso

Aalborg Universitet
DMS10
Grp.43a, Pon. 103

SIZE	DWG. NO.	REV.
A	Torso	
	WEIGHT:	SHEET 1 OF 1



Note:			NAME	DATE
		DRAWN		
		CHECKED		
		Forventet færdig:		
	MATERIAL	COMMENTS:		
	Number			
	SCALE:1:1			

15.A BeltTightning

Aalborg Universitet
DMS10
Grp.43a Pon.103

SIZE	DWG. NO.	REV.
A	BeltTightning	
	WEIGHT:	SHEET 2 OF 6

Nummer	Betegnelse*	Materiale	Antal
15.1	Aksel Ø30	Nylon	12
15.2	Kugleleje	SKF 628-6-2Z	24
15.3a 15.3b	Skinne 20x10	Al	6 2
15.4	Skinne 20x10	Al	1
15.5	Skinne 20x10	Al	1
15.6a 15.6b	Rør Ø10x2x6** Rør Ø10x2x28,5**	Al	2 3
15.7a 15.7b 15.7c 15.7d 15.7e	Unbrako bolt + møtrik + skiver	M6x45 M6x40 M6x30 M6x25 M5x45	6 2 4 2 3

Note:
* Forslag til råemne dimension
** DxtxL

MATERIAL

Number

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

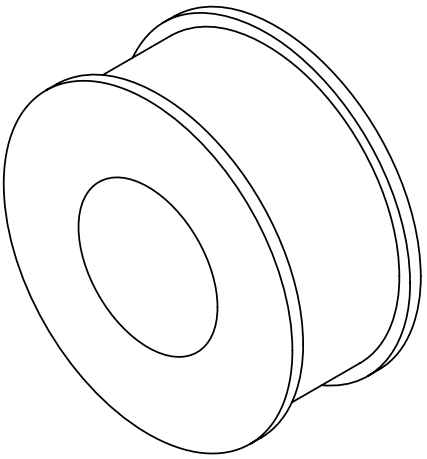
COMMENTS:

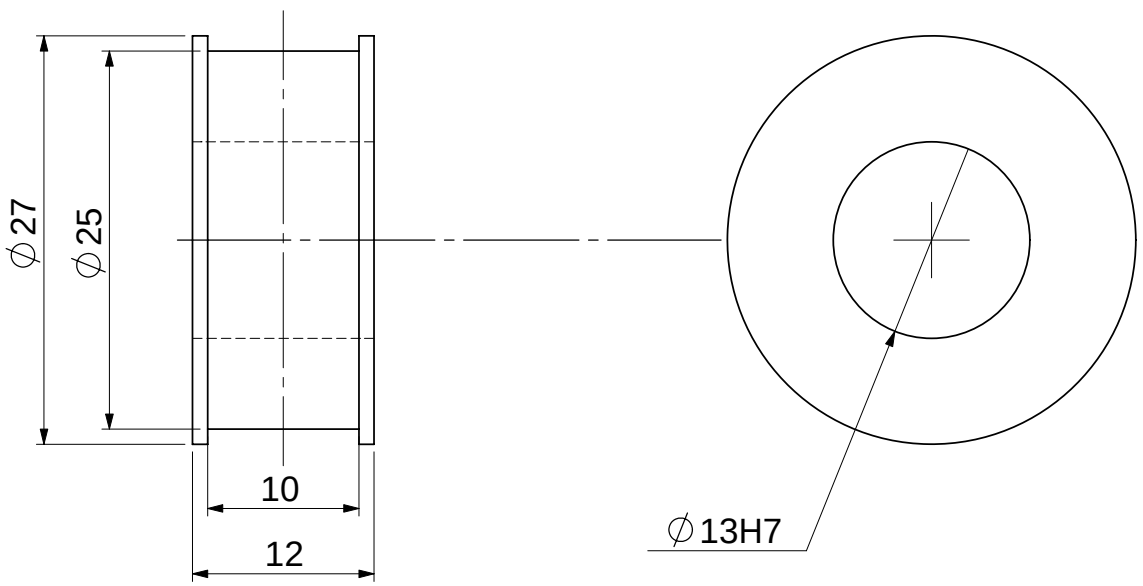
15.0 BeltThightning

Aalborg Universitet
DMS10
Grp.43, Pon103

SIZE A | DWG. NO. BeltTightning | REV.

WEIGHT: | SHEET 2 OF 6





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

MATERIAL Nylon

Number 12 stk.

SCALE:2:1

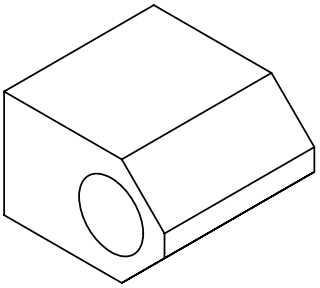
15.1 BeltTightning

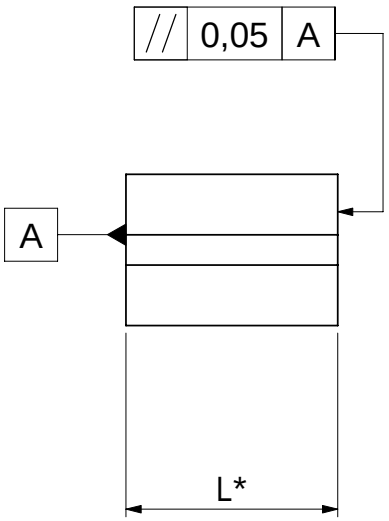
Aalborg Universitet
DMS10
Grp.43A, Pon103

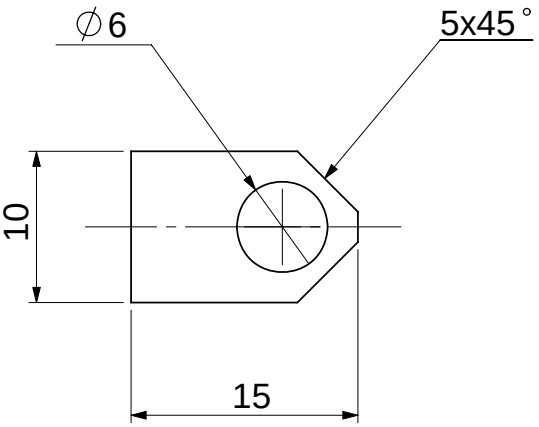
SIZE	DWG. NO.	REV.
A	BeltTightning	

WEIGHT:

SHEET 3 OF 6







Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

*
6stk. 15.3a, L=14mm
2stk. 15.3b, L=21mm

MATERIAL Al
Number 8 ialt*
SCALE:2:1

	NAME	DATE
DRAWN		
CHECKED		

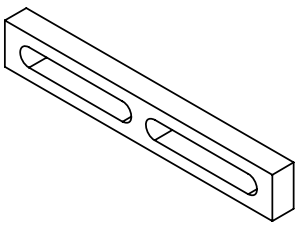
Forventet færdig:

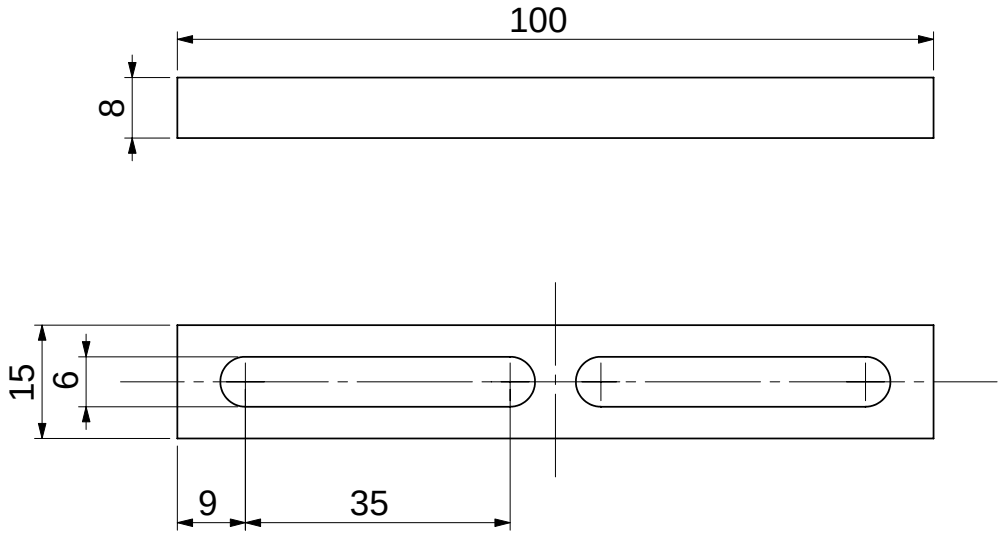
COMMENTS:

15.a-b BeltTightning

Aalborg Universitet
DMS10
Grp.43A, Pon103

SIZE	DWG. NO.	REV.
A	BeltTightning	
	WEIGHT:	SHEET 4 OF 6





Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

MATERIAL: Al

Number: 1 stk.

SCALE:1:1

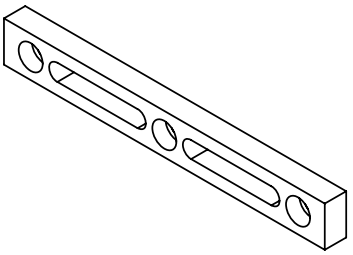
15.4 BeltTightning

Aalborg Universitet
DMS10
Grp.43A, Pon 103

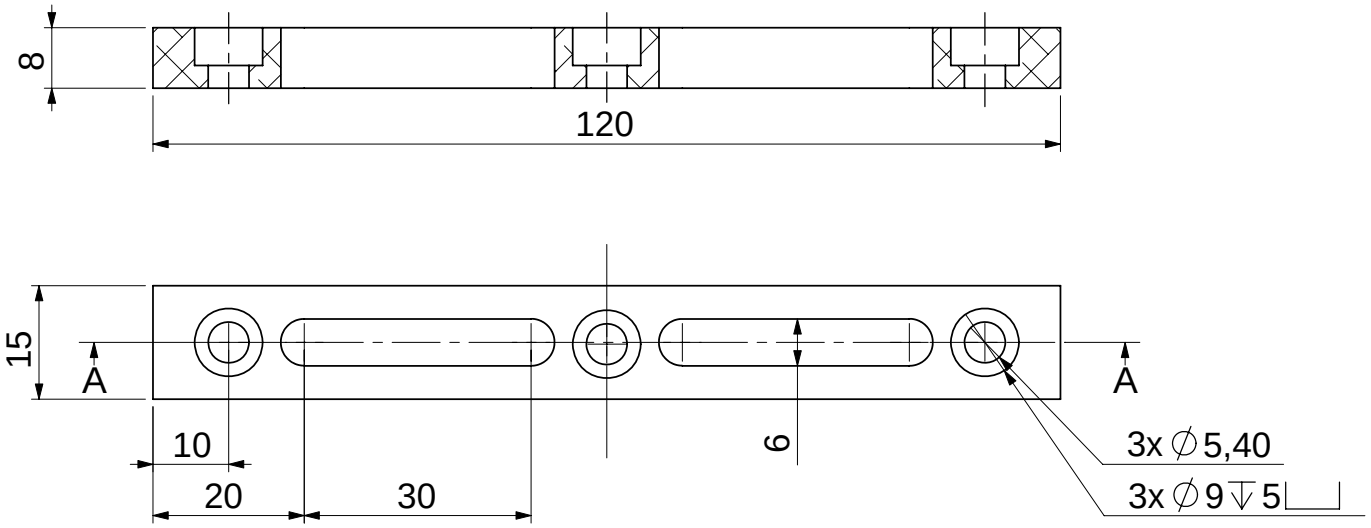
SIZE	DWG. NO.	REV.
A	BeltTightning	

WEIGHT:

SHEET 5 OF 6



SECTION A-A



Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

MATERIAL: Al

Number: 1 stk.

SCALE:1:1

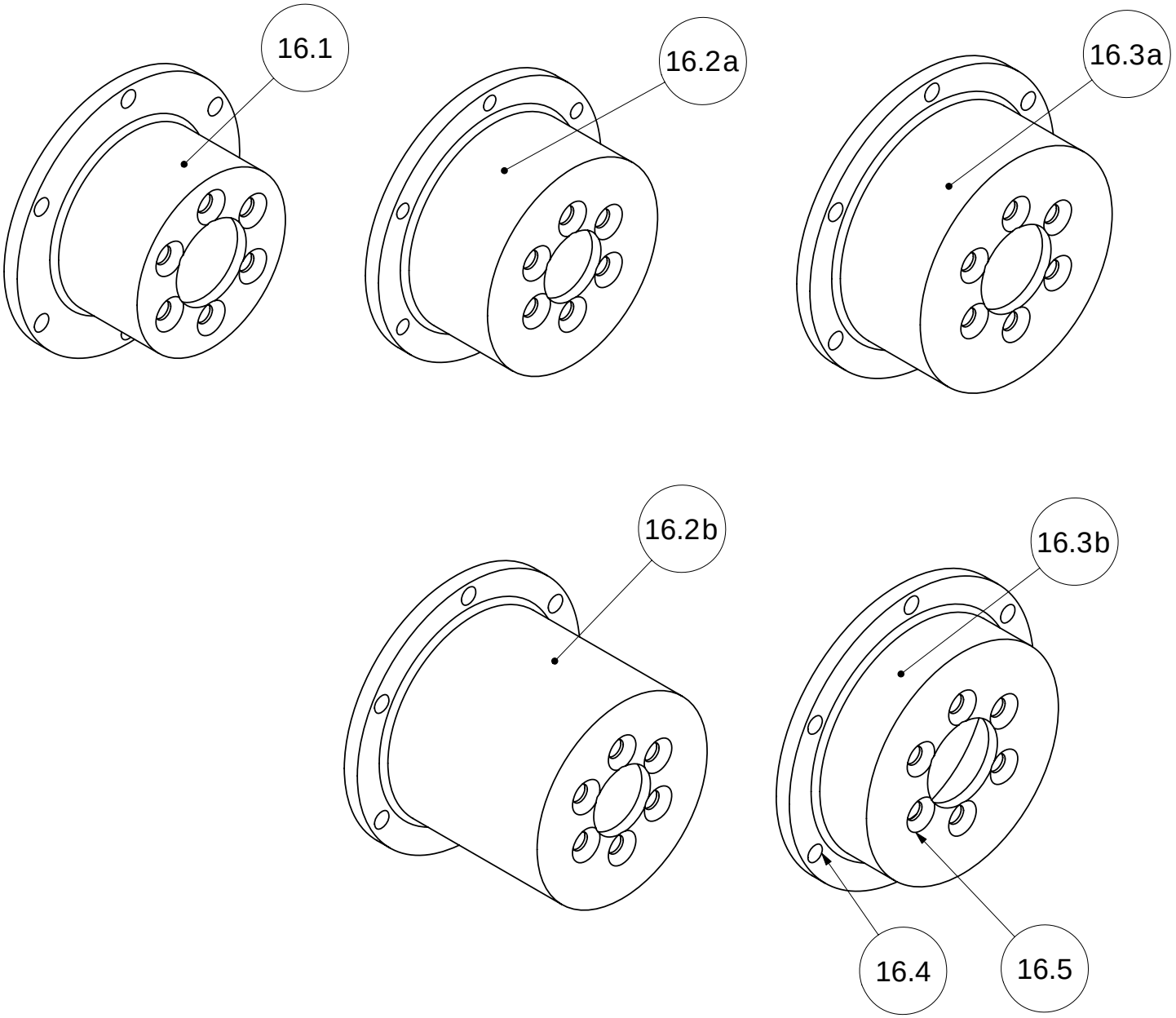
15.5 BeltTighning

Aalborg Universitet
DMS10
Grp.43A, Pon103

SIZE	DWG. NO.	REV.
A	BeltTightning	

WEIGHT:

SHEET 6 OF 6



Nummer	Betegnelse	Materiale	Antal
16.1	60W	Al	1
16.2a	90Wa	Al	1
16.2b	90Wb	Al	1
16.3a	150Wa	Al	2
16.3b	150Wb	Al	6
16.4	Unbrako skrue	M3x10	66
16.5	Skrue m/indv. 6kt. DIN7991	M3x6	66

Note:

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

MATERIAL

Number

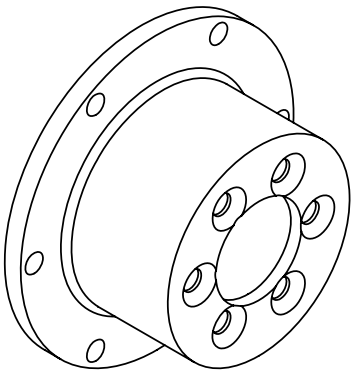
SCALE:1:1

COMMENTS:

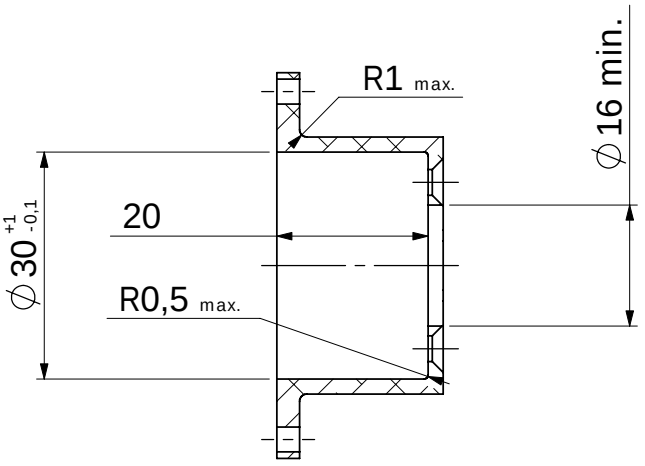
16.A MotorMounts

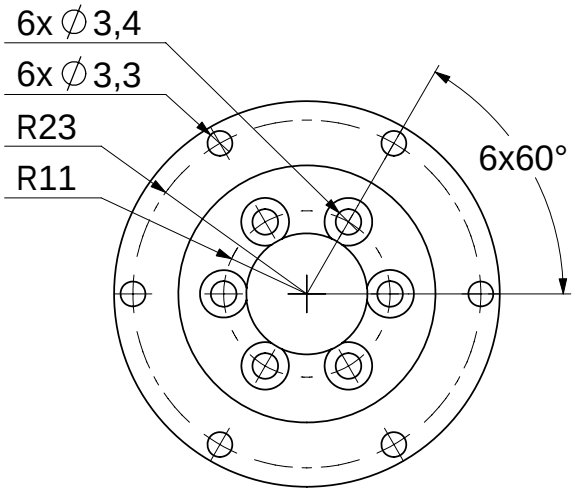
Aalborg Universitet
DMS10
Grp.43A, Pon103

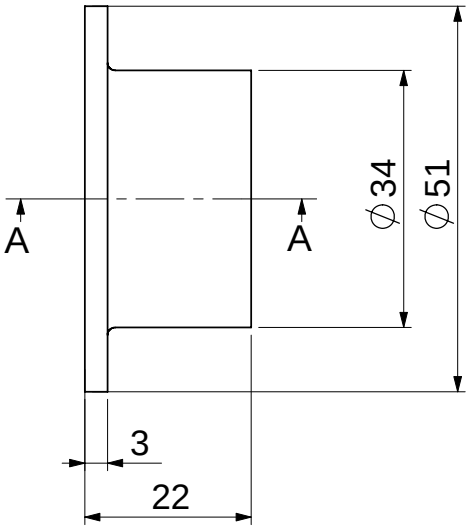
SIZE	DWG. NO.	REV.
A	MotorMounts	
	WEIGHT:	SHEET 1 OF 6



SECTION A-A







Note:
For ikke tolerance specificeret mål benyttes DS/ISO 2768-m

MATERIAL Al

Number 1 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

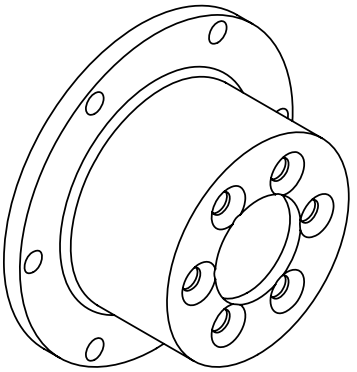
COMMENTS:

16.1 MotorMount 60W

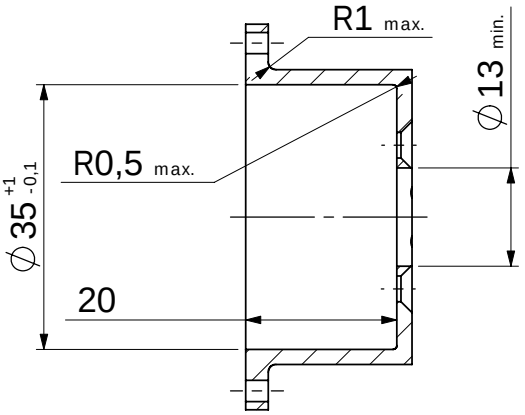
Aalborg Universitet
DMS10
Grp.43a Pon.103

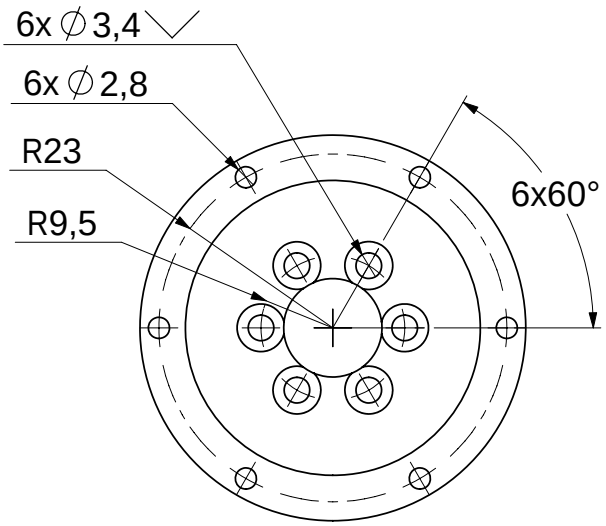
SIZE A | DWG. NO. MotorMounts | REV.

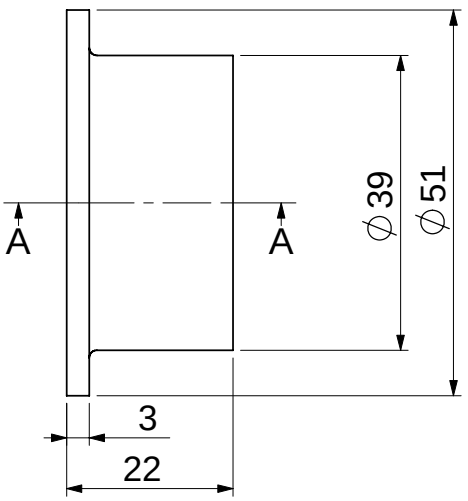
WEIGHT: | SHEET 2 OF 6



SECTION A-A







Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

MATERIAL: Al

Number: 1 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

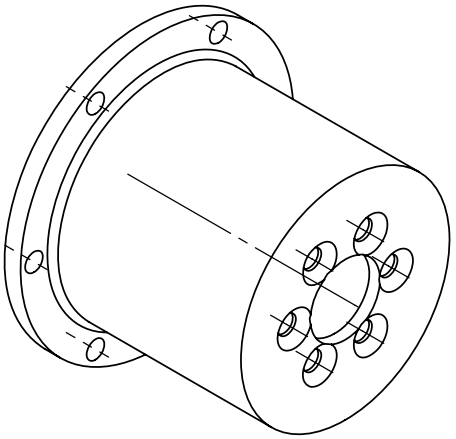
Forventet færdig:

COMMENTS:

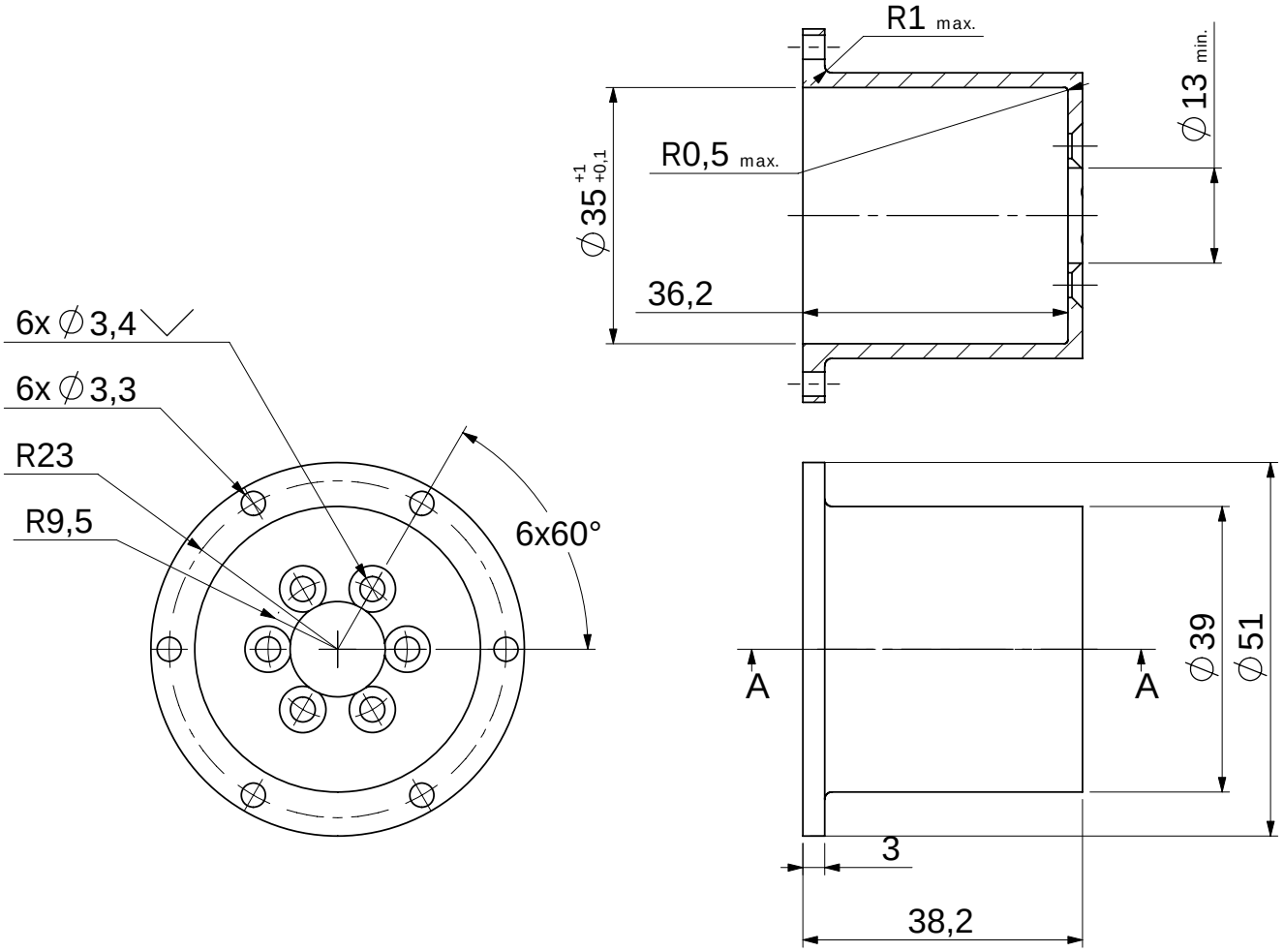
16.2a MotorMount 90Wa

Aalborg Universitet
DMS10
Grp.43A, Pon103

SIZE	DWG. NO.	REV.
A	MotorMounts	
	WEIGHT:	SHEET 3 OF 6



SECTION A-A



Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

MATERIAL: Al

Number: 1 stk.

SCALE:1:1

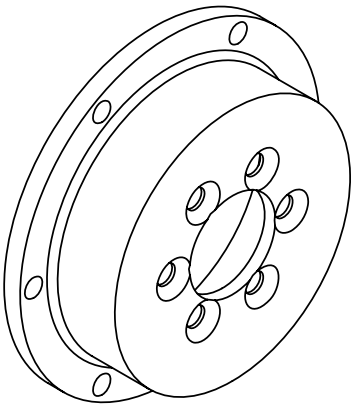
16.2b MotorMount 90Wb

Aalborg Universitet
DMS10
Grp.43A, pon103

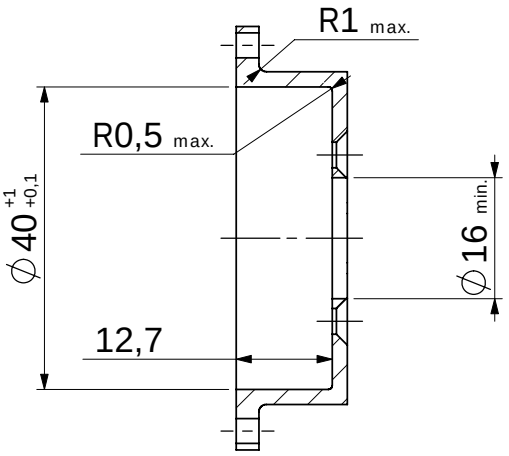
SIZE	DWG. NO.	REV.
A	MotorMounts	

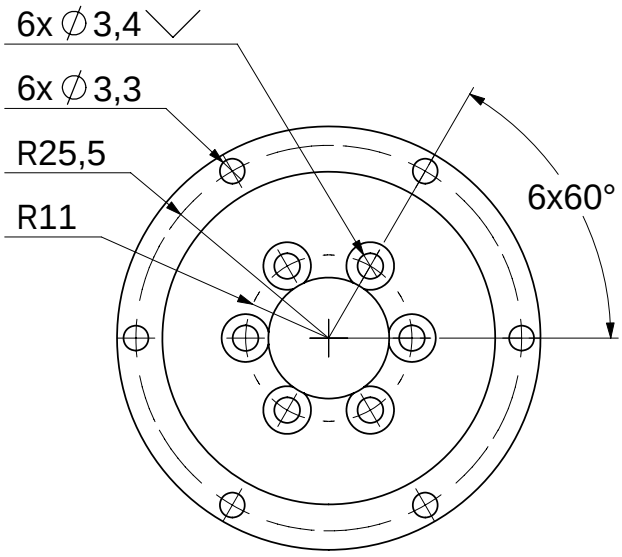
WEIGHT:

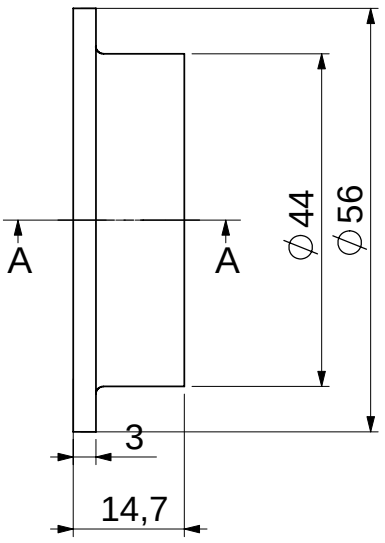
SHEET 4 OF 6



SECTION A-A







Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

MATERIAL: Al

Number: 2 stk.

SCALE:1:1

	NAME	DATE
DRAWN		
CHECKED		

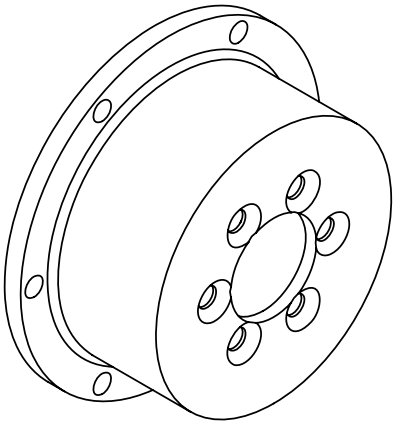
Forventet færdig:

COMMENTS:

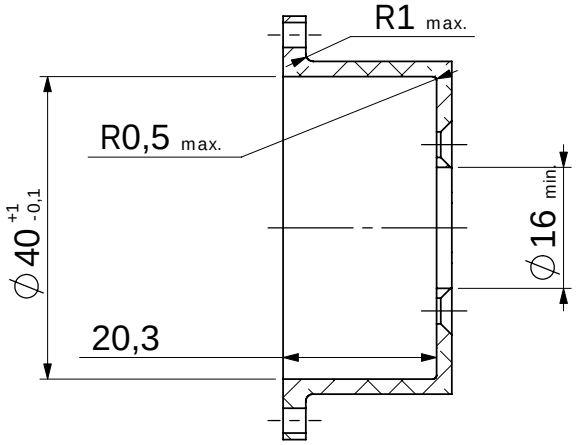
16.3a MotorMount 150Wa

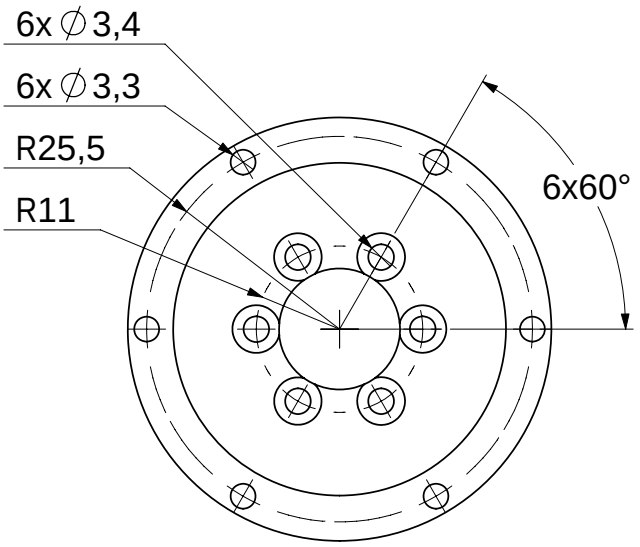
Aalborg Universitet
DMS10
Grp.43A, Pon103

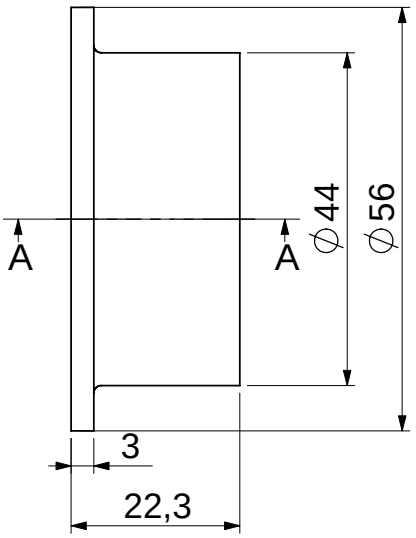
SIZE	DWG. NO.	REV.
A	MotorMounts	
	WEIGHT:	SHEET 5 OF 6



SECTION A-A







Note:
For ikke tolerance specificeret mål.
Benyttes DS/ISO 2768-m

	NAME	DATE
DRAWN		
CHECKED		

Forventet færdig:

COMMENTS:

MATERIAL: Al

Number: 6 stk.

SCALE:1:1

16.3b MotorMount 150Wb

Aalborg Universitet
DMS10
Grp.43A, Pon103

SIZE	DWG. NO.	REV.
A	MotorMounts	

WEIGHT:

SHEET 6 OF 6