Design of a Visual Servoing System for a SCARE type Robot to Perform a Placing Operation

Master Thesis

Author:
Thomas Pank ROULUND

Supervisor:
Ole MADSEN
Rasmus Skovgaard ANDERSEN

AALBORG UNIVERSITY
STUDENT REPORT

http://www.m-tech.aau.dk/
Electro-Mechanical System Design
Fibigerstræde 16
Telefon 99 40 71 17
Fax 99 40 71 10
http://info@m-tech.aau.dk

Title:

Design of a Visual Servoing System for a SCARE type Robot to Perform a Placing Operation

Theme:

Master Thesis

Project period:
P10, EMSD4

Project group:
4.119f

Authors:

Thomas Pank Roulund

Supervisors:

Ole Madsen

Rasmus Skovgaard Andersen

Printed copies: 1

Pages: 87

11 annex and 1 appendix

Completed: 02.06.2017

Synopsis:

This project covers the design of a visual servoing system for a SCARA type robot. The project is made based on a desire from Aalborg University to develop a visual servoing system. It was decided the system should be implemented on AAU's Smart Lab where its' purpose is to assist placing the lid on the housing of a demo assembly. The system is made to be configurable and efficient, which means the visual servoing system is operating without knowledge about the position, orientation or velocity of the housing. Additionally the lid should be placed on the housing, while the housing is moving past the robot. The vision system is using blob detection to derive the position and orientation of the housing, by converting the RGB image from an USB camera to the HSI colorspace. This is done to make the vision system more robust towards illumination changes. A Kalman filter is used to predict the future position and velocity of the housing, and this is used as an initial reference for the robot. Once this reference was tested on a emulated version of the robot, it showed that the robot controller is not able to track the reference without a steady state error. To improve this PI controllers are implemented to determine a new position and velocity reference for the robot, which reduced the tracking errors such they are now lower than the maximum allowed errors. Because the robot's is not fitted with an suction cup as end-effector, it is not possible to verify physically that the robot places the lid on the housing, but measurement from actual robots shows it is able to track the reference and move in while pretending to place the lid. At the time of the placement, it is verified that the position errors are below the maximum limit.

Resúme

Formålet for denne rapport er at designe et visuelt servosystem med hensigt på at regulere en SCARA robot. Opgaven er skrevet på baggrund af et ønske fra Aalborg Universitet om af udvikle et sådant system, og problemanalysen arbejder på at undersøge om det var muligt at benytte et visuelt servosystem til at assistere deres Smart Lab. Analysen konkluderede at et visuelt servosystem kunne bruges til at hjælpe med placeringsopgaver, enten ved at placere en kasse på en palle eller placere et låg på kassen. Det var ligeledes et krav fra Aalborg Universitet at dette system skal være både konfigurerbart og effektivt, hvilket ledte til kravene, at systemet skal fungerer uden information omkring placering eller orientering af pallen/kassen. Ligeledes var pallen placeret på et transportbånd som systemet heller var ikke tilladt at vide, hvor hurtigt bevæger sig. Der blev slutteligt forslået en løsnings strategi til at løse denne opgave, og det ledte til problemformulering:

Er det muligt at benytte den visuelt servosystem designforslag sammen med SCARE type robotten sådan at the resulterende visuelt servosystem er i stand til at følge et objekt på transportbåndet og udfører en placeringsopgave?

Opgaven afgrænses derefter til at kigge på placeringen af låget på kassen, til at starte med designe systemet til at følge en blå kasse, så der er bedre kontrast mellem transportbåndet og kassen.
Da robotten styres af en industriel kontrolboks, som er et lukkede system, det var derfor nødvendigt at undersøge hvordan robot bedst styres og hvilken type reference MATLAB skal sende til kontrolboksen. Denne analyse kommer frem den eneste mulighed for at styre robotten således at robotten ikke hakker i sin bevægelse, er ved konstant at opdatere den maksimalt tilladte hastigheden og positions referencen.
For at finde denne positions reference var det først og fremmest nødvendigt at udvikle et system, der på baggrund af billeder fra et USB kamera, var i stand til at identificere position og orientering af kassen på transportbåndet. Dette gøres ved blob detektion og RGB billedet konverteres til HSI farvespektrummet da dette er robust når det kommer til ændringer i belysningen. Selve referencen til robotten bliver bestemt ved at opsætte et Kalman filter til at forudsige positionen af kassen ét tids skridt fremme. Kalman filteret var også i stand til at estimere hastigheden af kassen. Det viser sig at kontrolboksen ikke var i stand til at få robotten til at følge denne reference uden en for stor positionsfejl, derfor opsættes en kontrolstrategi, hvor hastighedsreference bestemmes på baggrund af positions fejlen og positionsreference bestemmes ved hjælp af positionen fra Kalman filteret samt hastigheden af kassen. Det blev besluttet at bruge PI-kontrollere og disse tunes iterativt ved hjælp af en emulerede model af robotten. Med denne kontrolstrategi var styringsboksen nu i stand til at få robotten til at følge kassen og holde positions fejlen under den maksimalt tilladte fejl. Derfor blev den samme strategi implementeret på den rigtige robot, og de samme resultater blev opnået. Dog var robotten ikke monteret med en sugekop da forsøgene blev udført, så det var kun muligt at konkludere at i det øjeblik robotten fiktivt ligger låget på kassen, var placeringsfejlen under den maksimale fejl. Derfor blev det konkluderet at det forventes at det designede visuelt servosystem

opfylder kravsspecifikation men det det blev også konkluderet at det er nødvendigt med yderligere test for at kunne konkludere på, hvor robust løsninger er overfor ændringer i transportbåndets hastighed og retning.

Preface

This 4th semester master thesis is written by a student attending the Electro-Mechanical System Design engineering line at Aalborg University. The project period spans from the 2nd of February to the 2nd of June.

The references are created based on the Harvard method, meaning that each reference is written with both author and year of publish. All figures are named X.Y, where X shows the coherent chapter and Y the figure number. Whenever a table or figure is used a describing text follows to explain the context. If the figure doesn't include a reference its made by the group itself. The chapters in the annex and appendix are written with letters to differentiate between main matter and attachments. Matrices are noted as bold letters while vectors are noted with a vector arrow above the notation.
A .ZIP file is attached to the uploaded project, this includes the annex, appendix and all SolidWorks parts.
Software The following software has been used in the process of making this project.

Name	**Area of use**	**Name**	**Area of use**
LaTeX	Assembling the project	SolidWorks	3D drawing of the test setup
Maple 16	Calculations	Matlab	Visual servoing system and calculations

Thomas Pank Roulund

Contents

Chapter 1

Introduction

Visual servoing of a robot is the method of taking a picture, extract the features in this image and moving the robot accordingly, which is refereed to as look-then-move . The reason for why this report is working with this field, comes from some of the problems discussed in (Corke, 1996, p. 2). The arguments are that while robots are being increasingly used in the industry, the robots are still blind, and therefore there are some fundamental issues which visual servoing is able to help improve. The first problem is that robots require highly structured work environments, if for example the robot has to do a pick-and-place operation, the object which is being picked, needs to be at a well defined position and orientation. Secondly the robot needs to be "taught" how it should move and where certain objects are placed such that the robots avoids collision. Thirdly high speed robot needs to be mechanical designed such that the dynamic forces does not cause elastic deformation, which introduces a position error, if the robot is performing a task while moving at high speeds. The mechanical aspects of robot is a mature technology, while camera's and control units performance to price ratio is continuously improving. By adding visual servoing to a robot, its' configurability is also increased, because the time needed to teach the robot is reduced. Additional if going back to the example of the pick-and-place operation, the object can now be placed completely arbitrarily in a unknown work environment, since the visual servoing system it able to detect the object. Additionally it is stated that visual servoing is able to replace traditionally off-line trajectory plane, since the visual servoing system is able to react to the pose of the robot or the location of the end-effector and then act accordingly.
There is also organisations, whose purpose is to decide the best cause of action for the future robotic technology. One of these is SPARC who publish a Strategic Research Agenda (SRA) report of focus points for the robotic technology. In this report visual servoing is mentioned as a key technology since it enables better grasping of an object and identifying the workspace (SPARC, 2015, p. 261,281). Likewise robots should in general be more configurable, which as mentioned, is achieved with visual servoing.
Because of this, Aalborg University has a desire to develop a visual servoing system, and this leads to the initiating problem

Could one of Aalborg University's test setups benefit from a visual servoing system?

Chapter 2

Problem Analysis

Aalborg University are part of the industry 4.0 initiative and are dedicating research into this topic. Most of this is done on their Smart Lab, which is a modularised production line, shown on figure 2.1.

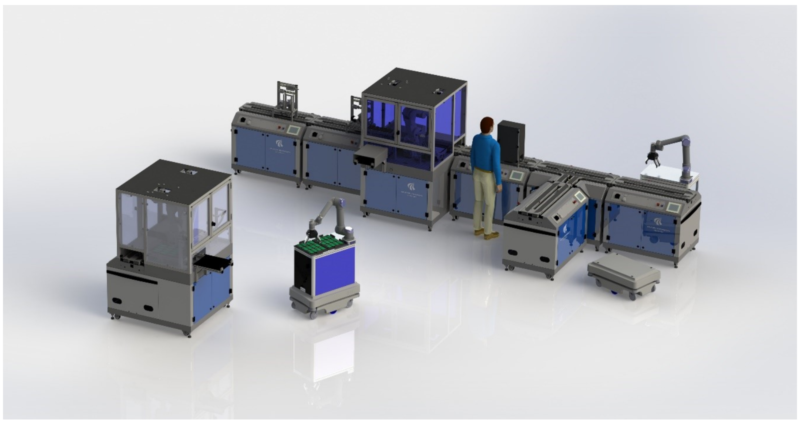


Figure 2.1: A concept model of the Smart Lab located at Aalborg University (Madsen and Møller, 2017)

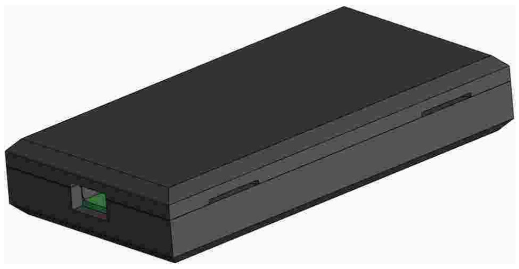
Each module is fitted with a conveyor belt, which if placed next to an other module, is able to transport an object on the conveyor belt to the neighbouring module. A key feature of this production line systems is, that it is highly configurable because of the modules. With this in mind, Aalborg University is interested in investigating if it is possible to develop a visual servoing system for a robot, which increases the overall configurability of the Smart Lab. The purpose of this chapter is to identify a process on the Smart Lab which can benefit from from a visual servoing system. Furthermore the test setup made available by Aalborg University is presented along with an solution strategy for the report.

2.1 Smart Lab

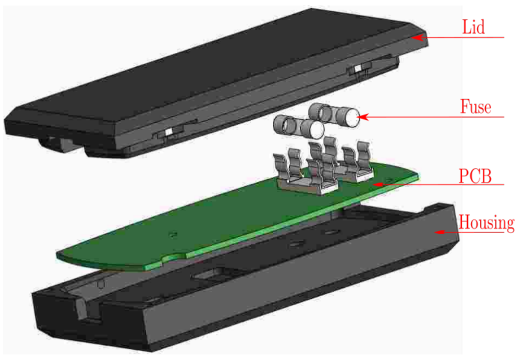
The Smart Lab is made for educational and research purposes, therefore it is also necessary to have an demo assembly to test production line. The purpose of this section is to analyse the Smart Lab and determine how a visual servoing system can be implemented on the Smart Lab. This is done in two parts, firstly the section presents the demo assembly's individual parts and secondly what is happening to the assembly at the different modules of the Smart Lab.

2.1.1 The Assembly

The demo assembly is made for testing the Smart Lab and consists of several components and require manufacturing. The demo assembly is shown on figure 2.2a and the assembly order is shown as a exploded view on figure 2.2b.



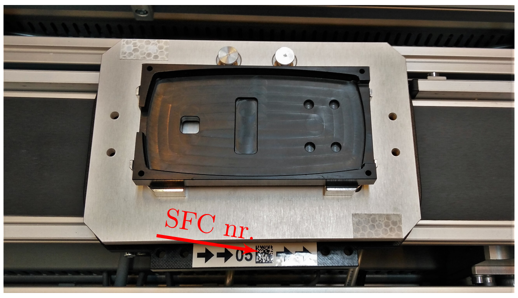
(a) The assembly



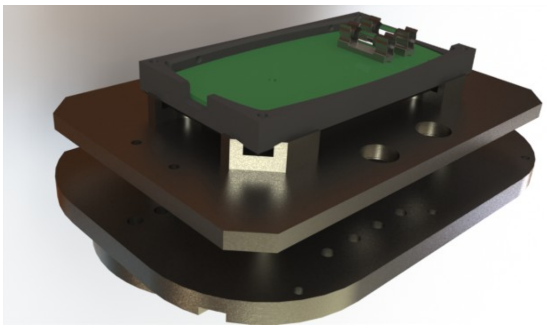
(b) Exploded view showing the assembly order of the assembly

Figure 2.2

The assembly consist of a housing, the PCB, two fuses which are inserted into a spring socket and a lid. The lid and housing comes in both black, blue and white and the color of these two needs to match. The first operation performed by the Smart Lab is placing the housing a pallet, which fixes the housing for remaining operations. The pallet is shown figure 2.3, which also shows how the housing is placed on the pallet.



(a) The pallet with a housing, showing the unique tag for the pallet



(b) The assembly on the pallet

Figure 2.3

Furthermore each pallet is equipped with both a unique Shop Floor number (SFC) which specifies the product on the pallet, this is information like color of housing etc. Secondly the pallet has a RFID chip, which keeps track of all the processes the product is undergoing. Each work station of on the Smart Lab is equipped with a RFID reader and writer, so it is possible to track a product live in the Smart Lab. The reason the pallets has these two features, is it enables mass customization since it is predetermined which work stations the pallet has to pass. The lid is designed with a hook such that it snaps together with the housing, when a certain force is applied to the lid. To ensure the lid snaps correctly, both the housing and lid have a fillet which guides the lid when the force is being applied, this fillet is shown on figure 2.4.

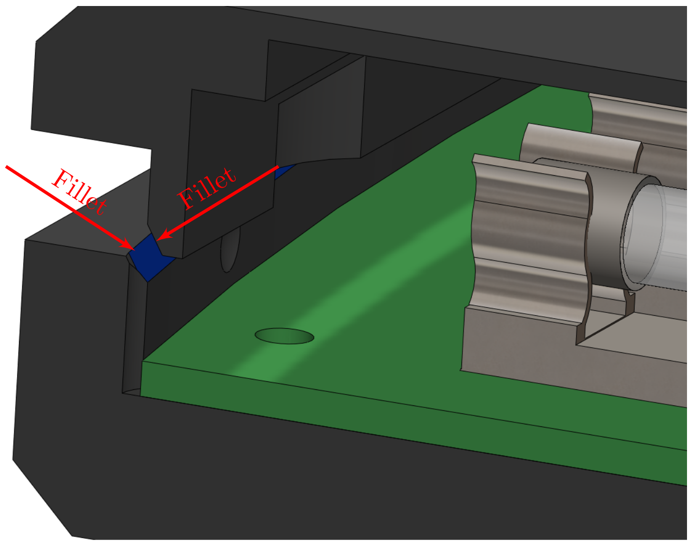


Figure 2.4: Cross section of the demo assembly showing the fillet, which purpose is to guide the lid onto the housing

These fillets also determine the placement tolerances for the lid, meaning acceptable offset the lid can have with respect to the housing, such that it still glides into the correct position. From the CAD model, the following three tolerances are determined:

$x_{\max}$ [mm]	$y_{\max}$ [mm]	$\theta_{\max}$ [deg]
± 1.7	± 1.7	± 2

Table 2.1: The placement tolerances for the assembly

These placement tolerances are maximum for the two directions and it should be noted that while the lid still slides into the correct position if $x_{ref} - x = x_{\max}$ and $y_{ref} - y = y_{\max}$ but if this is the case then it is necessary that $\theta_{ref} - \theta \approx 0$. The overall limits are therefore for this report defined as $x_{\max} = y_{\max} = \pm 1.5$ [mm] and $\theta_{\max} = 1^o$

2.1.2 Production Process

The Smart Lab concept also includes machines such as drones and mobile robots, these are outside the scope of this analysis and are therefore omitted. As mentioned the demo assembly undergoes several operations, including machining and assembling. Figure 2.5 shows a flow chart describing the smart lab process and which workstations the pallet passes.

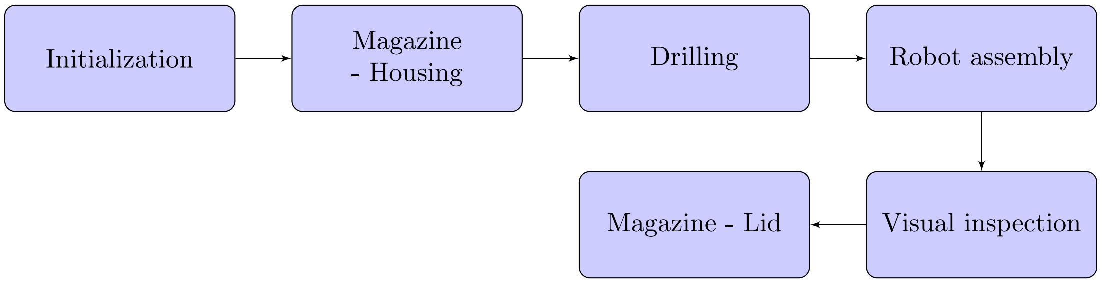


Figure 2.5: Process flow chart for the Smart Lab

A brief description of each block is given below, but common for all workstations, is the pallet is transported by the conveyor belt, until it reaches a stop block at the workstation. Once the workstation is done with its' operation, the stop block is lowered and the pallet continues.

- Initialization: The pallet is initiated and starts moving on the conveyor.
- Magazine - Housing: The pallet arrives at a magazine which drops a housing onto the pallet.
- Drilling: The pallet arrives at the drilling workstation, which performs the necessary machining.
- Robot assembly: A robot is responsible for inserting the PCB in the housing, likewise this robot also inserts the fuses in the sockets.
- Visual inspection: A camera inspects the product on the pallet, ensuring the PCB is correctly located in the housing, and the fuses are properly inserted in the sockets.
- Magazine - Lid: If the product passes the visual inspection, the finishing step is to place a lid on the housing, and the demo assembly is complete.

There are additional steps associated with the Smart Lab, but these are software steps associated with logging of workstation processes and tracking the pallets, these are therefore not relevant for this analysis. The physical location of the workstations on the Smart Lab are indicated on figure 2.6

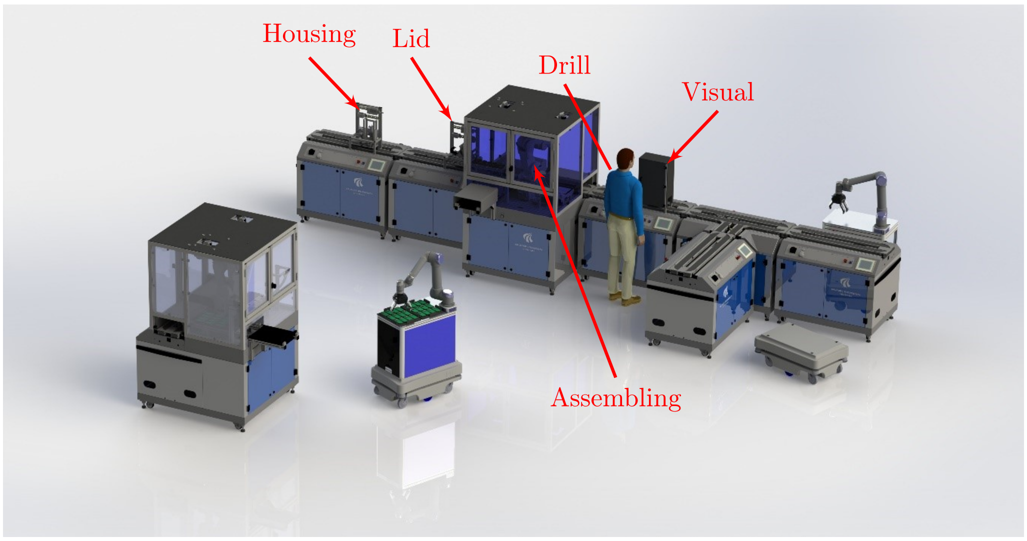


Figure 2.6: The Smart Lab's different work stations, the drilling station is not part of the CAD but is located at the marked location on the physical Smart Lab (Madsen and Møller, 2017, modified)

With the steps clarified, it is possible to investigate where a visual servoing system might be implemented. Neither the visual inspection or the drilling station has the need for an visual servoing system controlling a robot, since they both require the pallet to remain stationary while they perform their task. The remaining workstations are all related to placing parts on the pallet. Focusing firstly on the existing robot, the task of inserting the sockets is challenging for a robot, since it has to push with an insertion-force. This work station could benefit from an additional robot working in collaboration with the existing robot. Likewise the visual servoing could reduce the assembly time at this module, since the pallet would not have to stop inside the module. lastly the two remaining modules concerns the placing of the housing on the pallet and placing the lid on the housing. Both these task could also be done with a visual servoing system, by having the vision system detecting the position and orientation of the pallet/housing and placing the housing/lid accordingly. Keeping in mind the desire from Aalborg University, that the system should be both efficient and configurable, the visual servoing system should be made in such a way, that it is able to do the placement task without being connected to the Smart Lab, meaning no information about conveyor speed is available.
At this time, there is no room in the assembling module for installing an additional robot either inside or outside the module, it is therefore decided to focus on the placement of the housing/lid.

2.2 Test Setup

Because of the amount of active project on the Smart Lab, it is not possible to work directly on it. Instead Aalborg University has made a robot test setup available, which consist of a robot and a conveyor belt. This is used to emulating a robot located next to the Smart Lab. The purpose of this section is to present the robot which is used for the remainder of the report and specify the workspace around the robot. A CAD model of the test setup is made in SolidWorks and is available on the appendix CD.

2.2.1 The Adept Robot

The robot used for this test setup is a Adept Cobra s600 robot, shown on figure 2.7 and consist of three revolute joints and one prismatic. These joints are indicated on the figure with red lines.

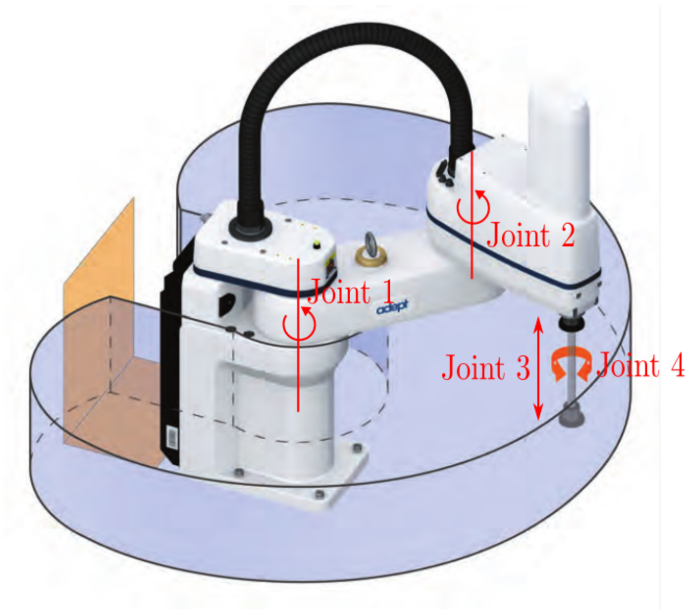


Figure 2.7: The available work space for the Cobra s600 (Adept, 2017, modified)

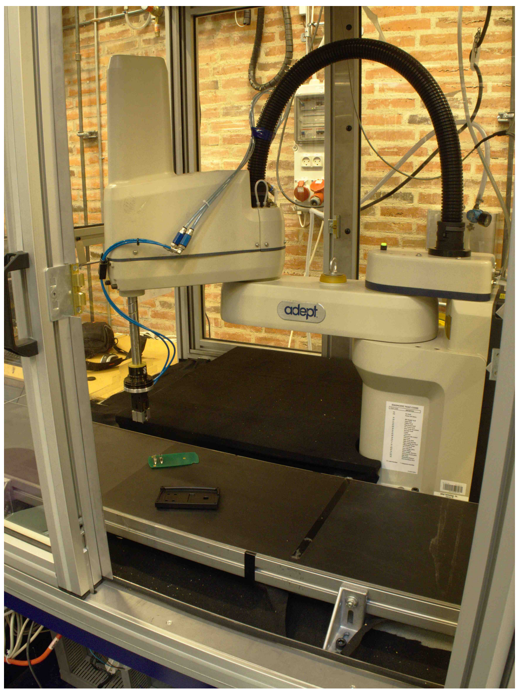
Joint 1	$\pm 105^o$
Joint 2	$\pm 150^o$
Joint 3	[0,210] [mm]
Joint 4	$\pm 360^o$
Inner radius	163 [mm]
Outer radius	600 [mm]

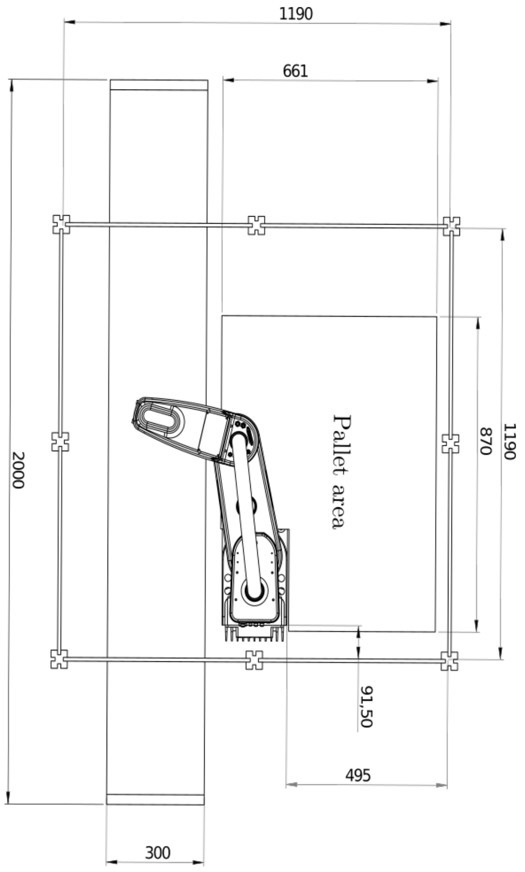
Table 2.2: Joint ranges for the Cobra s600 (Adept, 2017)

The robot's available workspace is determined from the joints limits, and this is indicated on figure 2.7 by the blue envelope. The joints' ranges are specified in the datasheet for the Cobra s600, and are summarised in table 2.2 along with the inner and outer radius of the workspace.

2.2.2 Work space

With the work space of the Cobra defined, it is now necessary to define the available workspace in the test setup. The Cobra is placed inside a cage next to a conveyor belt, shown on figure 2.8a. By making a CAD model of the workspace, it is possible to better understand possibilities and limitations of it. A plan view of this CAD model is shown on figure 2.8b, where only important features are included.


(a) Picture of the actual test setup



(b) Plan view of the work space created from the CAD model, all measurements are in millimetre. The CAD model of the Cobra s600 is available online (Adept, 2017)

Figure 2.8

It is worth noting, that the robot is able to hit the sides of the cage, if this happens, the robot will stop, since the doors in the cages are fitted with an emergency stop, so the robot is only operational when the doors are closed. The conveyor belt is controlled with a frequency converter, such that it is able to operate at different velocities.

2.3 Solution strategy

Visual servoing is a widely used tool and therefore it is possible to find inspiration about a system scheme which is able to solve the placing task the visual servoing system should perform. In (Corke, 1996, p. 154) a control scheme is presented, which is well suited for this application, and is called "Dynamic position-based look and move". The idea is a cascade control scheme, the inner loop is the internal control of the industrial robot and the outer loop is the visions detection of the housing. The method also introduces a predictor, which purpose is to predict the position of the housing on the conveyor belt one sample ahead. The reason for this is, that if using the current position of the housing, the robot then moves its' end-effector to this position, but now the housing is further ahead. Therefore by predicting the future position, it proposed the robot tracks the housing better. This idea is shown on the block diagram on figure 2.9.

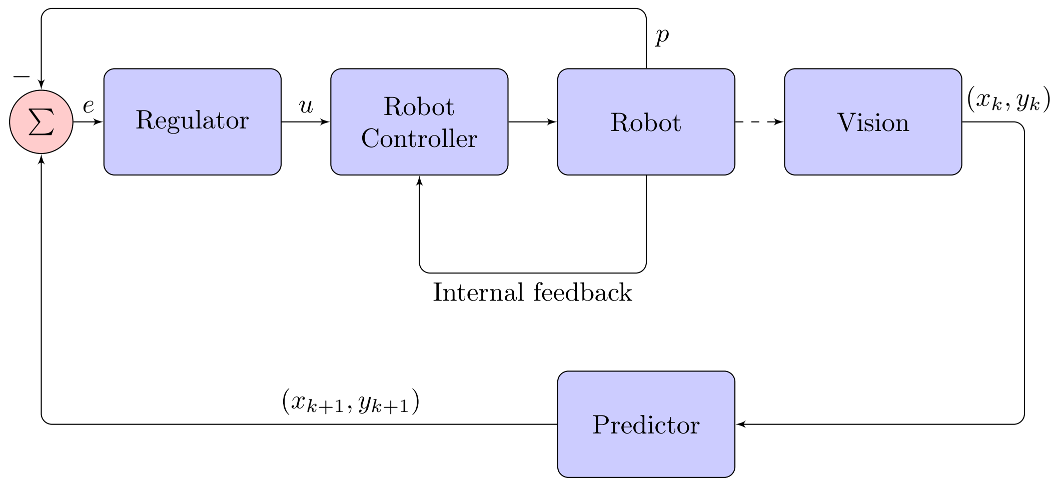


Figure 2.9: The proposed visual servoing control scheme

The notation on the figure is, that p is the position of the robot, (x_k, y_k) is the current position of the housing on the conveyor belt and (x_{k+1}, y_{k+1}) is the predicted future position. The error, e, is then defined as the difference between robot's position and the housing's which a regulator uses to determine a reference for the robot controller. The internal loop consisting of the robot controller and robot is considered as a black box.

It is now been determined that the best operation for a visual servoing system on the Smart Lab is helping with the placing of either the lids or the housing. For this a test setup is made available which has a SCARA type robot and a conveyor belt. Lastly a strategy for how the visual servoing system should be made is proposed.

Chapter 3

Problem Statement

The analysis made in chapter 2 yielded a clear task and solution strategy, which is formulated into the problem statement the remainder of this report focuses on solving.

Is it possible to use the proposed visual servoing scheme together with the SCARE type robot such that the resulting visual servoing system is able to track an object on conveyor belt and perform a placing operation?

3.1 Project delimitation

It is decided to focus on the placing a lid on the housing, such that a buffer of lids is placed inside the test setup and the vision system has to detect position and orientation of the housing. Because the focus is to develop a visual servoing system, it is decided only to use the blue housing, since there is better contrast between this color and the black conveyor belt. Likewise only one housing will be placed on the conveyor time. Lastly the system should be robust towards changes in conveyor speed, but if these changes happens close to when the visual servoing system is placing the lid, the system might not be able to react to the changes before releasing the lid. In this case it is acceptable that the systems fails to place the lid correctly. Lastly the lid connects to the housing with a snap connection, the task is reduced to place the lid, such that it is located correctly on the housing, and only has to be pushed for the snaps to engage.

3.2 Requirements

The following are the requirements for the vision system:

1. The total placing error of the lid on the housing from the visual servoing scheme's three modules; Vision, Prediction and Regulator must all be lower than the maximum offset defined in section 2.1.1 to $x_{\max} = y_{\max} = 1.5$ [mm] and $\theta_{\max} = 1^o$
2. The end-effector should track the housing smoothly, to avoid oscillations.
3. The vision system needs to be able to differentiate between the housing and other objects entering the camera's region of interest, for example, the robot.
4. The vision system needs to be robust towards changes in illumination in the workspace.
5. The visual servoing should operate without any information about conveyor speed or initial position and orientation of the housing

6. The predictor needs to converge within 2 [s], this is based on the robot needs to be able to track the housing in case it leaves the available workspace of the robot.
7. Because the robot has to be able to place the lid on the housing before the housing leaves the robot's available work space, the regulator needs to bring the robot to the steady state tracking value within 2 [s].

Chapter 4

Architecture

The Cobra s600 comes with a robot controller, which is designed by Adept and makes it possible move the robot to a reference in either the Cartesian- or Joint space. The robot controller is connected to a computer via an Ethernet cable and software for the controller is made in a Adept program called ACE. The visual servoing system is made in MATLAB and it is therefore necessary to set up the connections such that MATLAB is able to get all the needed information and send the necessary control signals to the robot controller. These connections are shown on figure 4.1

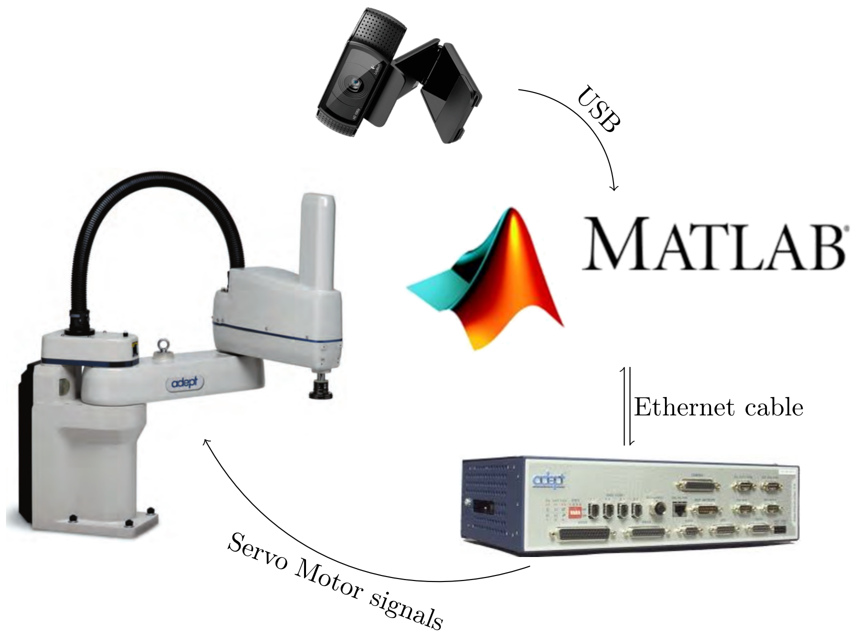


Figure 4.1: The connection between the hardware and software used in the visual servoing system (Adept, 2017) (Logitech, 2017)(Mathworks, 2017)

The purpose of this chapter is analyse the connection between MATLAB and the robot controller and derive the best approach for operating the robot through MATLAB.

4.1 Robot Architecture

An Industrial robots such as the Adept Cobra s600 is high performance high precision machine, and comes with a control box and programmed with software develop by Adept. This does create some limitations since the software is not open, which means the user is only able to change a limited amount of settings on the robot. The purpose of this section is to analyse what information u should contain.

4.1.1 Adept ACE

Adept ACE comes with a fully integrated programming language, but since the visual servoing system is made in MATLAB, it is necessary to set up a communication between MATLAB and ACE. The communication protocol is achieved by using a unique IP address and opening a communication port, which MATLAB can then send a string to, which ACE interoperates as u. The flow chart on figure 4.2 shows the logic behind this communication.

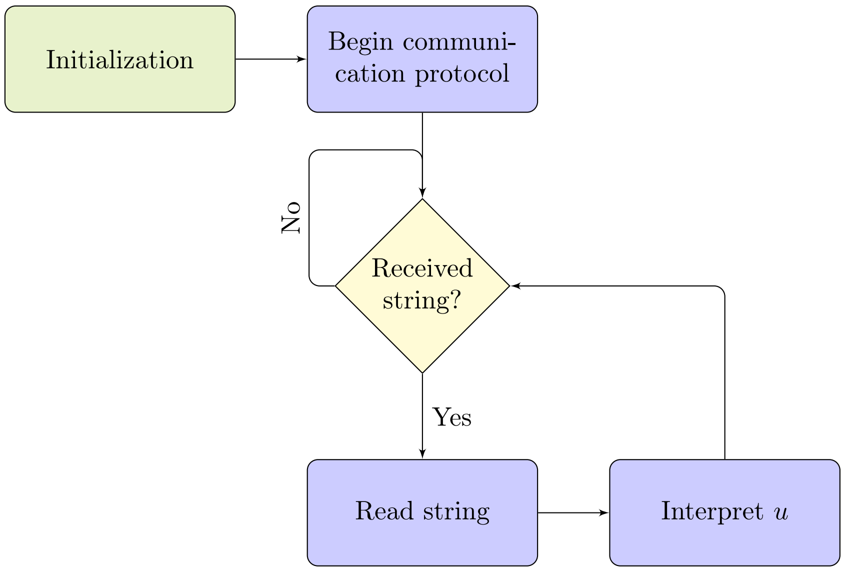


Figure 4.2: Flow chart describing the logic of the communication protocol

To specify what u is, it is necessary to investigate how to robot controller plans to move from point to point.

In robotics, it is common to operate in two spaces; the Joint space and the Cartesian which is liked by the robot's kinematics. The default movement from the robot current orientation to a point is done using the bang-coast-bang approach. This technique is illustrated on figure 4.3. (De Luca, 2017)

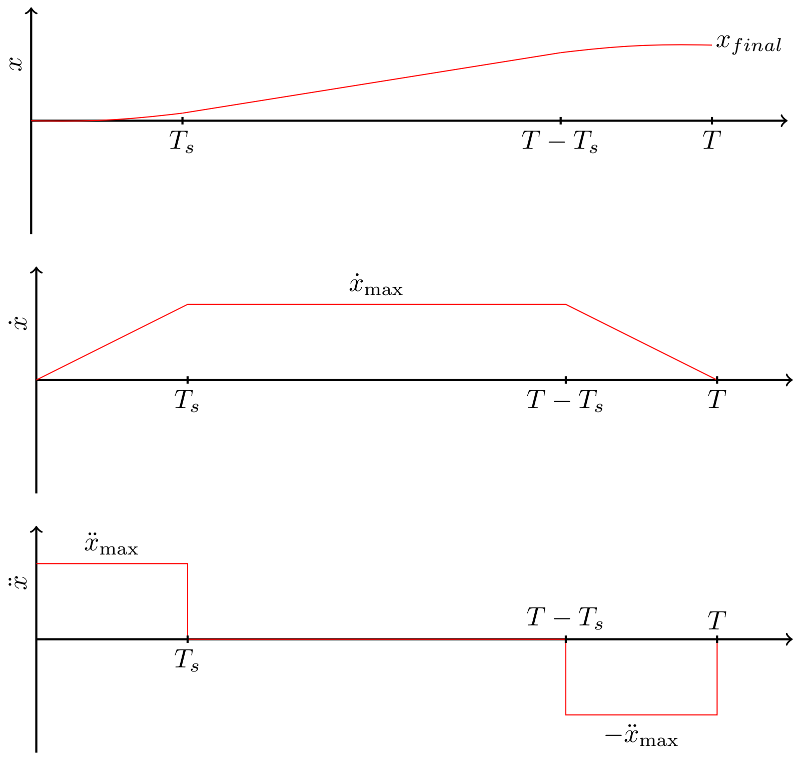


Figure 4.3: The bang-coast-bang technique

There T_s is the rise time and T is the total time, these are defined as: (De Luca, 2017)

$$T_s = \frac{\dot{x}_{\max}}{\ddot{x}_{\max}} \tag{4.1.1}$$

$$T = \frac{x_{final}\,\ddot{x}_{\max} + \dot{x}_{max}^2}{\ddot{x}_{\max}\,\dot{x}_{\max}} \tag{4.1.2}$$

$$x_{final} > \frac{\dot{x}_{\max}^2}{\ddot{x}_{\max}} \tag{4.1.3}$$

Condition (4.1.3) is the coasting condition, if this is violated, then the motion becomes, bang-bang which means there is no coasting. A good solution for having the robot track and object would be to determine an appropriate velocity for each joint and then continuously updating these velocities. This is not available on the robot controller, where it is only possible to set an maximum acceleration and velocity. The problem with the bang-coast-bang technique is the acceleration and deacceleration causes unwanted vibrations. Tests shows, that it is possible to achieve continuous motion if the destination, x_{final}, changes before the robot reaches it, it is therefore necessary to ensure x_{final} is updated before $T - T_s$. The string which MATLAB sends should therefore contain a position reference and a maximum speed for the robot controller:

$$\text{str} = \begin{bmatrix} \text{M} & X & Y & Z & 180 & 180 & \theta & \dot{p} \end{bmatrix} \tag{4.1.4}$$

The two times 180 is necessary because of syntax in ACE, but is not used for anything. The parameter M is the master, which chooses what ACE should perform, once it receives a signal, where the two cases in focus for this report, is getting ACE to send the time stamp and end-effector position back to MATLAB and updating the maximum speed and the point the robot is currently moving towards. The sequence diagram on figure 4.4 shows how this is done.

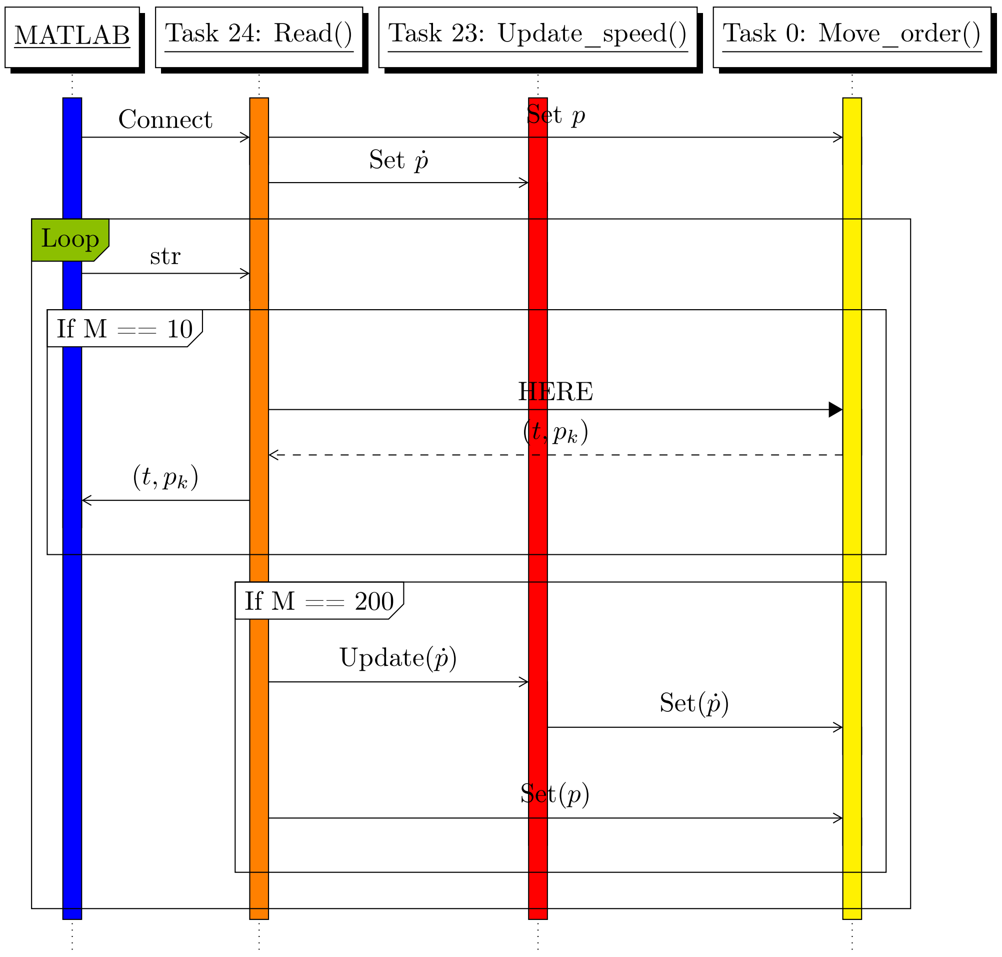


Figure 4.4: Sequence diagram for control of the robot

The robot controller supports 24 different task, which if using common micro processor thermology, are interrupts with a fixed frequency. It proved necessary to use three different task to get the desired performance. A description of these are:

- Read(): Handles the communication with MATLAB and interoperates the string, str, such that it reads the master value. The "HERE" command reads the current position of the end-effector and this is then sent to MATLAB along with a time stamp.
- Update_speed(): This task only function is to receive the maximum velocity reference, and set it as the current maximum velocity.
- Move_order(): This task is continuously ordering the robot controller to move the robot to the reference point. This ensures that when a new reference is set, the robot will automatic move towards it, even if it did not finish moving to the previous reference point

The control signal for the robot controller, u, is therefore actually both a position and velocity reference for the robot. This reference is either a point in the Cartesian or joint configuration vector, the only difference between the two, is whether the robot controller needs to do the inverse kinematics or MATLAB does it.

Chapter 5

Vision System

Vision systems are associated with heavy computational needs, since most operations require to inspect the value of or modify every single pixel in the image. This is an important aspect when using visual servoing, since the vision system might introduce a delay in the control approach, shown on figure 5.1, if the computation takes longer than the update rate of the camera.

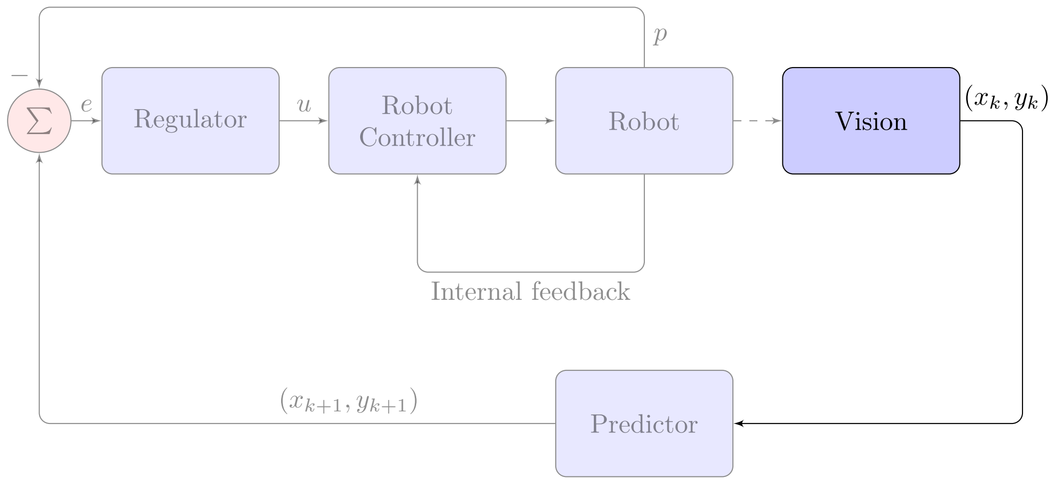


Figure 5.1: The current part of the control approach proposal

The vision system designed in this report, is therefore made using a MATLAB toolbox developed by Peter I. Corke to perform the necessary image manipulations to minimize the computation time (Corke, 2017). The purpose of this chapter is to develop a vision system which detects the position and orientation of the housing on the conveyor belt, as is necessary for the proposed control scheme. This is done by presenting the camera used for the vision system along with a model of the camera. Afterwards a description of the image manipulations steps necessary to mathematically derive the position and orientation of the housing is made. The vision system is develop using MATLAB and the script is available in annex A.2.1.

5.1 Camera

Cameras are often assumed to be perfect pinhole cameras, but because of manufacturing this is never the case. it is therefore necessary to perform calibration tests of the camera, such that the distortion can be quantified. The purpose of this section is present the camera used for the visual feedback and to determine the intrinsic and extrinsic parameters of the camera.

5.1.1 Logitech C920

The USB camera made available for the test setup is a Logitech C920, which upper operation limits are 30 frames per second (FPS) and 1920x1080 resolution, an image of the camera is shown on figure 5.2.

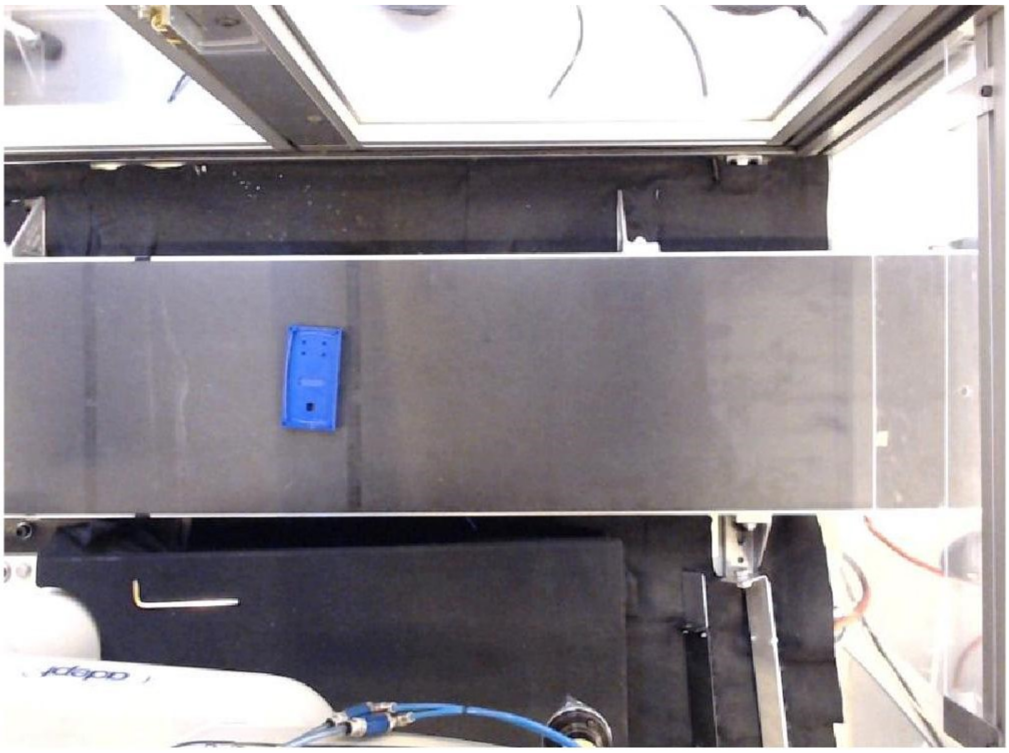
Figure 5.2: The USB camera used in the test setup (Logitech, 2017)

The camera also comes with an auto focus function and auto light adjustment. It is possible to turn these features off though MATLAB, and it is decided to turn off the auto focus since the housing moves in the plane, and change of focus settings also changes the intrinsic parameters. The auto light adjustment is left turned on, since the workspace around the test setup both have fluorescent lighting and sky windows. The lighting conditions in the test setup is therefore never constant. The way to define the camera in MATLAB is done with the following code:

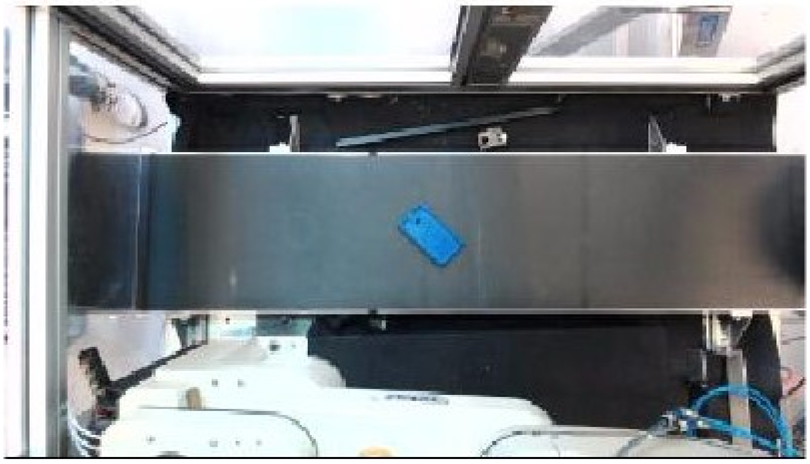
```matlab
% Define video source
        vid = videoinput('winvideo', 2, 'MJPG_320x180');
% Define camera settings structure
        src = getselectedsource(vid);
% Define number of images
        vid.FramesPerTrigger = 1;
        vid.TriggerRepeat = Inf;
% Define frame rate
        src.FrameRate = '30.0000';
% Set focus mode
        src.FocusMode = 'manual';
% Define fixed focus
        src.Focus = 0;
% Create videoplayer object
        hvpc = vision.VideoPlayer;
% Frames is stored in memory
        vid.LoggingMode = 'memory';
% USB camera is enabled
        start(vid)
% Image and time stamp is stored
```

```
        [im, time] = getdata(vid,1,'double');
```

In this script, the resolution is chosen to 320x180, the choice of resolution is a compromise between computation time and quality of the housing. The quality is deemed to low if the error between centroid derived in section 5.2.1 and the actual center of the housing is to large. Two different resolutions are shown on figure 5.3.



(a) Frame from the test setup at 640x480 resolution. Scale is reduced to 50% of original size



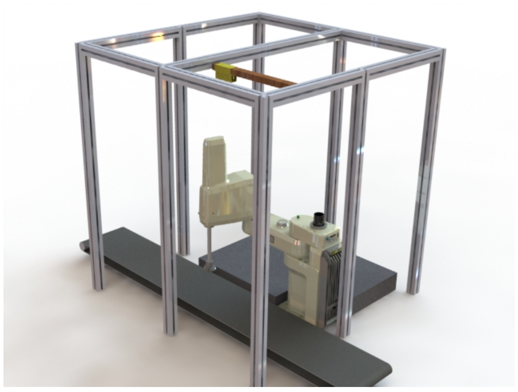
(b) Frame from the test setup at 320x180 resolution

Figure 5.3

The low resolution frame is more pixelated than the higher resolution frame, but there the number of pixels are also significantly lower:

$$\frac{640 \cdot 480 - 320 \cdot 180}{640 \cdot 480} = 0.81 \qquad (5.1.1)$$

Since computation time is an important factor, it is desirable to use the lowest resolution and increase it, in case it turns out to be insufficient.
Camera for visual system are usually either attached to the robot, hand held, or mounted with and overview of the region of interest. If using a hand held camera, it is necessary to compensate for the movement of the robot. Since the main aspect of the report is to develop visual servoing, it is decided to initially keep the vision system simple, by mounting the camera above the conveyor belt. The is done by designing and 3D printing a housing. The housing fits on a bar and is mounted as shown on figure 5.4a and a cross section view of the housing is shown on figure 5.4b.


(a) The test setup with the camera suspended above the conveyor belt

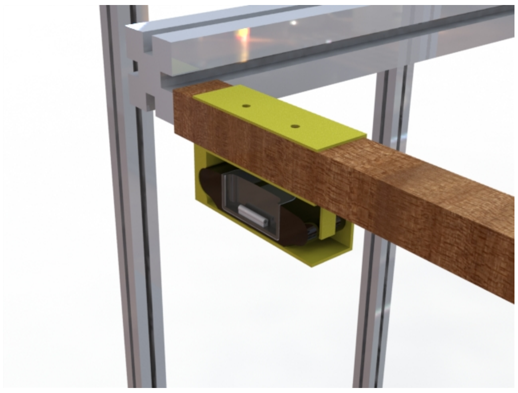

(b) Cross section of camera housing

Figure 5.4

The idea of this mounting is to ensure the stiffness of the beam, such that when the robot is moving, the camera is not experiencing any vibrations. If the camera is vibrating, it will cause significant noise on the vision system.

5.1.2 Camera Modelling

When using a camera for vision systems, it is necessary to compensate for the lens not being perfect. Because lenses are not perfect, they distort images with a fisheye effect, which needs to be removed from an image. This is done by determining the intrinsic parameters of the camera, these are the focal length in the horizontal and vertical direction in the image and true middle of the image, called the principal point. Furthermore there are also a skew factor and five distortion factors.
If a global point consist of three coordinates $\begin{bmatrix} X & Y & Z \end{bmatrix}^T$, the approach for determining the pixel values is firstly to project the point onto the image plane as shown on figure 5.5

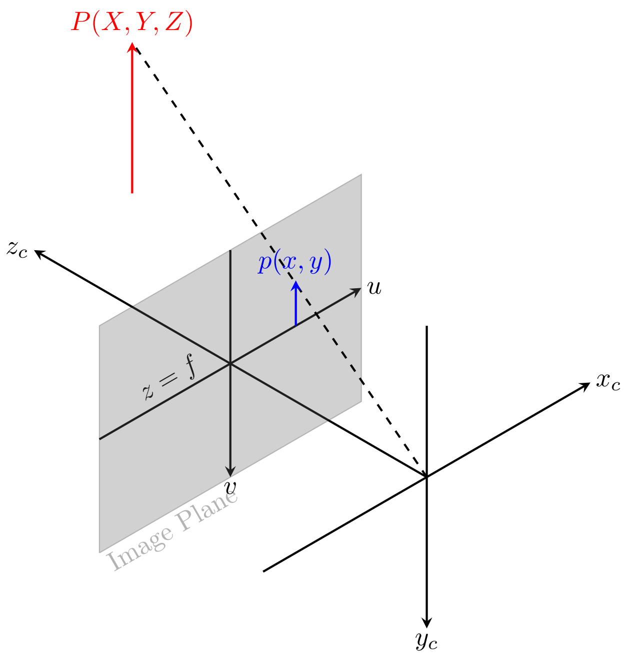


Figure 5.5: Projection of point onto the image plane

This is done by defining the normalised pinhole image projection: (Caltech, 2017)

$$\vec{x}_n = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \end{bmatrix} \tag{5.1.2}$$

Next it is necessary to add the distortion, these consist of radial- and tangential distortion. The radial distortion is the fish eye effect, which causes straight lines to become curved lines on the image plane, and is defined by three factors r_{di}. The tangential distortion is from the image sensor within the camera not being perfectly aligned with the lens, this is defined by two factors t_{di}. The distortion is added with the following expression:

$$\vec{x}_d = (1 + r_{d1}\, r^2 + r_{d2}\, r^4 + r_{d3}\, r^6)\vec{x}_n + \vec{dx} \tag{5.1.3}$$

$$r = x + y \tag{5.1.4}$$

$$\vec{dx} = \begin{bmatrix} 2\, t_{d1}\, x\, y + t_{d2}(r^2 + 2\, x^2) \\ t_{d1}(r^2 + 2\, y^2) + 2\, t_{d2}\, x\, y \end{bmatrix} \tag{5.1.5}$$

Where x_d is the distorted point in the image plane and dx is the tangential distortion vector. It is now possible to use this point to determine the final pixel values, by using the so called camera matrix:

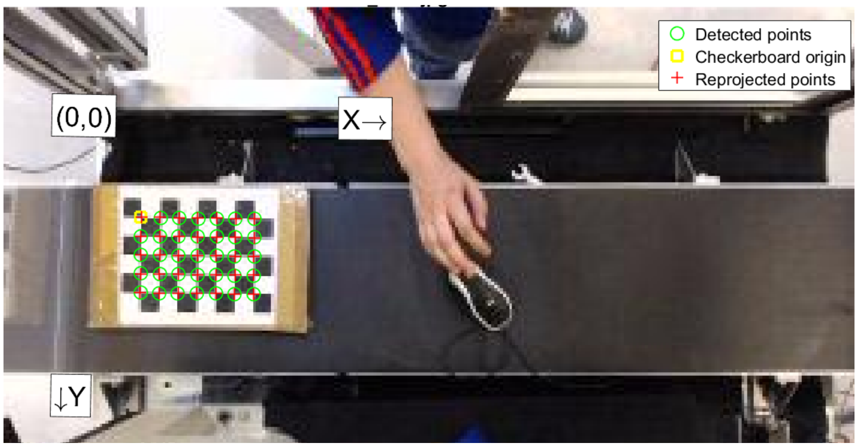
$$\mathbf{C} = \begin{bmatrix} f_x & \alpha & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.1.6}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{C}\, x_d \tag{5.1.7}$$

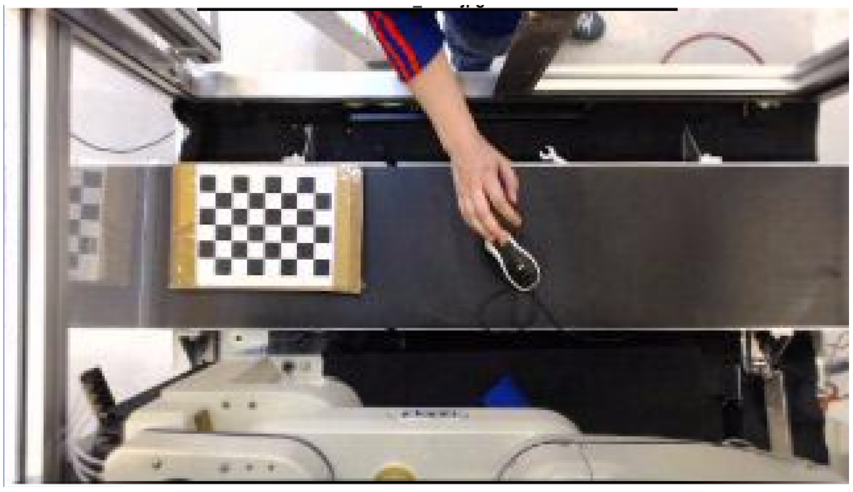
Where f_x and f_y are the focal lengths, u_0 and v_0 are the principal point, and α is the skew factor, which is non-zero if the pixels are not completely square. This concludes how a point is projected onto the image plane and, but in case of visual servoing, it is necessary to map an image point into some global coordinate system. For this the extrinsic parameters are needed, which are used to form a transformation matrix. (Corke, 2013)

$$^{C}\mathbf{A}_G \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} \tag{5.1.8}$$

The intrinsic and extrinsic parameters are determined with a toolbox. The one available in MATLAB requires around 20 images of a chessboard in different locations within the camera's field of view, at different positions and orientations. The toolbox then detects the intersect point between the checker squares, the size of the squares are defined before initialisation of the toolbox. The result of this is showed on figure 5.6a.



(a) The chessboard where all intersect points are detected



(b) The chessboard where the image is undistorted

Figure 5.6

Figure 5.6b shows the same image, where the undisort function is applied. The distortion is most significant near the edges of the image, but in general there is almost no distortion. A total of 31 images is used and figure 5.7 shows the location and orientation of all images with respect to the camera.

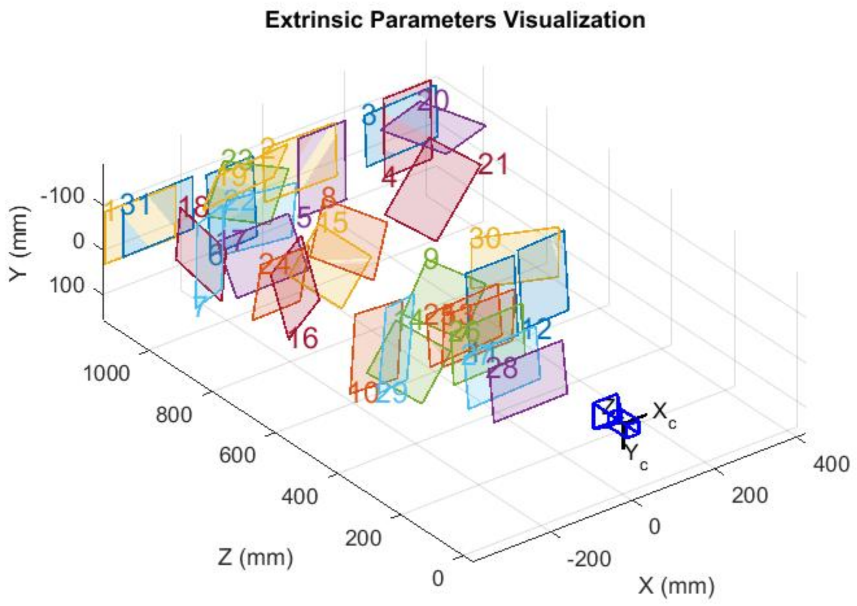


Figure 5.7: The position and orientation of the images used for determining the camera parameters

The images cover the whole conveyor belt, but there is also images close to the camera and at different orientations, which ensures the toolbox is able to estimate all the distortion factors and the skew factor. As seen on figure 5.6a the toolbox assigns a coordinate system for each detected checker board, and it is decided to use the coordinate system on the figure as the global coordinate system. The estimated distortion factors are listed in table 5.1

	1 [-]	2 [-]	3 [-]
r_{di}	0.0272 ± 0.0068	-0.1080 ± 0.0343	0.0307 ± 0.0562
t_{di}	0.0002 ± 0.0006	0 ± 0.0007	

Table 5.1: The C920 camera's five distortion factors

It is clear there is almost no tangential distortion, and also the radial distortion factors are low which is also apprent from the undistorted image on figure 5.6b. The factors in the camera matrix is estimated to be:

$$\mathbf{C} = \begin{bmatrix} 244.9928 \pm 1.377 \text{ [pixels]} & 0.0013 \pm 0.0494 & 164.2548 \pm 0.7613 \text{ [pixels]} \\ 0 & 245.2202 \pm 1.3689 \text{ [pixels]} & 88.8660 \pm 0.7130 \text{ [pixels]} \\ 0 & 0 & 1 \end{bmatrix} \quad (5.1.9)$$

The toolbox also calculates the extrinsic parameters for each image on figure 5.7. It requires solving equation (5.1.3) to obtain the pinhole coordinates. Because of the equation's order, it is desirable to solve this equation numeric, which MATLAB does with the command.

```
[x y] = pointsToWorld(cameraParams,camera_rotation,camera_trans,[u v])
```

Where the camera_rotation is a rotation matrix and camera_trans is projection vector and cameraParams is a structure with all the intrinsic parameters.

5.2 Image processing

The purpose of this section is to identify the location housing in a image. This is done in two parts, firstly by detecting the blob in the image and secondly calculating the centroid of the blob.

5.2.1 Blob detection

A single frame from the camera in the test setup is shown on figure 5.8.

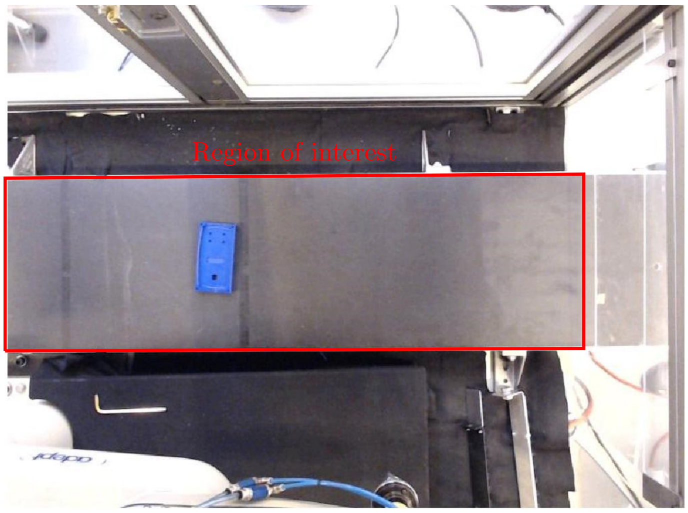


Figure 5.8: A single image from the camera in the test setup

First step is to define the region of interest, which is shown on the figure. This is done by determining the pixels spanning this region, and creating a new image containing all the pixels within the region of interest. The resulting image is shown on figure 5.9.

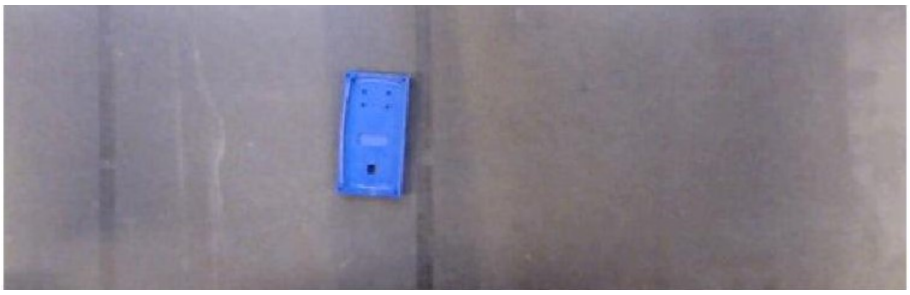


Figure 5.9: The region of interest, which is the conveyor belt

With the new image only showing the housing part and conveyor belt, the next step is to determine the difference between these two. The first attempt is background subtraction, the idea is, that if the background is static, then by subtracting it from the image, only the housing will remain. A base picture of the background is shown on figure 5.10.

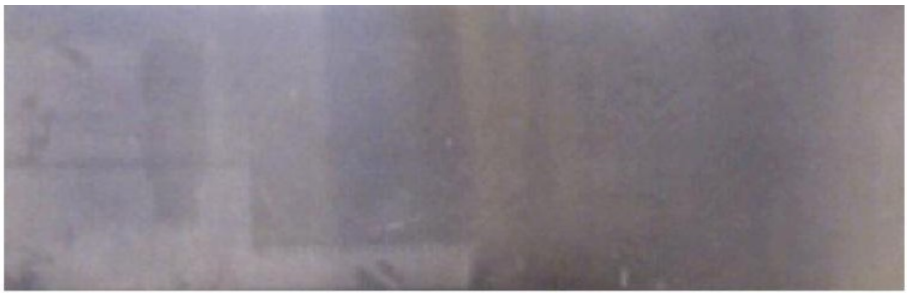


Figure 5.10: The base image, which only shows the conveyor belt

Comparing this base image with figure 5.9, a conflict with the assumption about static background arises. The conveyor belt's appearance is not uniform and there are tape lines from previously experiments, these might cause noise later in the vision system. Before the subtraction it is decided to work with greyscale images, where the frame and base are currently in the RGB space, this is done with a weighted summation

$$f(u,v) = \alpha\, R + \beta\, G + \gamma\, B \tag{5.2.1}$$

$$\alpha + \beta + \gamma = 1 \tag{5.2.2}$$

In this first attempt $\alpha = 1$ and $\beta = \gamma = 0$, which means only the red color channel is used. The background subtraction is done with the following equation:

$$g(u,v) = |f(u,v) - b(u,v)| \tag{5.2.3}$$

Where u is the pixel number in the horizontal direction, v the pixel number in the vertical direction, f is the frame and b is the base image, the absolute value is used, since it is likely some pixels' value might become negative and negative pixel values does not correspond to a nuance of grey. Figure 5.11 shows the result of subtracting the base image from figure 5.9.

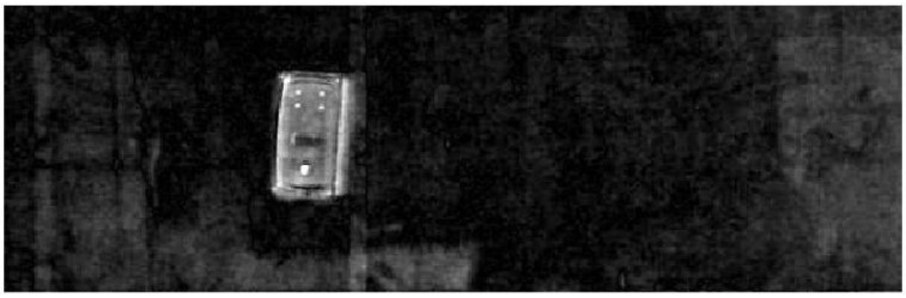


Figure 5.11: The resulting image after the base is subtracted from the frame, the frame is normalised to better see the details

The background subtraction did not manage to remove the entire background, which results in some additional grey areas in the frame besides where the housing is located. Next step is to try and remove these additional grey areas which is acting a noise in the frame. This is done by performing thresholding on the frame, which transform it from greyscale form into binary form. To determine which thresholding value to use, it is necessary to make a histogram for figure 5.11. The appropriate thresholding value is a compromise between false positive and false negatives, meaning backgrounds pixels which are set to 1 and pixels belonging to the housing set to 0. The histogram is shown on figure 5.12.

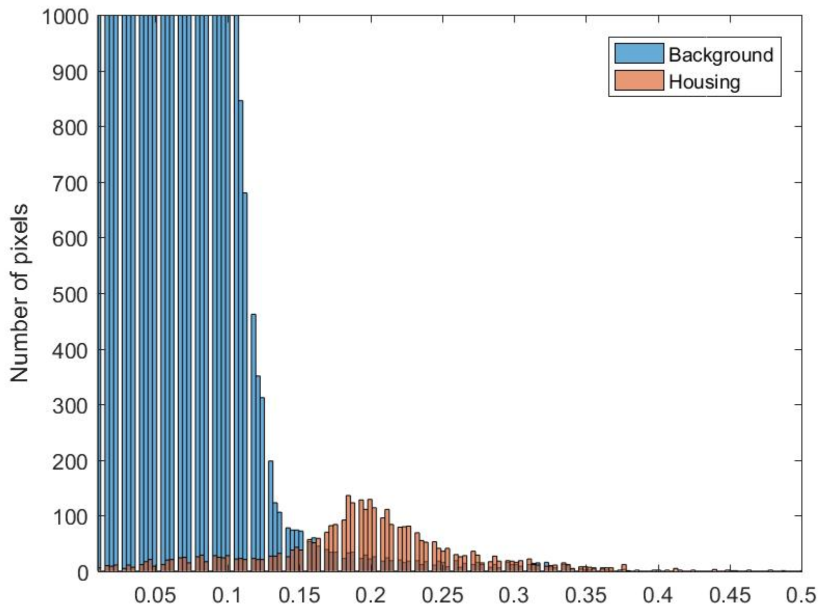


Figure 5.12: Histogram for the frame in figure 5.11

The leftmost part of the histogram is the background, so a candidate thresholding value is $t = 0.15$. The logic used for creating the resulting binary frame is:

$$T = \begin{cases} g(u,v) \geq t \rightarrow T(u,v) = 1 \\ g(u,v) < t \rightarrow T(u,v) = 0 \end{cases} \tag{5.2.4}$$

Doing this for the entire frame yields the binary frame shown on figure 5.13

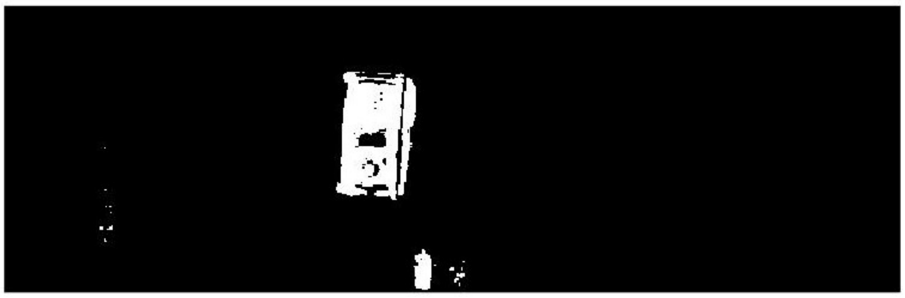


Figure 5.13: The resulting binary frame

A blob is defined as a connected white area, but the problem here is, the blob generated by the housing is deformed and broken. Following the same approach for different frames, shows that the blob is always deformed and broken, so it is necessary to use a different method for blob detection.

HSI colorspace

In the book (Moeslund, 2012) it is proposed to use the HSI color space for video, since it is feasible to assume that the housing color is unique throughout the video sequence. The acronym HSI stands for; Hue, Saturation and intensity. Hue is the pure color, Saturation is the amount of white light mixed with the Hue color. Lastly the Intensity is calculated as. (Moeslund, 2012)

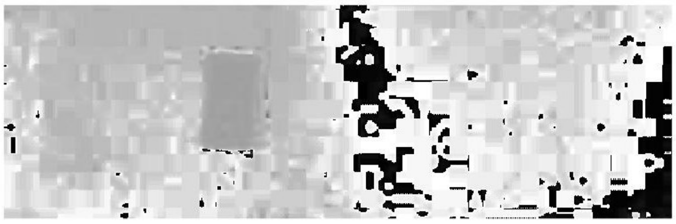
$$I = \frac{R+G+B}{3}, \quad I \in [0, 255] \tag{5.2.5}$$

The two remaining are calculated from the RGB color space as such: (Moeslund, 2012)

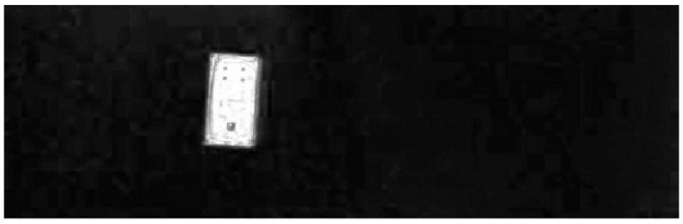
$$H = \begin{cases} cos^{-1}\left(\frac{1}{2}\frac{(R-G)+(R-B)}{\sqrt{(R-G)(R-G)+(R-B)(G-B)}}\right), & G \geq B \\ 360^o - cos^{-1}\left(\frac{1}{2}\frac{(R-G)+(R-B)}{\sqrt{(R-G)(R-G)+(R-B)(G-B)}}\right), & G < B \end{cases}, \quad H \in [0, 360[\tag{5.2.6}$$

$$S = 1 - 3\,\frac{\min\{R, G, B\}}{R+G+B}, \quad S \in [0, 1] \tag{5.2.7}$$

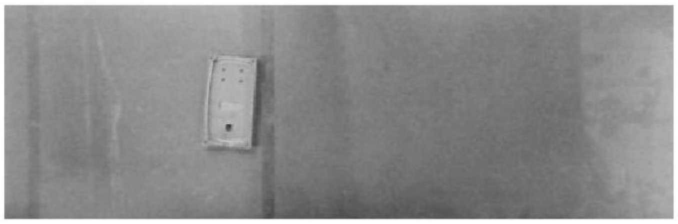
Doing this for the frame in figure 5.9 yields the three new color channels for the frame, which are shown on figure 5.14.



(a) The Hue channel for frame, the frame normalised to better see the details



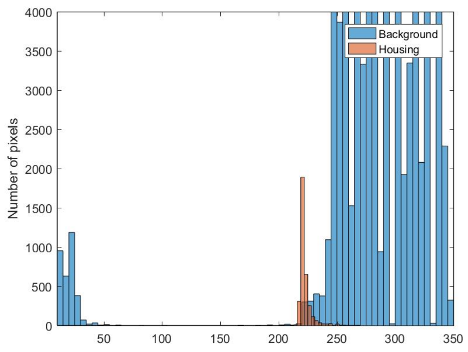
(b) The Saturation channel for frame



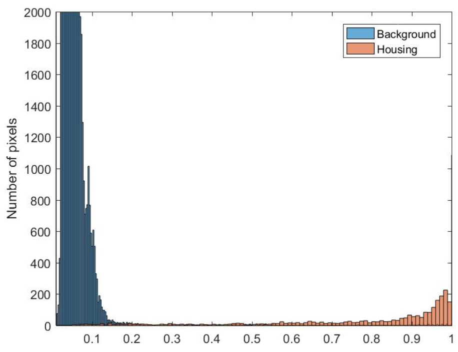
(c) The Intensity channel for frame

Figure 5.14: The HSI color space channels

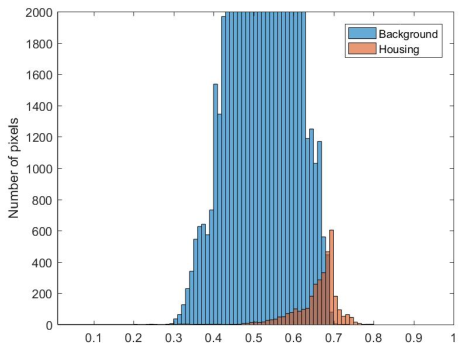
The Saturation channel shows only the housing and the line that caused noise before is also gone. This indicates that the HSI color space is better suited for this reports application. It is also possible to do background subtraction on the HSI frame, but to begin with, the thresholding is done on the Saturation and Hue channel. The Hue channel is used as a precaution in case there happens to be noise on the Saturation channel on other frames. The histogram for the three channels are shown on figure 5.15.



(a) Histogram for the Hue channel



(b) Histogram for the Saturation channel



(c) Histogram for the Intensity channel

Figure 5.15

By probing the pixel values on figure 5.14 it is possible to determine the range for the Hue and Saturation values, which are as follows:

$$H \in [180, 230] \tag{5.2.8}$$
$$S \in [0.4, 1] \tag{5.2.9}$$

The programming used for thresholding is:

```
    g = (HSI(:,:,1)>=180 & HSI(:,:,1) <=230 &
         HSI(:,:,2) >= 0.4 & HSI(:,:,2) <= 1);
```

The code is from MATLAB, where the syntax $(:, :, j)$ indicates all pixels in the u- and v-direction and j indicates which channel. As before this thresholding yields a binary image, which is shown on figure 5.16.

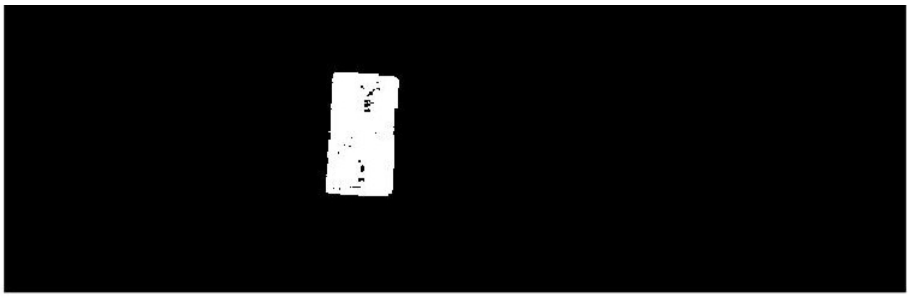


Figure 5.16: The resulting binary frame after thresholding the HSI frame

There is some small holes in the blob but all the background is removed and the blob is deemed a adequate representation of the housing. Lastly there is the case of when the robot is within the region of interest and it is necessary to ensure that the robot is not causing the vision system to fail. The frame on figure 5.17 is used for this test, note the resolution is significantly lower for this frame.

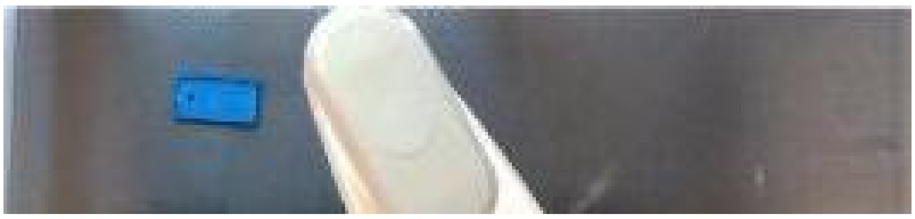


Figure 5.17: Frame from the camera, where the robot is within the region of interest

Using the described thresholding the binary image for this frame is derived and shown on figure 5.18

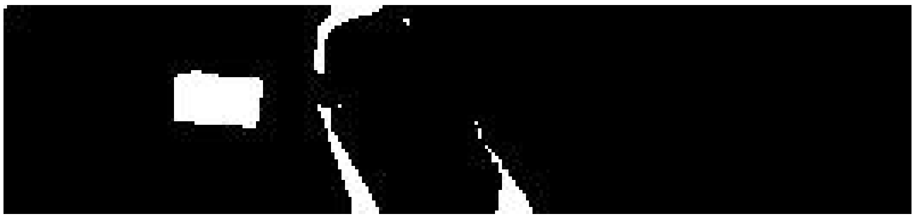


Figure 5.18: Frame from the camera, where the robot is within the region of interest

The robot is causing false blobs to appear on the binary image, which is not acceptable and they therefore has to be remove. Since the thresholding are already using the Hue and Saturation channel, an idea is to see the histogram for Intensity channel, this is shown on figure 5.19.

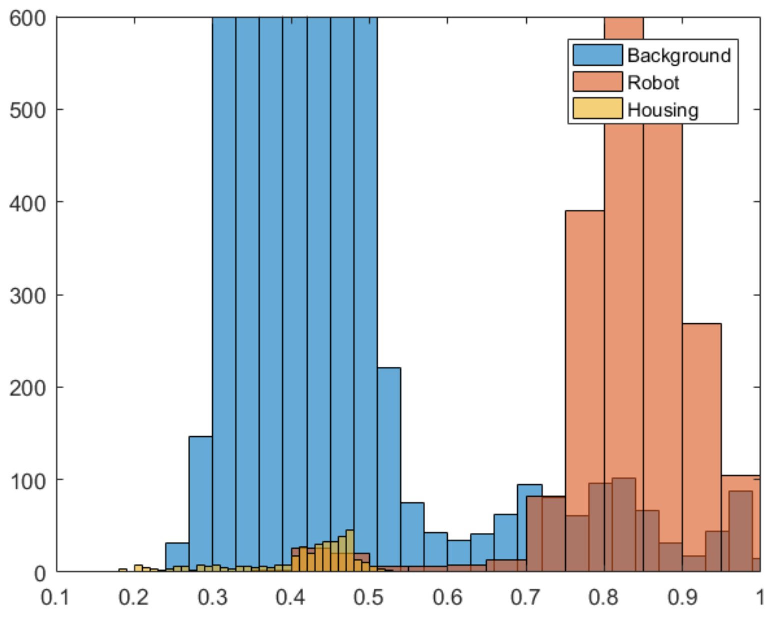


Figure 5.19: Histogram for the Intensity channel

Even though the housing and background is still coincident, the robot is clearly separated from the rest, where the pixels from the robot that is coincident with the is few pixels of background that was not removed in the plotting process. By adding an extra statement to the thresholding, the vision system is now also able to remove the robot.

```
g = (HSI(:,:,1)>=180 & HSI(:,:,1) <=230 &
HSI(:,:,2) >= 0.4 & HSI(:,:,2) <= 1 & HSI(:,:,3) < 0.6);
```

5.2.2 Determination of centroid and orientation

The purpose is to determine position and orientation of the housing such that the robot can place a lid correctly on it. Everything presented in this section is done automatic in MATLAB with the usage of a single command, but it is still necessary to present all the relevant theory behind the command: (Corke, 2013)

```
blob = iblobs(g,'area',[min max])
```

Firstly the vision system, needs to determine the centroid of the blob found in the previous section and afterwards it is possible to find the orientation. The first step is to determine the size of the blob and number of blobs in the frame, this is done by comparing a pixel to its' neighbours. If the neighbours are white pixels they are connected and if they are black, they are not. Thereby it is possible to set up a book keeping matrix of the same size as the frame which keeps track of which pixels are background and which pixels belongs to a given blob. The MATLAB command uses the binary frame and only returns the blobs which areas are within the limits defined by the option 'area' and the limits [**min max**], the area is defined as the number of pixels within the blob. Once the blob is known it is necessary to determine the bounding box, this is the smallest possible box which still fully contain the blob. Firstly it is necessary to determine the maximum and minimum non-zero pixel in both the u- and v-direction within the blob, since these will form the corners of the box. The result of this is shown on figure 5.20, where the box derived from the binary frame is plotted on top of the frame from the camera. (Corke, 2013)

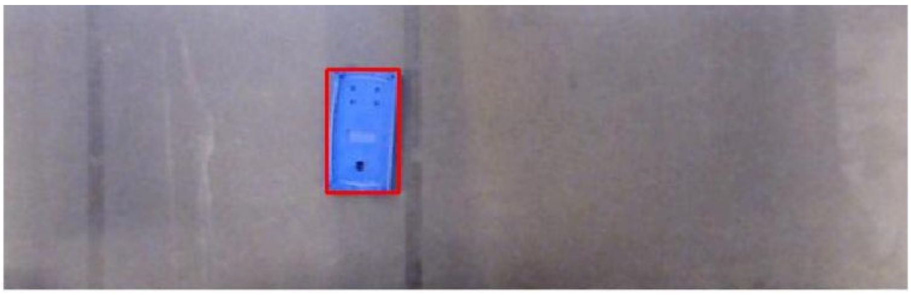


Figure 5.20: The frame from the camera where the bounding box is laid on top

As expected the box fully contains the housing, which verifies the quality of the binary frame. The next step is to calculate the center of the blob, this is done using the image moment. If all the pixels within the housing blob and only these are moved to a new frame $H(u, v)$, which is the same size as the binary frame, then the moment for H is defined as: (Corke, 2013)

$$m_{qp} = \sum_{(u,v)\in H} u^p \, v^q \, H(u,v) \tag{5.2.10}$$

The sum $(p + q)$ is the order of the moment, where the zero order moment is defined as

$$m_{00} = \sum_{(u,v)\in H} H(u,i) \tag{5.2.11}$$

A physical interpretation of the moments, is mass distribution where the pixels represent units of area and mass. With this in mind it is possible to calculate the centre of mass or more commonly within vision, the centroid of the region as:

$$u_c = \frac{m_{10}}{m_0} \tag{5.2.12}$$

$$v_c = \frac{m_{01}}{m_{00}} \tag{5.2.13}$$

Adding this centroid to the previous plot yields the result shown on figure 5.21.

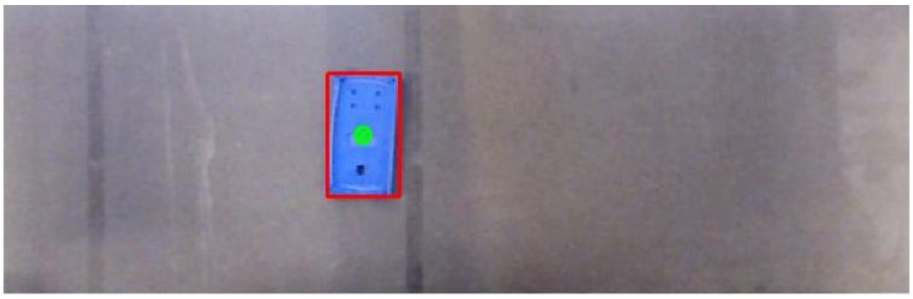


Figure 5.21: The frame from the camera with the bounding box and the centroid plotted

Lastly to calculate the orientation of blob, this is done by introducing the central moments μ_{pq} defined as:

$$\mu_{pq} = \sum_{(u,v)\in H} (u - u_c)^p (v - v_c)^q H(u,v) \tag{5.2.14}$$

The central moments are invariant with respect to position and relate to the moments in the following way:

$$\mu_{10} = 0 \tag{5.2.15}$$

$$\mu_{01} = 0 \tag{5.2.16}$$

$$\mu_{20} = m_{20} - \frac{m_{10}^2}{m_{00}} \tag{5.2.17}$$

$$\mu_{02} = m_{02} - \frac{m_{01}^2}{m_{00}} \tag{5.2.18}$$

$$\mu_{11} = m_{11} - \frac{m_{01}\, m_{01}}{m_{00}} \tag{5.2.19}$$

Giving these central moments a physical interpretation, they describe the inertia of the blob and can be inserted in a inertia matrix.

$$J = \begin{bmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{bmatrix} \tag{5.2.20}$$

The orientation is calculated by finding a equivalent ellipse which has the same inertia matrix as H, this is done by using the eigenvectors of the inertia matrix. The eigenvectors gives the ellipses' principal axes. The major axis, v_y, correspond to the eigenvector associated to the largest eigenvalue of the inertia matrix and likewise the minor axis, v_x, is given by the eigenvector associated with the smallest eigenvalue. Thereby the orientation with respect to the horizontal axis is given by:

$$\theta = \tan^{-1}\left(\frac{v_y}{v_x}\right) \tag{5.2.21}$$

The equivalent ellipse is added to the frame from the camera on figure 5.22.

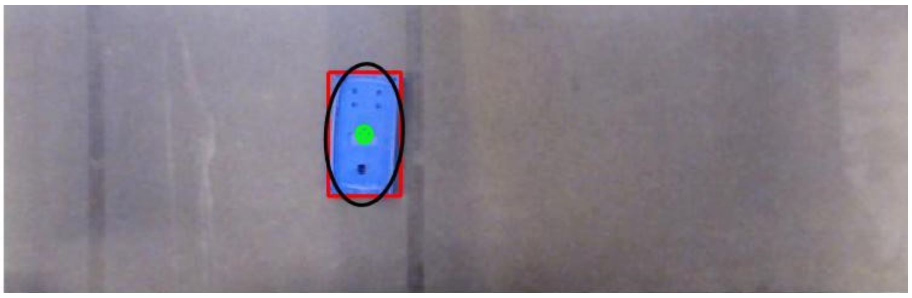


Figure 5.22: The frame from the camera with the bounding box, centroid and equivalent ellipse plotted

A main limitation to keep in mind, is that the orientation is not to a specific point on the housing, since it is simplified to a blob, this means that say the housing is rotated 180^o on figure 5.22, the orientation will remain the same. This means that since the lid needs to be correctly aligned it might cause problems later one, that the derived orientation is not to a specific part of the housing.

5.2.3 Tracking

With the method for finding the centroid and orientation of the housing in a single frame done, it is necessary to test if this also works in a video. The reason for this is, that the background conditions are highly dynamic due to changes in lighting. These changes comes both from the artificial lighting in the work shop where the test setup is located, and from the windows in the ceiling. Therefore several videos were made under different weather conditions and on different times of the day, to verify the robustness of the vision system. The test performed, is that the conveyor belt is set to a constant velocity and the housing is placed on the belt in a arbitrary orientation. The plots shown on figure 5.23 are the calculated centroid throughout the video.

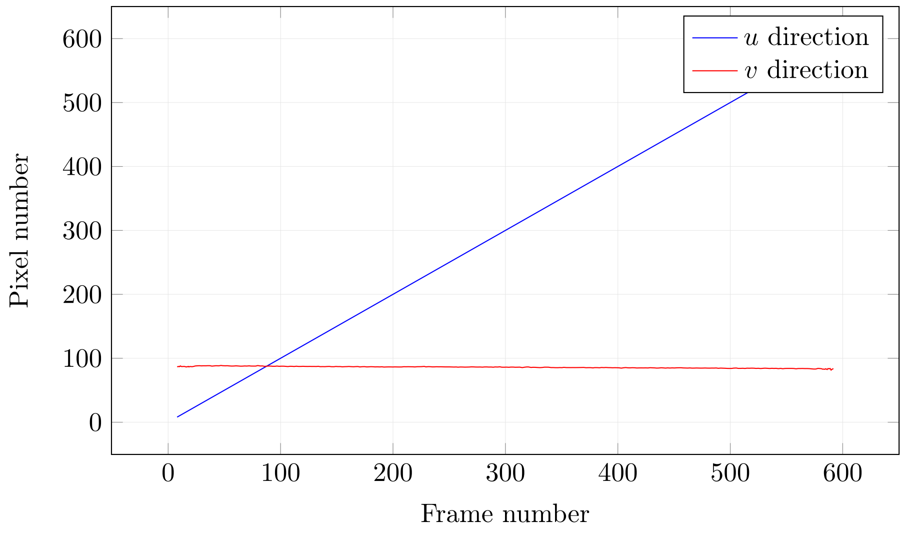


Figure 5.23: The position of the centroid in the (u, v) space throughout the video

As expected the centroid is moving along the horizontal direction, u. The small change in the vertical direction, v, is due to the camera's orientation is not perfectly aligned with the conveyor belt. The plot shown on figure 5.24 is the detected orientation of the housing.

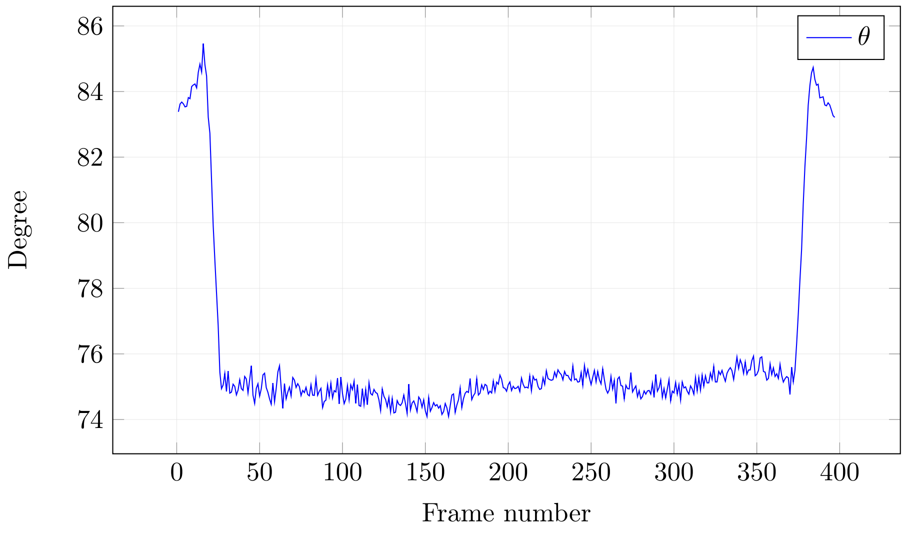


Figure 5.24: The orientation of the housing throughout the video

There is more noise on the orientation than position, especially in the beginning and ending, where the vision system is only able to detect part of the housing. It therefore stands to reason, that maybe the vision system should only calculate the position and orientation once the housing is fully inside the camera's view. Furthermore, the orientation of the housing does not change during the video, so it is necessary to filter the signal to remove the noise on the orientation. The reason the orientation is more noisy than the centroid, is that while the centroid is only dependent on the size of the blob, the orientation is dependent of number of pixels in the blob. When the vision system is operating continuous, there are holes opening and closing within the blob, which means the moment is changing. One approach to improve the binary image and remove some of these holes, is to use the dilation-erosion method (Moeslund, 2012). This is a two step approach where firstly holes are removed by dilating the blob. This is done by focusing on a pixel and checking the neighbouring pixels within some radius. A pseudo MATLAB code is:

```
for i = 2:lg-1
        for j = 2:hg-1
                A=g(i-1:i+1,j-1:j+1);
                tmp = sum(A(:))
                if tmp >=1
                        g_d(i-1:i+1,j-1:j+1)=1;
                else
                        g_d(i,j)=0;
                end
        end
end
```

The code checks if all pixels in a radius of 3 is equal to 1, and if this is true, then the pixel in focus is set to 1. The problem with dilation is it also moves the edges of the blob, which is not desirable, it is therefore necessary to try and move the edges back, such that the original size of the blob is kept. This is done with the erosion operation, which works in the same as dilation, except here if one of the neighbouring pixels are zero all the pixels within the radius is set to zero. Thereby the edges are reduced back to the original position. The result on the orientation is shown on figure 5.25

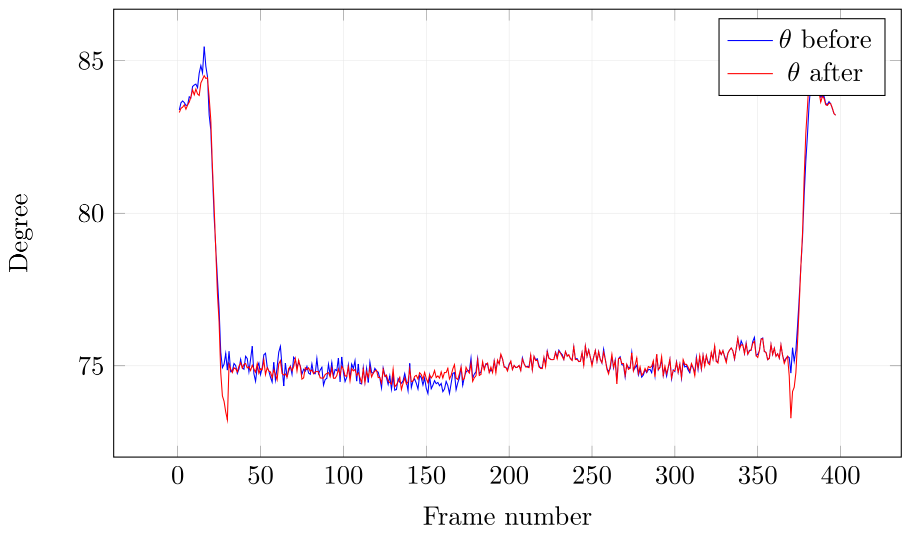


Figure 5.25: The orientation of the housing throughout the video

The dilation-erosion makes the orientation more affected by the housing entering the field of view, but it also removes some of the osculation, though the orientation is still not constant. The benefit of this method is, it is computational cheap to use the dilation algorithm several times and then the erosion algorithm the same number of times to ensure all the holes in the blob is remove.
The last state determine is the velocity of the housing, this is done using the backward Eulor

method

$$\dot{u}_k = \frac{u_k - u_{k-1}}{\Delta t} = (u_k - u_{k-1})\, f_c \tag{5.2.22}$$

$$\dot{v}_k = \frac{v_k - v_{k-1}}{\Delta t} = (v_k - v_{k-1})\, f_c \tag{5.2.23}$$

Numeric differentiation as this, always introduces increased noise, which is also evident on the graphs on figure 5.26, where the velocities for the entire video is shown.

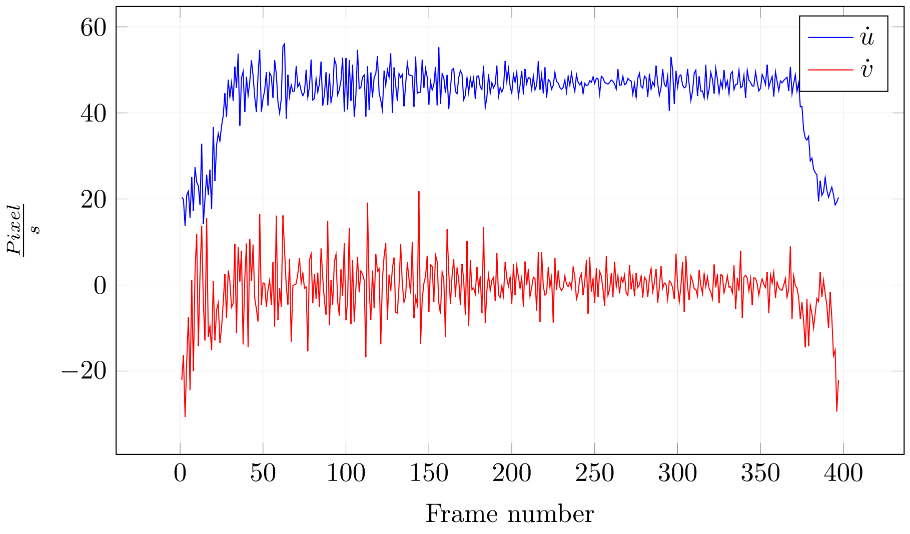


Figure 5.26: The calculated velocities in the horizontal- and vertical direction

As before, the velocity calculation is affected when the housing enters and leaves the camera's field of view. Even though the velocities are noise, it is still possible to say, that the velocity in the u-direction is oscillating around $43\frac{pixel}{s}$ and around $0\frac{pixel}{s}$ in the v-direction. This is also to be expected since, as mentioned, the conveyor is set to a constant velocity.

Chapter 6

Modeling

The idea is to setup a prediction for the future position of the housing from the detected position and orientation of the housing, as shown on figure 6.1. To do this, it is necessary to setup a mathematical model which approximates the housing on the moving conveyor belt.

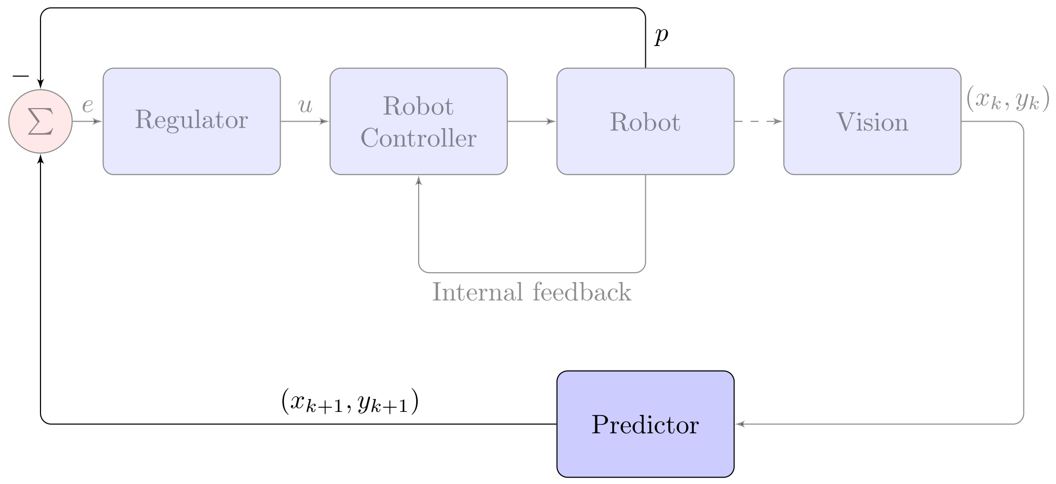


Figure 6.1: The current part of the control approach

Likewise when using a robot, it is in most cases the end-effector which is meant to do something, like the case of this report, where it has to place the lid on the housing. But there is no sensors telling where the end-effector is in the global coordinate system, only encoders returning joint values. It is therefore necessary to derive the link between joint values and end-effector location, which is done with the kinematic relations. The purpose of this chapter is to derive this kinematics for the Cobra s600, and also determine the transformation from the global coordinate system introduced in section 5.1.2 to the location of the end-effector. Likewise the chapters derives a model for the housing on the conveyor belt, which forms the basis for the deviation of a predictor.

6.1 Robot model

Robots are described in two different spaces, the Joint space and the Cartesian space. Additionally for visual servoing system, there is also the image plane, so it is necessary to derive the transformation from this space to one of the robots two spaces. The purpose of this section is to derive the direct kinematics, linking the Joint space to the Cartesian and deriving the inverse kinematics which links the Cartesian space to the Joint. Lastly a method for transforming from the image plane to the robot's base frame is derived.

6.1.1 Kinematics

Even though the robot controller is able to pass both the Cartesian coordinates of the end-effector and joint values, it is still necessary to formulate the transformation matrix from the base to the end-effector. Following the standard David-Hartenberg convention for assignment of local coordinate systems, it is possible to derive the David-Hartenberg parameters. The local coordinate systems are shown on figure 6.2.

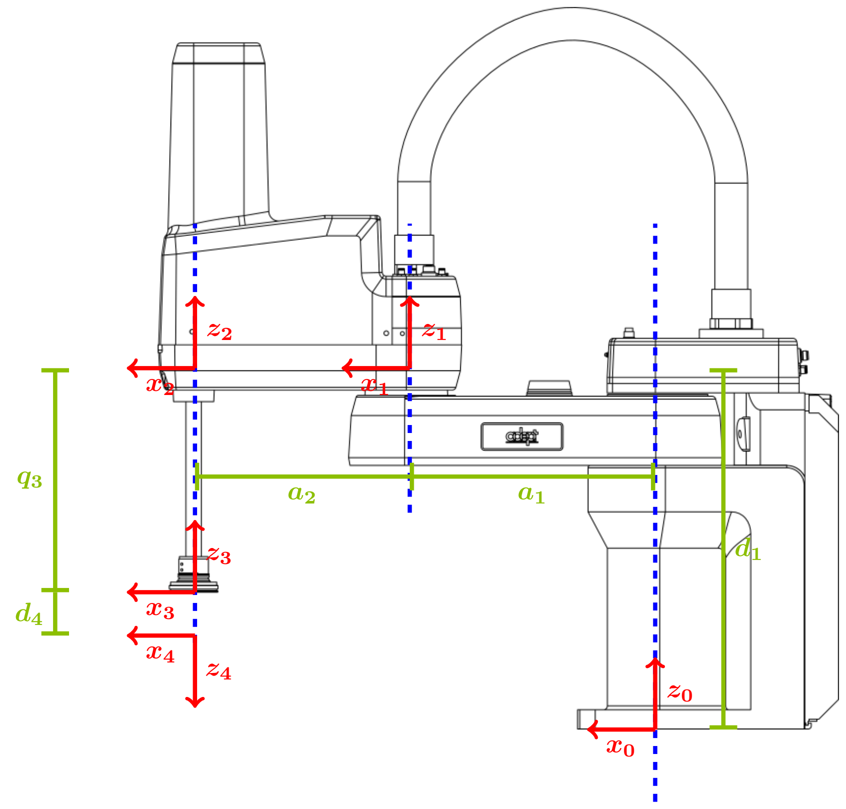


Figure 6.2: The Cobra s600 with local coordinate systems.

From the figure it is possible to setup the DH-table, which is shown in table 6.1.

i	α_i [-]	a_i [mm]	d_i [mm]	θ_i [-]
1	0	$a_1 = 325$	$d_1 = 387$	q_1
2	0	$a_2 = 275$	0	q_2
3	0	0	q_3	0
4	α_4	a_4	d_4	q_4

Table 6.1: The David-Hartenberg parameters for the Cobra s600. The measurements are from the data sheet found in appendix B

The last local coordinate system, number four, is made generic because the end-effector design is not decided yet. The general DH-matrix is as follows when using compact notation:

$$^{i-1}\mathbf{A}_i(q) = \begin{bmatrix} c\theta_i & -c\alpha_i\, s\theta_i & s\alpha_i\, s\theta_i & a_i\, c\theta_i \\ s\theta_i & c\alpha_i\, c\theta_i & -s\alpha_i\, c\theta_i & a_i\, s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.1.1}$$

To derive the transformation matrix from the base to the end-effector, it is necessary to multiply the four transformation matrices. This is done in appendix A. The final transformation matrix written on super compact form is:

$$^{0}\mathbf{A}_4(q) = \begin{bmatrix} c_{124} & -c_{\alpha_4}\, s_{124} & s_{\alpha_4}\, s_{124} & a_1\, c_1 + a_2\, c_{12} + a_4\, c_{124} \\ s_{124} & c_{\alpha_4}\, c_{124} & -s_{\alpha_4}\, c_{124} & a_1\, s_1 + a_2\, s_{12} + a_4\, s_{124} \\ 0 & s_{\alpha_4} & c_{\alpha_4} & d_1 + q_3 + d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.1.2}$$

$$\vec{p}(q) = \begin{bmatrix} X \\ Y \\ Z \\ \phi \end{bmatrix} = \begin{bmatrix} a_1\, c_1 + a_2\, c_{12} + a_4\, c_{124} \\ a_1\, s_1 + a_2\, s_{12} + a_4\, s_{124} \\ d_1 + q_3 + d_4 \\ q_1 + q_2 + q_4 \end{bmatrix} \tag{6.1.3}$$

Where $\vec{p}(q)$ describes the end-effector location, and the last row is the orientation of the end-effector. The kinematics is verified with Adept ACE, which is able to compute the precise location of the end-effector. This means that by computing the kinematics for some arbitrary joint values and placing the robot in the same location, the kinematics is correct if the same end-effector location is achieved. The result of this is listed in table 6.2

q_1 [deg]	q_2 [deg]	q_3 [mm]	q_4 [deg]
10	10	50	0
	X [mm]	Y [mm]	Z [mm]
Kinematics	595	-56.4	337
Robot	595	-56.4	337

Table 6.2: The verification of the kinematics, the orientation components of the verification is omitted, since the kinematic only computes the position vector

Which verifies that the DH-assignment of local coordinate systems matches that of ACE.

6.1.2 Transformation from image plane to robot's base frame

To connect the robot to the vision system, it is necessary to derive a method for transforming a point on the image plane to the robot's base frame. The initial method is to use the global coordinate system introduced in section 5.1.2, where figure 6.3 shows the kinematic transformation concept.

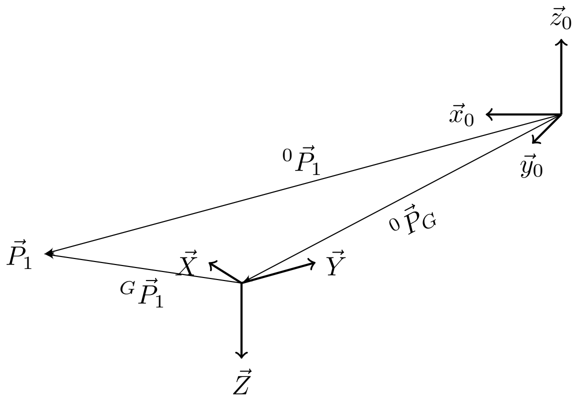
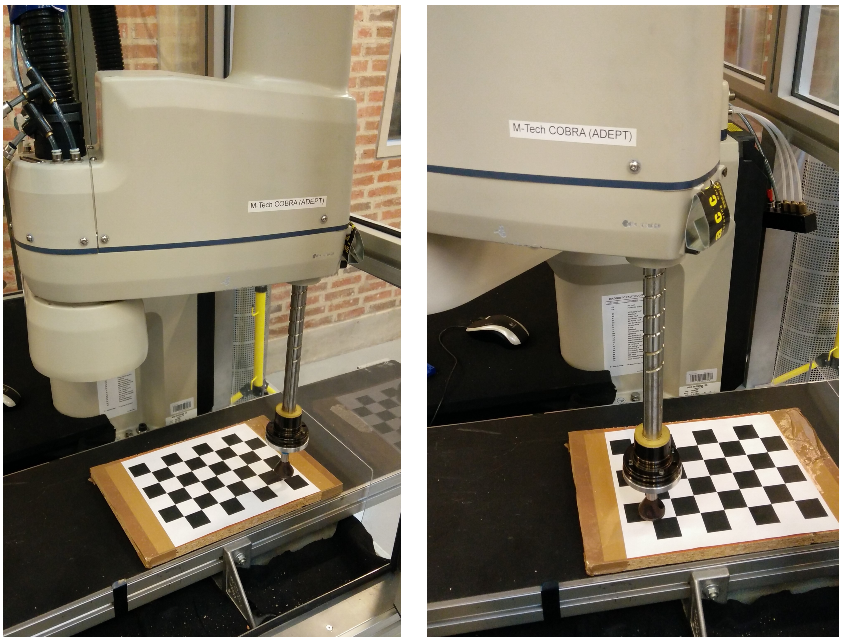


Figure 6.3: The connection between the global coordinate system and the robot's base frame

The goal is to determine the vector ${}^0P_1$ and this is done with the following expression:

$$ {}^0\vec{P}_1 = {}^0\vec{P}_G + \mathbf{R}\ {}^G\vec{P}_1 \tag{6.1.4} $$

Where R is the unknown rotation matrix, and ${}^GP_1$ is given by the transformation from the image plane to the global coordinate system. The vector between the two coordinate systems, ${}^0P_G$ is derived, by moving the robot's end-effector to the origo of the global coordinate system, this is shown on figure 6.4a.



(a) The robot where the end-effector position is coincident with the global origo

(b) The robot where the end-effector position is moved five squares in the X-direction

Figure 6.4

Assuming the two vertical axis Z and z_0 are parallel, the unknown rotation matrix R is written on the form:

$$\mathbf{R} = \begin{bmatrix} \cos(\Theta) & \sin(\Theta) & 0 \\ \sin(\Theta) & -\cos(\Theta) & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{6.1.5}$$

The matrix is determined, by moving the end-effector to a point known both in the global coordinate system and in the robot's base frame. This point is shown on figure 6.4b where the end-effector is coincident with a point five squares of 29 [mm] from the global origo in the X-direction. The equation becomes:

$$\begin{bmatrix} P_{1,x} \\ P_{1,y} \\ P_{1,z} \end{bmatrix} = \begin{bmatrix} P_{0,x} \\ P_{0,y} \\ P_{0,z} \end{bmatrix} + \begin{bmatrix} \cos(\Theta) & \sin(\Theta) & 0 \\ \sin(\Theta) & -\cos(\Theta) & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} P_{g,x} \\ 0 \\ 0 \end{bmatrix} \tag{6.1.6}$$

Solving for cos(Θ) and sin(Θ) the resulting expressions becomes:

$$\cos(\Theta) = \frac{P_{1,x} - P_{0,x}}{P_{gx}} \tag{6.1.7}$$

$$\sin(\Theta) = \frac{P_{1,y} - P_{0,y}}{P_{gx}} \tag{6.1.8}$$

This approach is tested by placing the housing on the conveyor belt and then the vision system is used to detect the centroid. This centroid is then transformed to global coordinate system and from there to the robot's base frame. A problem arises when the housing is placed further away from the global origo than the point used to derive R. As the distance increases an error between the actual centroid of the housing and resulting end-effector position is introduced. This is not acceptable since this errors means the robot is not able to track correctly the housing on the conveyor, it is therefore decided to try an different approach.
Taking advantage of the placement of the camera, showed on figure 5.4 in section 5.1, it is possible to say that there is constant scaling. This means that the image plane and conveyor plane are parallel, and the robot's base plane is spanned by the x_0- and y_0-axis, and therefore a point in the base plane can be expressed as shown on figure 6.5.

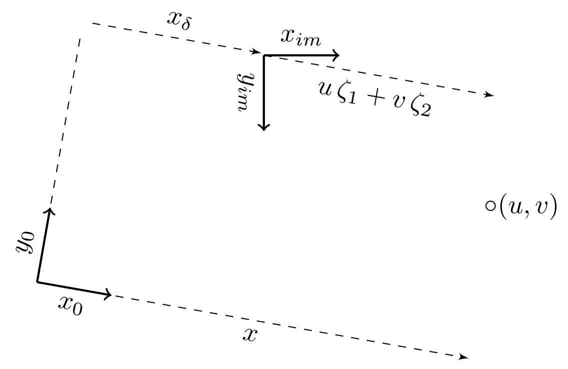


Figure 6.5: Correlation between an point on the image plane and robot plane if the scaling is constant

The same is true for the y coordinate, so the equations becomes

$$x = x_\delta + u\,\zeta_1 + v\,\zeta_2 \tag{6.1.9}$$

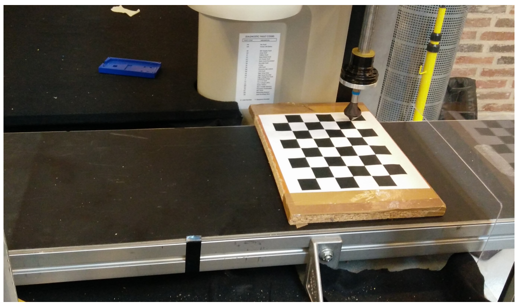
$$y = y_\delta + u\,\eta_1 + v\,\eta_2 \tag{6.1.10}$$

Which is rewritten to the form:

$$\vec{x} = \begin{bmatrix}1 & u & v\end{bmatrix} \vec{\zeta} \tag{6.1.11}$$

$$\vec{y} = \begin{bmatrix}1 & u & v\end{bmatrix} \vec{\eta} \tag{6.1.12}$$

The transformation vectors $\vec{\zeta}$ and $\vec{\eta}$ are constants and are estimated by using multiple points which is known both in the image- and robot plane. These points should be spread around the region of interest but still within the robot's workspace. Figure 6.6a shows how the chessboard is used to find these points by using MATLAB to determine the origo of the chessboard and moving the robot so the end-effector is coincident with this origo. Figure 6.6b shows the five points found


(a) The chessboard is placed within the region of interest and at the edge of the robot's available workspace

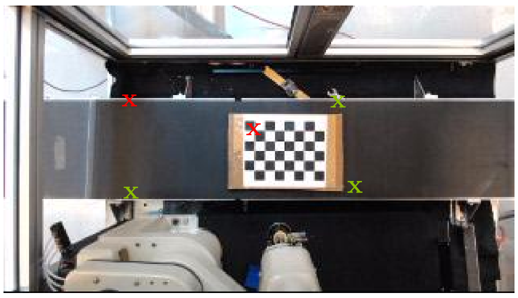

(b) The five points used to determine the two transformation constants

Figure 6.6

from this method. They are all placed within the region of interest but the three green points are placed based on the edge of the robot's available workspace. Using these five points, it is possible to set up the following systems of equations

$$\begin{bmatrix}x_1\\x_2\\\vdots\\x_5\end{bmatrix} = \begin{bmatrix}1 & u_1 & v_1\\1 & u_2 & v_2\\\vdots & \vdots & \vdots\\1 & u_5 & v_5\end{bmatrix} \vec{\zeta} \tag{6.1.13}$$

$$\begin{bmatrix}y_1\\y_2\\\vdots\\y_5\end{bmatrix} = \begin{bmatrix}1 & u_1 & v_1\\1 & u_2 & v_2\\\vdots & \vdots & \vdots\\1 & u_5 & v_5\end{bmatrix} \vec{\eta} \tag{6.1.14}$$

Because this system is overdetermined, it is solved using the least square optimization method, which is done with MATLAB. To validate the accuracy of this method, the percentage deviation is calculated as :

$$e_{x,i} = \sum_{i=1}^{5} \frac{x_i - \tilde{x}_i}{x_i} \cdot 100 = \begin{bmatrix}0.15 & 0.24 & -0.08 & -0.35 & -0.62\end{bmatrix}^T \tag{6.1.15}$$

$$e_{y,i} = \sum_{i=1}^{5} \frac{y_i - \tilde{y}_i}{y_i} \cdot 100 = \begin{bmatrix}-0.11 & 0.11 & -0.24 & 0.23 & 0.01\end{bmatrix}^T \tag{6.1.16}$$

Where $\tilde{x}$ and $\tilde{y}$ are the approximated positions in the robot's base plane using the image coordinates and the two transformations vectors.

6.1.3 Inverse Kinematics

Even though the robot controller supports moving to a point in the Cartesian space, it is still desirable to derive the inverse kinematics for the robot, since this is used to check if the housing is within the robot's available workspace. For a SCARA type robot, there exist algebraic expression for the conversion from the Cartesian space to the joint space, these are derived from the direct kinematics in annex B and the resulting expressions are:

$$q_1 = \operatorname{atan2}\left(\frac{-K_1\,K_4 - K_2\,K_3}{K_2\,K_4 - K_1\,K_3}\right) \tag{6.1.17}$$

$$q_2 = \pm\cos^{-1}\left(\frac{X^2 + Y^2 - a_1^2 - a_2^2}{2\,a_1\,a_2}\right), \quad |\cos(q_2)| \le 1 \tag{6.1.18}$$

$$q_3 = Z - d_1 - d_4 \tag{6.1.19}$$

$$q_4 = \phi - q_1 - q_2 \tag{6.1.20}$$

It is necessary to define whether to use the positive or negative q_2 angle, which defines if the robot should be respectively in left- or right arm configuration. Likewise $|\cos(q_2)|$ is used to check if the point in the Cartesian space is within the workspace, also if $|\cos(q_2)| = 1$ then $q_2 = 0$ which means the robot is in singular configuration, which is undesirable and should therefore be avoided. The constants in the expression for q_1 are defined as such:

$$K_1 = a_2\,c_2 + a_1 \tag{6.1.21}$$

$$K_2 = -a_2\,s_2 \tag{6.1.22}$$

$$K_3 = -X \tag{6.1.23}$$

$$K_4 = -Y \tag{6.1.24}$$

This concludes the deviation of the inverse kinematics for the robot.

6.1.4 Dynamics

Within Adept ACE there is also a emulation mode, which means that Adept has developed a dynamic model of the robot, where the same control strategies are implemented, such that the emulated robot has the same response as the actual robot. It is decided to use this emulation mode instead of deriving the dynamic equations for the robot, for while it is possible to derive these equations, without information about the control strategies within the robot controller, the dynamic model would not approximate the actual robot's response. Figure 6.7 and 6.8 shows a step response from the emulated robot and from the actual robot.

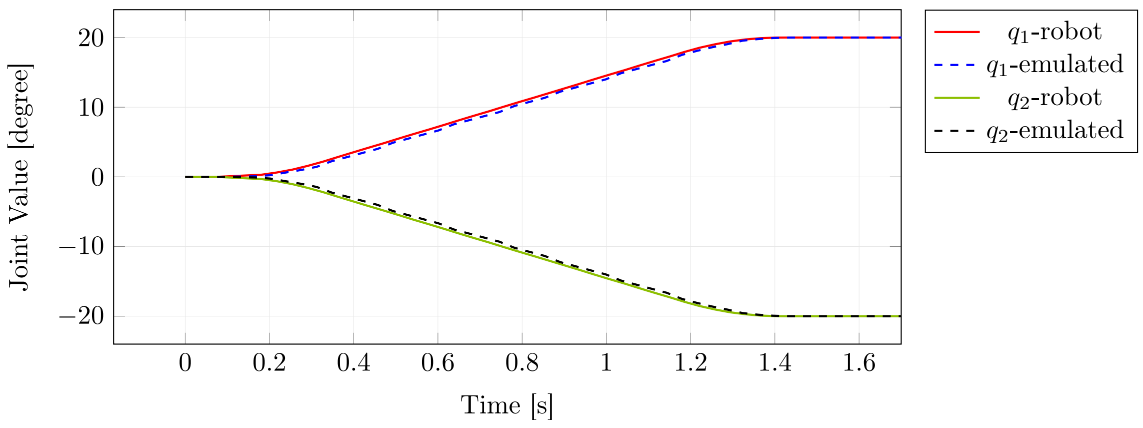


Figure 6.7: The step response for the joints q_1 and q_2 from both the actual robot and the emulated. The speed is set to 150 [mm/s] and acceleration is 100% of maximum acceleration

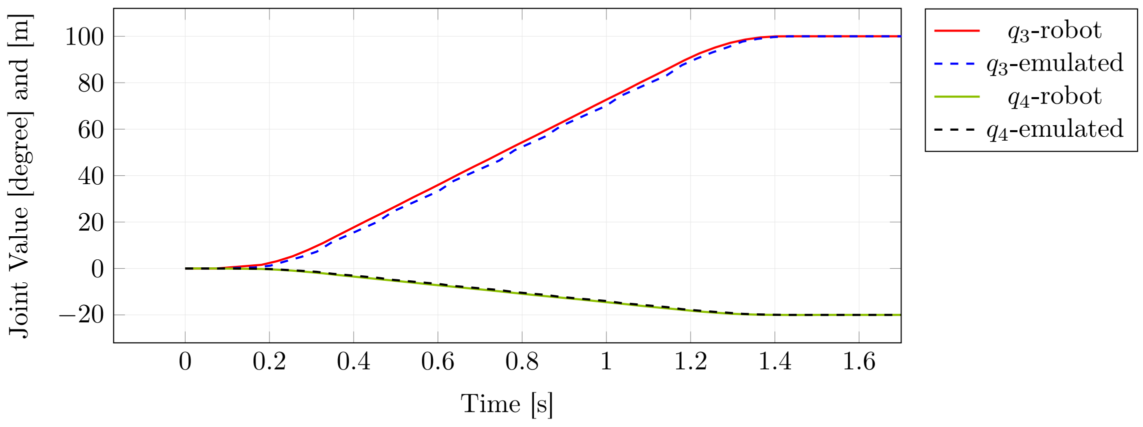


Figure 6.8: The step response for the joints q_3 and q_4 from both the actual robot and the emulated. The speed is set to 150 [mm/s] and acceleration is 100% of maximum acceleration

The emulated robot is slightly more damped than the actual robot, since its' rise and settle time are larger than the actual robots. This means that if a control strategy is functional on the emulated robot, it might have to be made less aggressive on the actual robot. Overall the emulated robot's response is deemed adequate as a representation of the actual robot's response. Likewise this test verifies that the robot is following the bang-coast-bang technique presented in section 4.1.

6.2 Conveyor belt dynamics

Because the velocity of the conveyor belt is unknown, it is necessary to develop a model of the housing moving, which describes the planer motion the conveyor belt causes. The purpose of this section is to derive this model for the housing and then use this model to set up the predictor, which is shown on figure 6.1.

6.2.1 Housing model

One of the requirements of this report, is that all information about the conveyor belt is unknown. It is therefore necessary to make assumption about the dynamics of the conveyor belt, which is that the acceleration is constant, but unknown, for a finite time. This enables the possibility to derive a simple model for position and velocity:

$$\ddot{x}(t) = a \tag{6.2.1}$$

$$\dot{x}(t) = \dot{x}(t_0) + a\,(t - t_0) \tag{6.2.2}$$

$$x(t) = x(t_0) + \dot{x}(t_0)(t - t_0) + \frac{a}{2}(t - t_0)^2 \tag{6.2.3}$$

Since the vision system operates in the discrete time, this models needs to be transformed to the discrete domain where $(t - t_0) = \Delta t$ which is the sampling time.

$$\dot{x}_k = \dot{x}_{k-1} + a \tag{6.2.4}$$

$$x_k = x_{k-1} + \dot{x}_{k-1}\,\Delta t + \frac{a}{2}\,\Delta t^2 \tag{6.2.5}$$

As described in chapter 4 it is necessary to predict the future position. This is done by redefining the model, also the model is written in the final state space form. (Corke, 1996)

$$\underbrace{\begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \end{bmatrix}}_{\vec{x}_{k+1}} = \overbrace{\begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}}^{\mathbf{F}} \underbrace{\begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix}}_{\vec{x}_k} + \begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix} a \tag{6.2.6}$$

The desired outputs of the model is position, this is defined as such:

$$y_k = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{\mathbf{H}} x_k \tag{6.2.7}$$

Where **H** is a matrix determining which state to use as output. The last term in (6.2.6) is not used further since no information about the acceleration is known.

6.2.2 Kalman Filter

For this model to work, it is necessary to have it converge on the measurement from the vision system, to achieve this, it is decided to use the Kalman filter. The reason for this is the Kalman filter is both an estimator and filter, which means it estimates all states in the model using the position output from the vision system. The Kalman filter also filters the noise generated by the vision system. The general implementation of a Kalman filter is showed on figure 6.9.

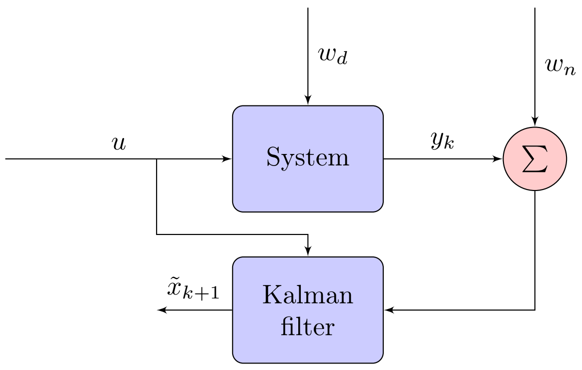


Figure 6.9: General implementation of a Kalman filter (Corke, 1996)

The disturbances are noted w_d and w_n are assumed to be Gaussian white noise with a variance of $\mathbf{V}_d$ and $\mathbf{V}_n$. The predicted state vector is noted with a tilde, $\tilde{x}_{k+1}$, and the output from the system is y_k. The Kalman filter consist of two steps, a correction step and a prediction step. These two steps are illustrated on figure 6.10. (Welch and Bishop, 2006)

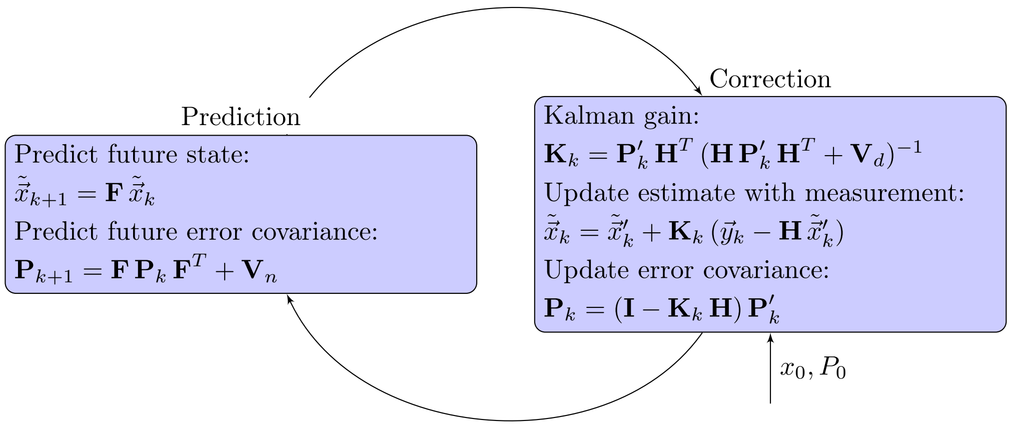

Correction
Prediction
Predict future state:
$\tilde{\vec{x}}_{k+1} = \mathbf{F}\,\tilde{\vec{x}}_k$
Predict future error covariance:
$\mathbf{P}_{k+1} = \mathbf{F}\,\mathbf{P}_k\,\mathbf{F}^T + \mathbf{V}_n$
Kalman gain:
$\mathbf{K}_k = \mathbf{P}'_k\,\mathbf{H}^T\,(\mathbf{H}\,\mathbf{P}'_k\,\mathbf{H}^T + \mathbf{V}_d)^{-1}$
Update estimate with measurement:
$\tilde{\vec{x}}_k = \tilde{\vec{x}}'_k + \mathbf{K}_k\,(\vec{y}_k - \mathbf{H}\,\tilde{\vec{x}}'_k)$
Update error covariance:
$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\,\mathbf{H})\,\mathbf{P}'_k$
x_0, P_0

Figure 6.10: Illustration of the two steps involved in a Kalman filter

The two new parameters which are introduced is the Kalman gain, $\mathbf{K}$, and the the error covariance matrix, $\mathbf{P}$. The idea with the correction box, is to check how accurate the previous prediction was with respect to the current measurement. The current estimated states, $\tilde{\vec{x}}_k$, is then corrected with the Kalman gain and error and likewise is the error covariance matrix. The prediction then uses the corrected estimated states to predict the future states. Since the model and disturbances are modelled as constant matrices, the Kalman gain and error covariance matrix reaches a steady state value within some number of iterations. The Kalman filter is initiated with an initial guess of $\tilde{\vec{x}}'_k$ and $\mathbf{P}'_k$ which is the input $\vec{x}_0$ and $\mathbf{P}_0$. The disturbance variance, $\mathbf{V}_d$, and sensor noise variance, $\mathbf{V}_n$, are diagonal matrices, which are used to define how how much the model and the sensors are trusted. This means that if for example the diagonal values of $\mathbf{V}_n$ is high, then there is not done any smoothing to the sensor signal. Typical the diagonal values in $\mathbf{V}_n$ and $\mathbf{V}_d$ is defined as the variance if each state, but in this report, the Kalman filter is tuned until a satisfactory result is achieved. The model presented in equation (6.2.6) is easily expanded such that it is applicable for the states in the vision system:

$$\begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \\ y_{k+1} \\ \dot{y}_{k+1} \\ \theta_{k+1} \\ \dot{\theta}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \\ y_k \\ \dot{y}_k \\ \theta_k \\ \dot{\theta}_k \end{bmatrix} \tag{6.2.8}$$

The poles of this discrete system are the eigenvalues of the matrix and are all 1. As a rule of thumb the poles of the prediction system should be 2 - 4 times faster than the original plant. Once the Kalman gain reaches its' steady state value, the Kalman equation becomes:

$$\tilde{\vec{x}}_{k+1} = \mathbf{F}\tilde{\vec{x}}_k + \mathbf{K}_k(\vec{y}_k - \mathbf{H}\tilde{\vec{x}}_k) \tag{6.2.9}$$

The poles of this system is defined as the eigenvalues of the closed loop matrix:

$$\vec{\lambda}_i = \mathrm{eig}(\mathbf{F} - \mathbf{K}_k\,\mathbf{H}) \tag{6.2.10}$$

The results of the Kalman filter is plotted below, where the predicted position along with the position from the vision system, which transformed into the robot's base frame with the method from section 6.1.2, is shown on figure 6.11.

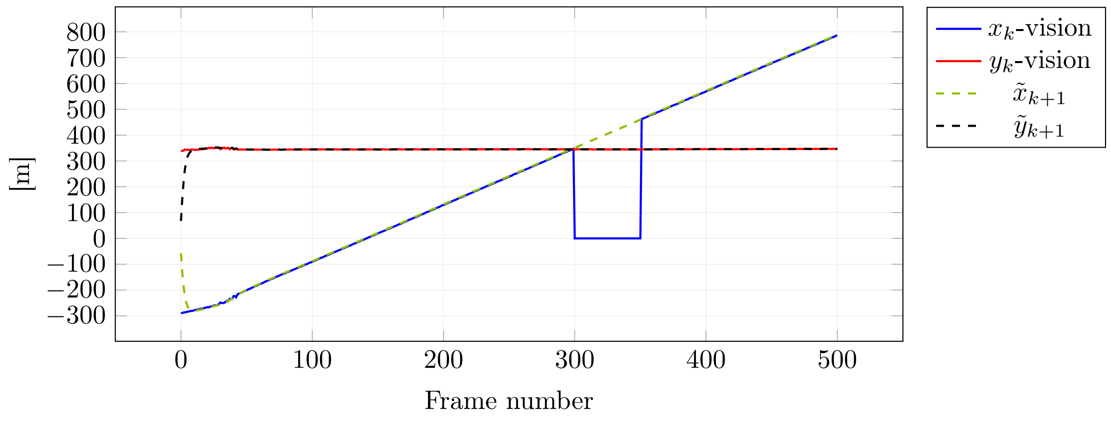


Figure 6.11: The estimated position in the horizontal and vertical direction and the same positions from the vision system

To emulate the robot blocking the view of the camera, the x_k values for frame 300 to 350 is set to zero. When this happens, the Kalman filter only uses the prediction part until the vision system again is able to detect the housing, in which case the Kalman filter resumes using the correction part. From the figure it is clear, that this method is working, since the x_{k+1} still matches x_k when it non-zero again. There is a small offset between x_k from the vision and x_{k+1}, which is due to the filter using the current values to predict the future position. By delaying the plot with one sample, it is possible to evaluate the accuracy of the prediction, this is shown on figure 6.12.

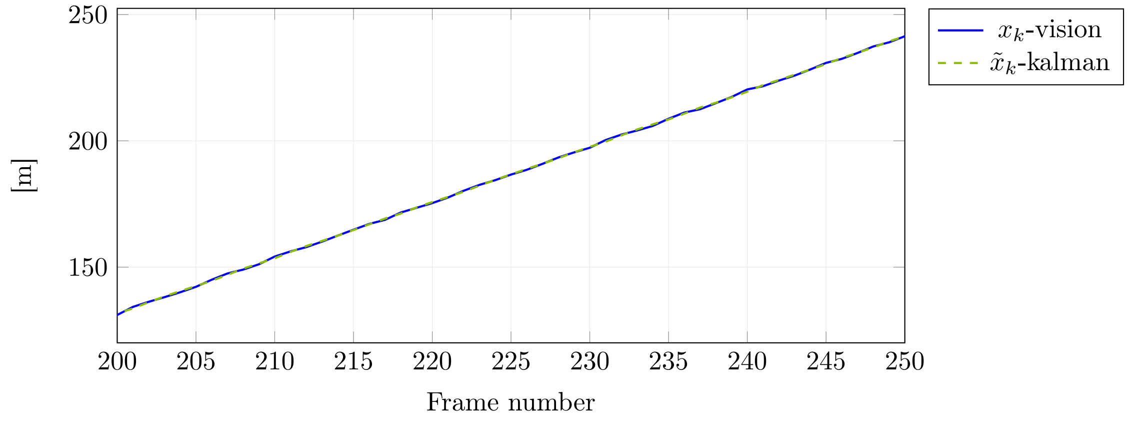


Figure 6.12: The estimated position in the horizontal direction delayed by one sample and the same position from the vision system

This shows the predictor accurately predicts the future position of the centroid. In section 5.2.3 it was shown that it is possible to derive the velocity with numeric differentiation. With the Kalman filter, the velocity is instead estimated, and the result is shown on figure 6.13.

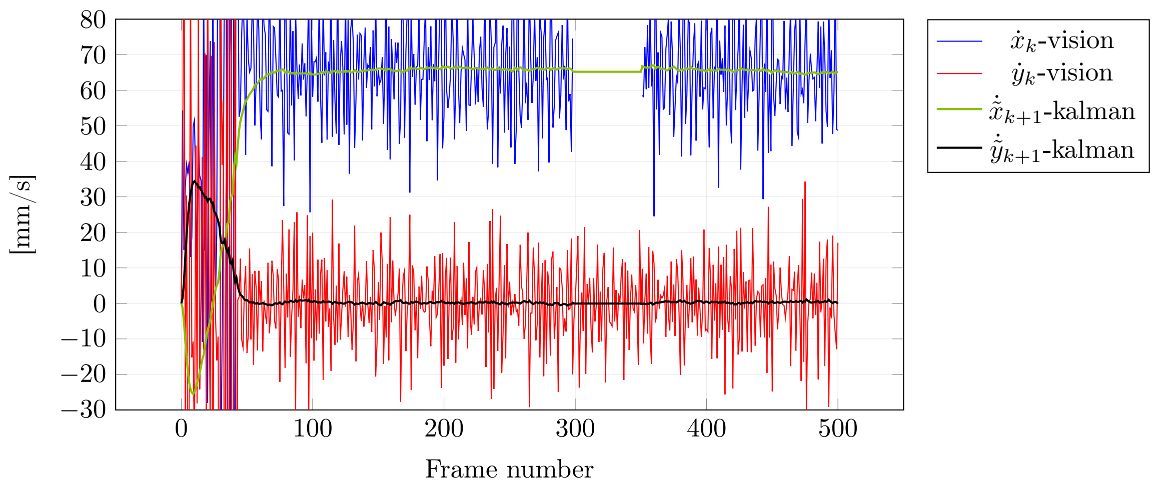


Figure 6.13: The estimated velocities from the Kalman filter and the velocities from the vision system, when converted to mm/s

For both the position prediction and velocity estimation, the initial error takes some samples before they are remove, because the Kalman filter parameters are still settling towards their steady state value. After around 50 frames the filter is completely settled, and since the camera is operating at 30 FPS, 50 frames equals 1.6 second. It is therefore necessary to wait for these parameters to settle before the control signals are forwarded to the robot. There is still some noise on the estimated velocities, but the tuning is a compromise between settling time and noise, the Kalman filter might require further tuning, once it is implemented on the robot.

The two remaining states are the orientation of the housing and the angular velocity. The orientation from the vision is already noisy, so it is important the Kalman filter removes some of this, furthermore the angular velocity is not calculated by the vision system, since numeric differentiation of a noise signal will only result in further noise. The orientation is shown on figure 6.14 and the angular velocity on figure 6.15.

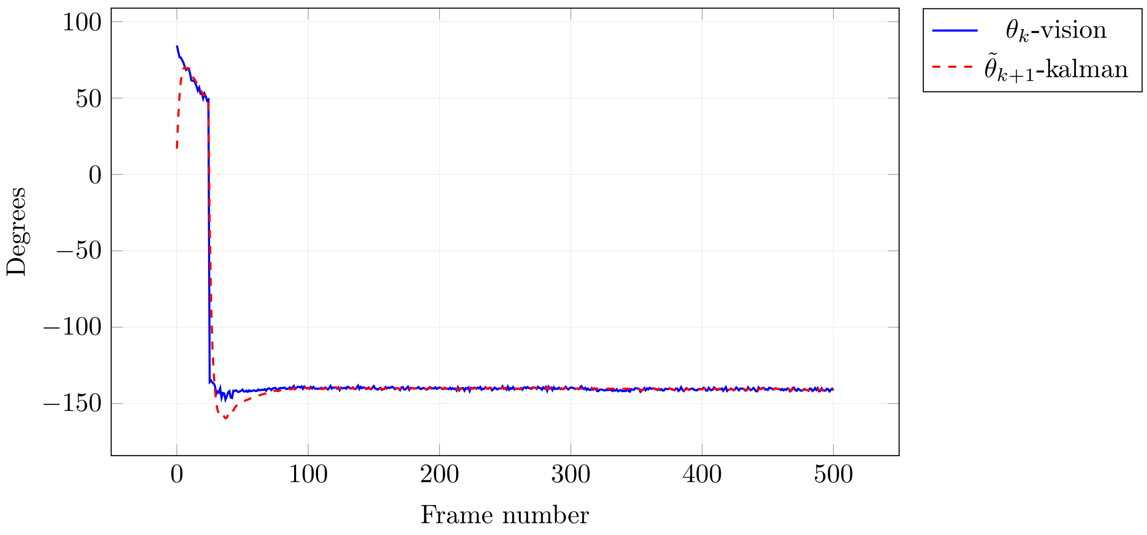


Figure 6.14: The predicted orientation and the orientation from the vision system

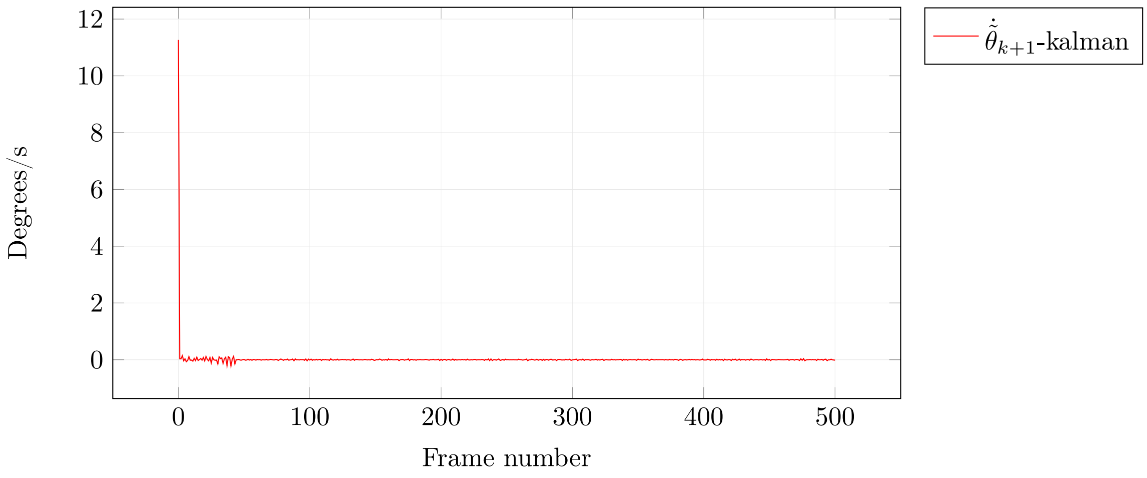


Figure 6.15: The estimated angular velocity

The predicted orientation is noiseless but has an overshoot before it settles. The angular velocity is oscillating around 0, which is to be expected.

The Kalman is not only able to predict future states, but also estimates the unknown states in the model. The resulting poles of the designed predictor is plotted on figure 6.16 along with the poles of the plant.

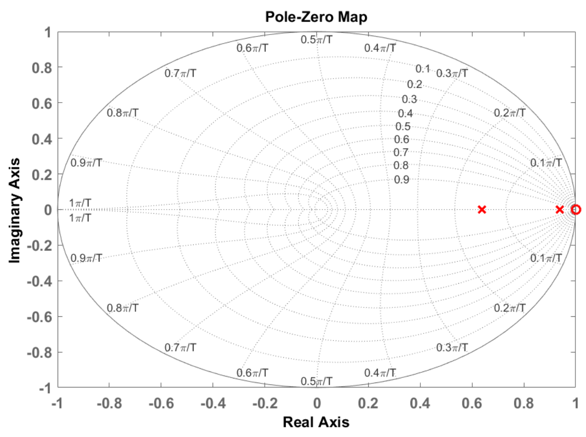


Figure 6.16: Pole-zero map of the closed loop system with the Kalman gain

There are in total six poles, three at 0.63 and three at 0.94 additionally there are three zeros, but they are all coincident at 1. This is not in complete agreement with the rule of thumb for designing a Kalman filter, but the poles at 0.94 are responsible filtering the signal, and if they are made faster, the settling time is reduced but larger overshoot and more noise is introduced, therefore it is decided to accept the current tuning for the filter. The weighing matrices are thereby determined to be:

$$\mathbf{V}_d = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix} \tag{6.2.11}$$

$$\mathbf{V}_n = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix} \tag{6.2.12}$$

With both a simulation model of the robot and a predictor for the housing's position on the conveyor belt, all the necessary steps for setting up a control strategy is finished.

Chapter 7

Control Strategy

As presented in chapter 4 the robot controller is already using unknown control strategies to ensure the performance of the robot. It is therefore necessary to investigate whether this control strategy is sufficient to track the reference generate with the Kalman filter in section 6.2 using the Cartesian values, such that the robot controller is doing the inverse kinematics. This is done using the emulated robot described in section 6.1.4 and where the prediction for the horizontal movement, $\tilde{x}_{k+1}$, is used since this has a ramp characteristics because the housing is only moving in one direction. The result of the emulated robot trying to track the prediction is shown on figure 7.1.

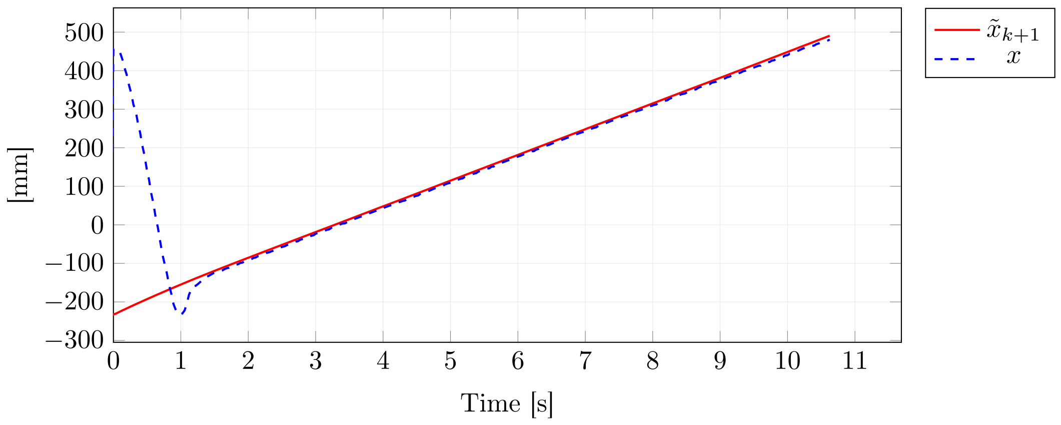


Figure 7.1: The result of the emulated robot trying to track the reference generated by the Kalman filter

The robot overshoots and then settles at a steady state value with an steady state error, which indicates the robot is only a type 1 system, and is therefore not able to track a ramp signal without an error. Because of this steady state error, the robots tracking error is around 8 [mm], which is more than the maximum allowed 1.5 [mm], it is therefore necessary to develop a control strategy to reduce the tracking error. Likewise the tracking is oscillating, which is expected to be caused by the maximum speed of the robot being set at a fixed 300 [mm/s], which is faster than the estimated $\dot{\tilde{x}}_{k+1}$ velocity. The challenge of this report, is the robot controller is a black box, so the purpose of this chapter is to develop an control strategy which can ensures the tracking error is below the maximum allowed error and that the robot is moving smoothly.

7.1 System overview

Before beginning to develop a control strategy and design the regulators, it is necessary to investigate the system. Beginning with the robot, for this design approach, the robot controller is viewed as a black box, meaning it is only possible to give it an input and measure its' output. As shown above, it is clear the robot controller's closed-loop system dynamics is a type 1 system because it has a steady state error on a ramp reference. This means the closed-loop system at most has one free integer, and the closed-loop system is reduced to a integer such that feedback system reduces to that of figure 7.2.

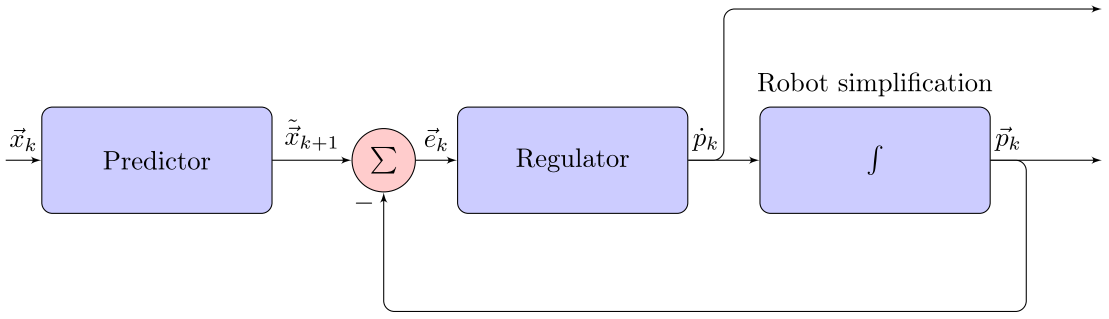


Figure 7.2: The simplified feedback system

This shows the feedback should be the position error between the robot end-effector and the predicted position of the centroid one sample ahead. The regulator which is to be designed, should then use this error to determine a position and velocity reference for the robot's end-effector.

7.2 Regulator design

For the regulator there are two considerations, which needs to be, the first is that while it is possible to define the maximum velocity, it is not possible to measure the actual velocity. Additional this maximum velocity is a global constraint, meaning it is not possible to set a velocity for the individual axis of end-effectors. Secondly, as described in section 6.1.4, it was not possible to derive a dynamic model of the robot, since the existing regulators on the robot controller are unknown. This is also why everything is treated as a black box, and therefore the regulators are chosen based on observations of the emulated robot's response and tuning of the controllers done iteratively using the emulated robot. Because of the limited control of the velocity, it is decided to split the regulator problem into a velocity- and a position regulator as shown on figure 7.3.

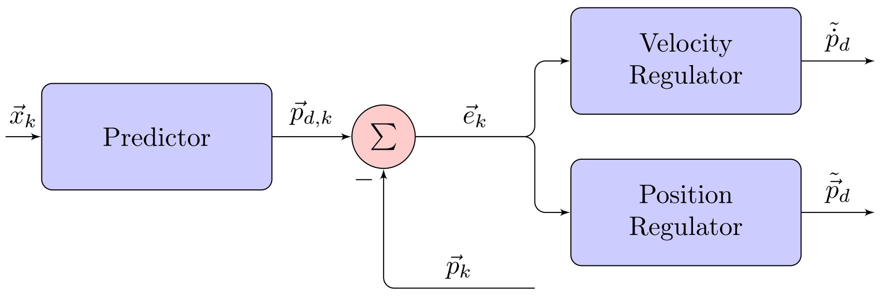


Figure 7.3: Overview of the two regulators

Where the position reference is redefined as $\vec{p}_d = \begin{bmatrix} x_d & y_d & \theta_d \end{bmatrix}^T = \begin{bmatrix} \tilde{x}_{k+1} & \tilde{y}_{k+1} & \tilde{\theta}_{k+1} \end{bmatrix}^T$.

7.2.1 Velocity Regulator

The Velocity Regulator has two purposes, minimize the convergence time, i.a. the time it takes before the robot is properly tracking the reference and ensuring the smoothness of the robot movement. The input for the controller is the position errors and the output is a 1D velocity for the robot. It is decided to use a PI-controller, since this ensures the error converges towards zero because of the integral term. The PI controller has the form:

$$\frac{G_1(s)}{e(s)} = K_{p,\dot{p}} + K_{i,,\dot{p}}\,\frac{1}{s} \tag{7.2.1}$$

It is necessary to discretize the integral part of the controller, before it is possible to implement it, this is done using the Trapezoidal method

$$\tilde{\dot{p}}(k) = \mathbf{K}_{p,\dot{p}}\,\vec{e}(k) + \mathbf{K}_{i,\dot{p}}\,\vec{E}(k) \tag{7.2.2}$$

$$\vec{E}(k) = \vec{E}(k-1) + \frac{T_s}{2}\left[\vec{e}(k) + \vec{e}(k-1)\right] \tag{7.2.3}$$

Where T_s is the sample time and $\vec{E}$ is the integrated error. The proportional gain $K_{p,\dot{p}}$ determines how aggressive the controller is and thereby how fast the robot settles. The integral term $K_{i,\dot{p}}$ ensures there velocity settles at a steady state value once the end-effector is tracking the trajectory without an error. The integral term also decreases the systems dampening, and could cause the velocity to oscillate as it settles, which is not desirable.

7.2.2 Position Regulator

The idea behind adding an position regulator is to remove the steady state error when the robot is trying to track a ramp input. The regulated should therefore converge towards a offset to add to the position reference, such that the robot ends up tracking the reference within the acceptable tracking errors. Since the Kalman filter estimates the velocity of the housing, it it also possible to do a feed-forward with these velocities. The feed-forward makes the overall regulator more responsive, since it is reacting to the velocity which is 90^o phase ahead of the position. Likewise to ensure the error converges to zero, a discrete PI controller is used.

$$\vec{G}_\delta = \dot{\vec{p}}_d\,T_s + \mathbf{K}_{p,p}\,\vec{e}(k) + \mathbf{K}_{i,p}\,\vec{E}(k) \tag{7.2.4}$$

The new position reference for the robot controller then becomes

$$\tilde{\vec{p}}_d = \vec{p}_d + \vec{G}_\delta \tag{7.2.5}$$

The two gains in the PI controller, should be choose such that they drive the steady state error towards zero.

7.2.3 Tuning of Regulators

The control strategy is to begin with kept simple, which means only the case of the housing placed on the conveyor belt is considered. In this case, the references for y and θ are as shown in section 6.2.1 constant. This means the regulators should focus on reducing the tracking error for x since this is the only changing variable.
The velocity regulator's two gains should be 1×3 row-matrices. To begin with the velocity reference is calculated only with the x- and y error, the gain therefore becomes

$$\mathbf{K}_{p,\dot{p}} = \begin{bmatrix}0.8 & 0.8 & 0\end{bmatrix} \tag{7.2.6}$$

This shows the proportional gain sums 80% of the two errors to determine the velocity reference. Secondly the integral term is determined to be significantly lower than proportional gain, such the proportional gain is dominating when the error is large and the integral is dominating when the error is small.

$$\mathbf{K}_{i,\dot{p}} = \begin{bmatrix}0.3 & 0.3 & 0\end{bmatrix} \tag{7.2.7}$$

The position regulator's gains are designed as if they are completely decoupled from each other, this means the gain matrices are diagonal matrices. When choosing the proportional gain matrix, it is important to consider, that the error initially is large, and therefore the proportional part of the controller should be set to a low value to ensure it does not increase the existing overshoot. While the robot is reaching it steady state tracking, the integral part of the regulator is integrating the error. This can be interpreted as stored energy, the integral gain should therefore be designed such that it is only releasing a small amount of this energy. The gains is defined as

$$\mathbf{K}_{p,p} = \begin{bmatrix}0.11 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix} \tag{7.2.8}$$

$$\mathbf{K}_{p,i} = \begin{bmatrix}0.01 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix} \tag{7.2.9}$$

It is decided only to use the PI controller for the x direction, since the two remaining references are step references, which the black box is already able to reach and adding additional dynamics is therefore not necessary.
It is now possible to implement the control strategy and designed regulators on the emulated robot, to evaluate the performance of the system. The position of the end-effector in the robot's base frame is shown on figure 7.4 along with the reference from the Kalman filter, $\vec{p}_d$, and the reference forwarded to the robot $\tilde{\vec{p}}_d$.

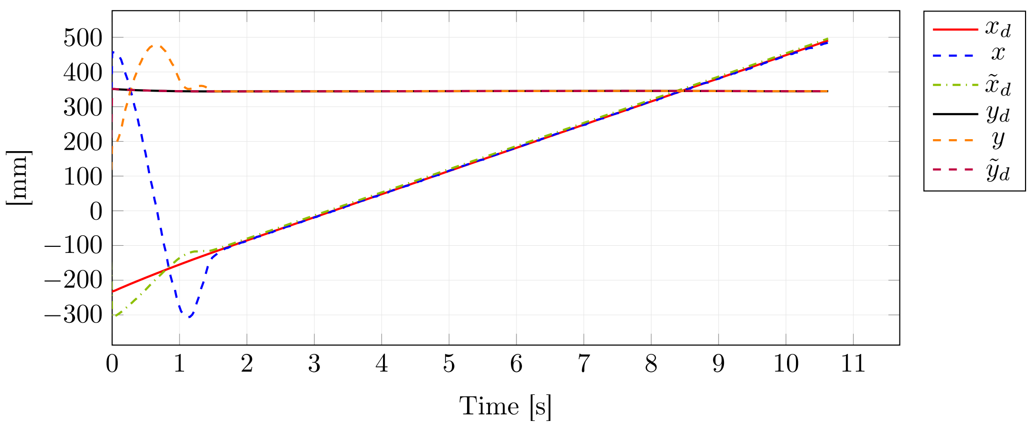


Figure 7.4: The end-effector's location in the plane plotted along with the reference generated by the vision system, and the modified reference

These plots shows how the control strategy has increased the overshoot slightly but it has improved the tracking error between the x_d reference and the end-effector's x-coordinate. Furthermore, it shows, as stated, the end-effector is able to track the y_d reference without a added PI controller. It is also necessary to verify the orientation of the end-effector is tracking its' reference, this is shown on figure 7.5.

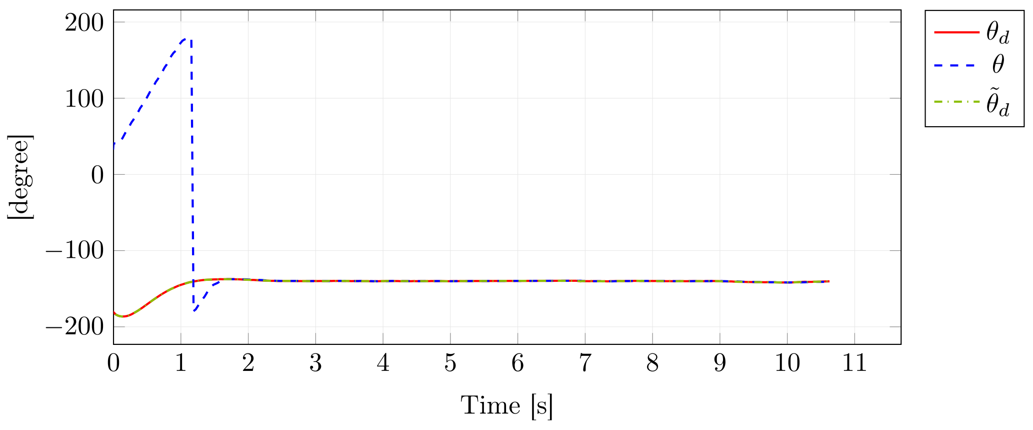


Figure 7.5: The end-effector's orientation plotted along with the reference generated by the vision system, and the modified reference

As with the y_d tracking, the orientation of the end-effector is tracking it reference with no visible error and overall the settling time for the end-effector is about 1.5 [s] which is acceptable. Next it is interesting to see the velocity reference for the robot, which is shown on figure 7.6.

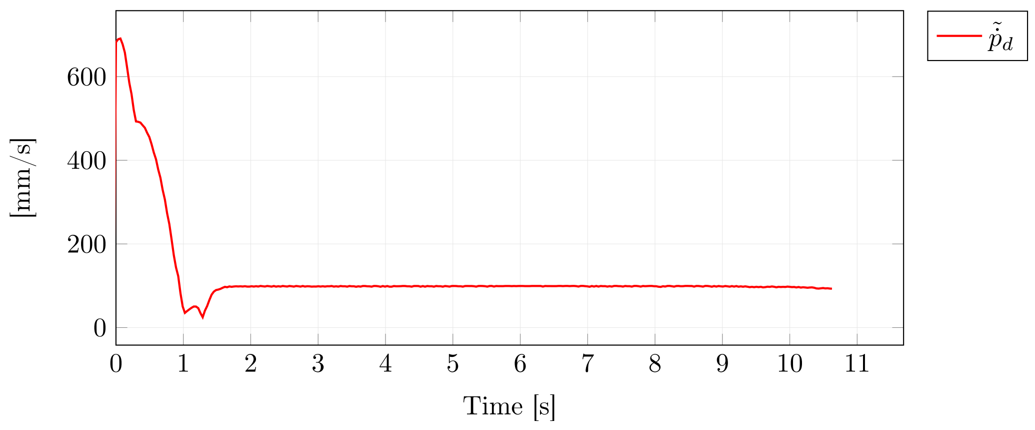


Figure 7.6: The velocity reference for the robot

It should be noted that the velocity reference is always positive, since ACE see it as a limit and not a target velocity. The velocity reference is initially high, and gradually lowers as the end-effector is nearing its' references. At the time of the overshoot, the velocity is slightly increased due to the increase in the error. The velocity reference settles at a steady state value, and due to the integrator effect it is expected there is no steady state error. Since the convergence time is acceptable the velocity controller is likewise acceptable, but the last test, is to plot the three tracking errors and verify that they are within the acceptable limits. This is done on figure 7.7.

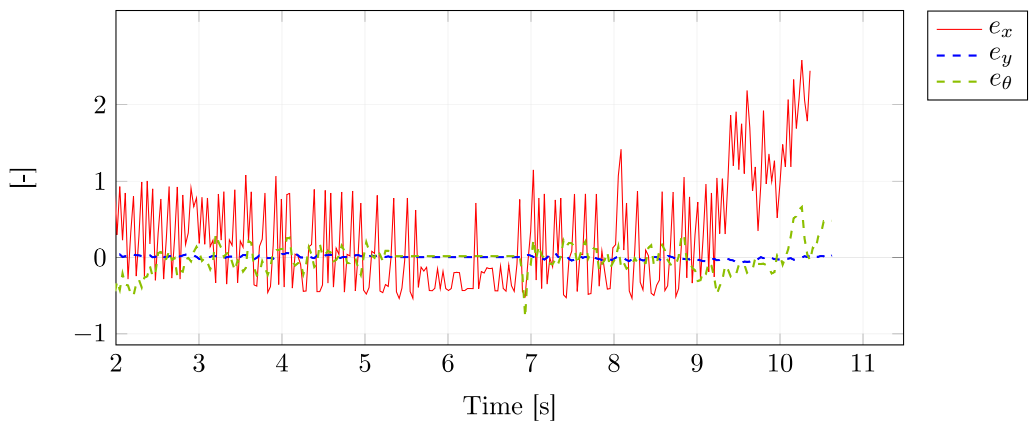


Figure 7.7: The three tracking errors plotted for the time $[2, 11]$, the errors are normalized with respect to their maximum allowed error

Besides a few outliers for the e_x error, all errors are within the limits of ± 1.5 [mm] for the position tracking and 1 [degree] for the orientation. After 9 [s] the housing is reaching the limits of the robot's available work space, and therefore the tracking error is increasing. This verifies that the control strategy and designed regulators are able to make the robot track the housing on the conveyor belt.

Chapter 8

Implementation

Now that the vision system, predictor, and regulator are all done, it is possible to setup a system overview for the complete visual servoing system and show the connections between the individual modules. This is done on figure 8.1.

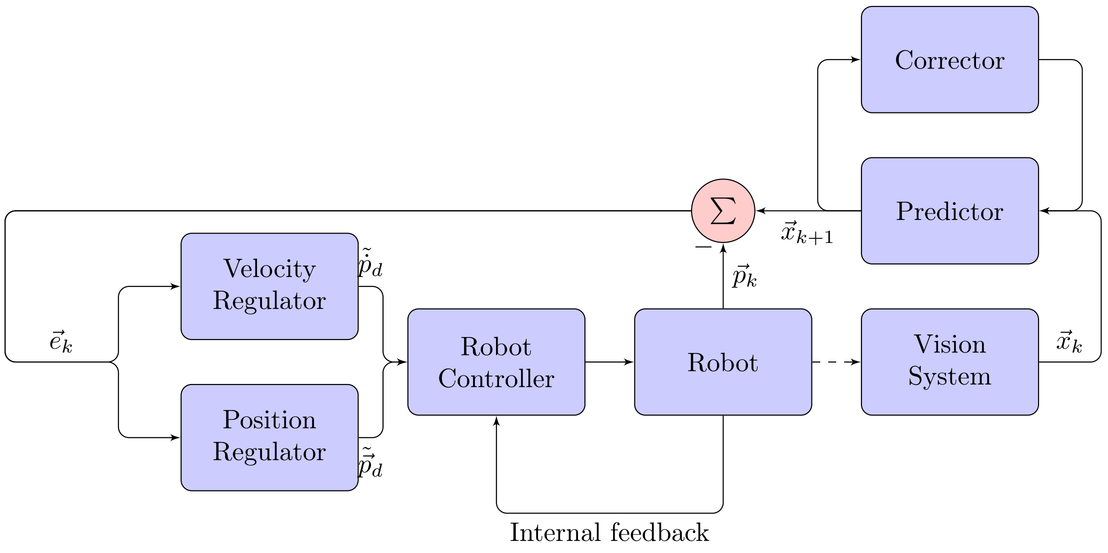


Figure 8.1: Complete visual servoing system overview

In the previous chapters where the individual modules was designed and tested, they used data form tests, and was therefore not interacting completely with each other. The purpose of this chapter is therefore to ensure the functionality of the visual servoing system. These task are

- Determine when the robot is tracking well enough to place the lidt and how to place the lid
- Ensuring the robot does not block the camera's view of the housing on the conveyor belt before it is time to place the lid
- Ensuring the Kalman filter is functioning even when the robot is blocking the camera's view

Once these task er completed, it is possible to implement the visual servoing system on the robot and test its' performance.

8.1 Visual servoing system

The visual servoing system is a combination of the modules described in the previous chapters, but they need to work in a structured way to ensure everything is functioning properly. It is therefore necessary to introduce some check and count statements to ensure the following steps are executed in the correct order.

1. The housing has to be fully within the region of interest, therefore the vision system set a parameter "detect" to 1 when the housing is detected, else detect=0

2. The visual servo system should not begin to command the robot, before the housing is within the robot's available workspace. The parameter "check" is introduced and is set to 1 if $|cos(q_2)| \leq 1$

3. The correction part of the Kalman filter should only be used when the detect parameter is 1, if the parameter is 0, then Kalman should relay on the prediction term. Secondly the Kalman initially needs time to converge, therefore a counter, Δ_d is counting upwards for each iteration the parameter detect=1. The constant Γ is introduced as a threshold for how many samples the Kalman filter needs to converge.

4. Because the conveyor belt is allowed to change velocity, it is necessary to maintain visual contact with the housing, the tracking reference from the Kalman filter is therefore offset with a parameter $\vec{\delta} = \begin{bmatrix} 200 & 0 & 0 \end{bmatrix}^T$. This ensures the robot is not blocking the camera's view but it will still track the movement of the housing.

5. Once the end-effector is tracking its' reference, it should move to the actual location of the housing. This movement is initiated by computing the average error vector for the last 15 samples and comparing the norm of this with the norm of the maximum allowed error.

$$\left| \sum_{i=0}^{14} \frac{\vec{e}(k-i)}{15} \right| \geq \left| \begin{bmatrix} x_{\max} & y_{\max} & \theta_{\max} \end{bmatrix}^T \right| \tag{8.1.1}$$

   Once this statement is no longer true, the robot should move towards the housing, by setting $\vec{\delta} = \vec{0}$. When this happens, the visual servoing system is relaying on the accuracy of the Kalman filter prediction

6. Since the end-effector has to move from the offset position to the actual position of the housing, it is again necessary to evaluate when it is properly converged. This is done as before, but only the previous five error vectors are used

$$\left| \sum_{i=0}^{4} \frac{\vec{e}(k-i)}{5} \right| \leq \left| \begin{bmatrix} x_{\max} & y_{\max} & \theta_{\max} \end{bmatrix}^T \right| \tag{8.1.2}$$

   When this statement is true, the q_3 is set such that the lid is lowered down to the housing. Because the end-effector is always moving towards the predicted future position of the housing, the lid is lowered with a speed of

$$\dot{p}_a = (q_{3,1} - q_{3,0})/T_s \tag{8.1.3}$$

   Where $q_{3,1}$ is the initial value of q_3 and $q_{3,0}$ is the final value. This ensure the lid is lowered and released onto the housing exactly when the housing's centroid is below the end-effector.

The following two pages shows the sequence diagrams for the visual servoing system.

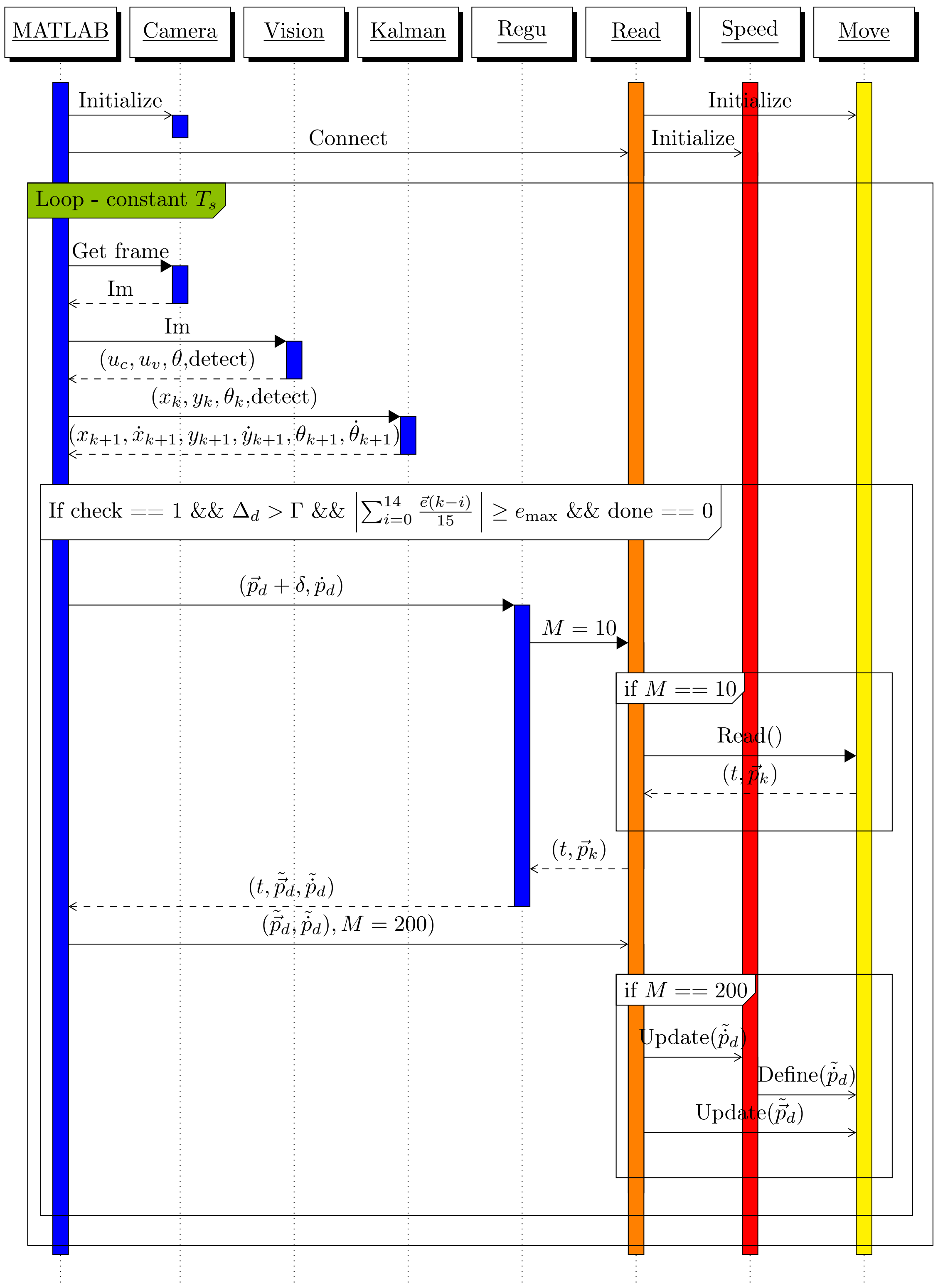

MATLAB
Camera
Vision
Kalman
Regu
Read
Speed
Move
Initialize
Connect
Initialize
Initialize
Loop - constant T_s
Get frame
Im
Im
$(u_c, u_v, \theta,$detect$)$
$(x_k, y_k, \theta_k,$detect$)$
$(x_{k+1}, \dot{x}_{k+1}, y_{k+1}, \dot{y}_{k+1}, \theta_{k+1}, \dot{\theta}_{k+1})$
If check $== 1$ && $\Delta_d > \Gamma$ && $\left|\sum_{i=0}^{14} \frac{\vec{e}(k-i)}{15}\right| \geq e_{\max}$ && done $== 0$
$(\vec{p}_d + \delta, \dot{p}_d)$
$M = 10$
if $M == 10$
Read()
$(t, \vec{p}_k)$
$(t, \vec{p}_k)$
$(t, \tilde{\vec{p}}_d, \tilde{\dot{p}}_d)$
$(\tilde{\vec{p}}_d, \tilde{\dot{p}}_d), M = 200)$
if $M == 200$
Update$(\tilde{\dot{p}}_d)$
Define$(\tilde{\dot{p}}_d)$
Update$(\tilde{\vec{p}}_d)$

Figure 8.2: Sequence diagram for the visual servoing system

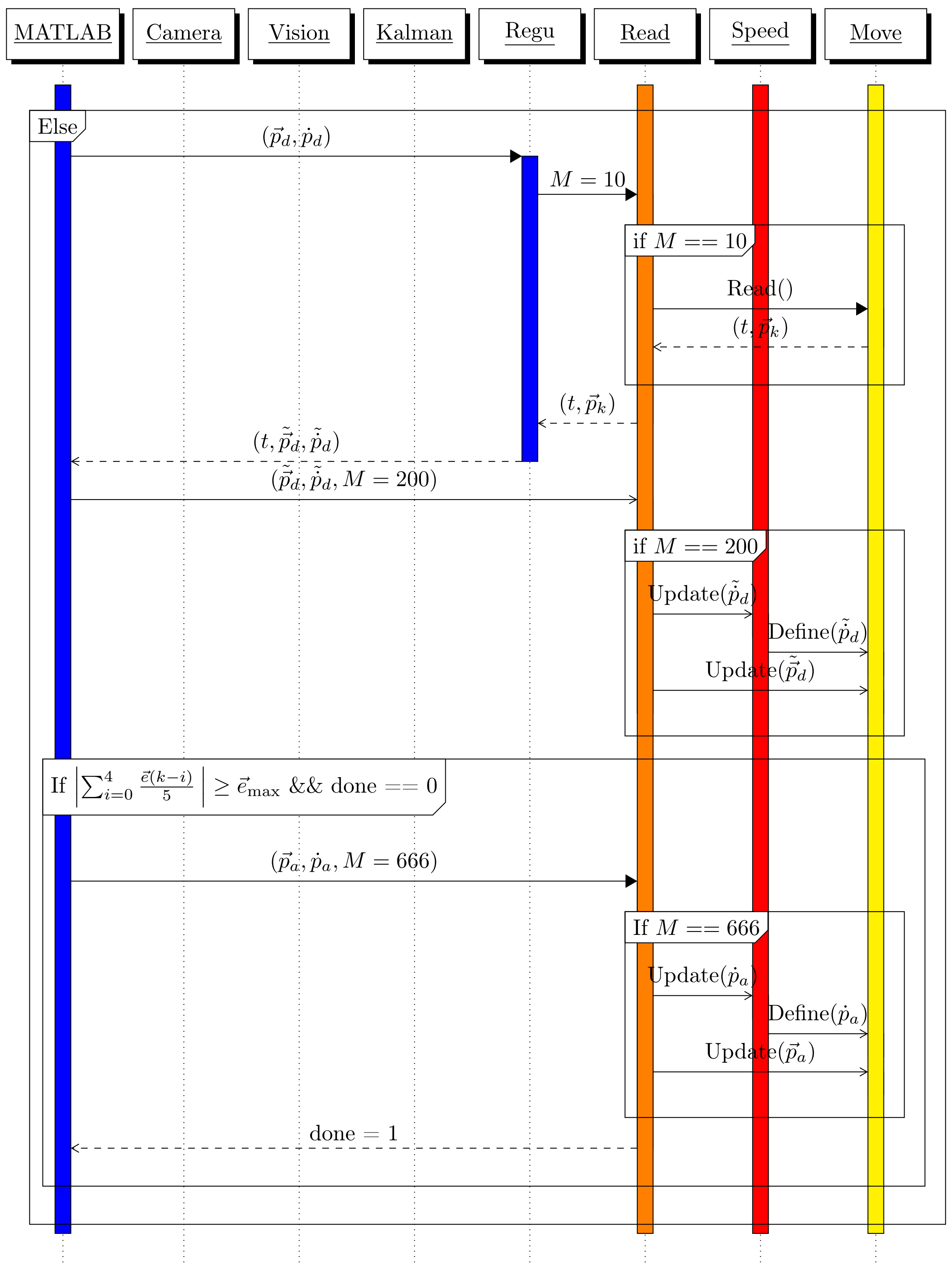

MATLAB
Camera
Vision
Kalman
Regu
Read
Speed
Move
Else
$(\vec{p}_d, \dot{p}_d)$
$M = 10$
if $M == 10$
Read()
$(t, \vec{p}_k)$
$(t, \vec{p}_k)$
$(t, \tilde{\vec{p}}_d, \tilde{\dot{p}}_d)$
$(\tilde{\vec{p}}_d, \tilde{\dot{p}}_d, M = 200)$
if $M == 200$
Update($\tilde{\dot{p}}_d$)
Define($\tilde{\dot{p}}_d$)
Update($\tilde{\vec{p}}_d$)
If $\left|\sum_{i=0}^{4} \frac{\vec{e}(k-i)}{5}\right| \geq \vec{e}_{\max}$ && done $== 0$
$(\vec{p}_a, \dot{p}_a, M = 666)$
If $M == 666$
Update($\dot{p}_a$)
Define($\dot{p}_a$)
Update($\vec{p}_a$)
done $= 1$

Figure 8.3: Sequence diagram for the robot placing the lid on the housing. This is an extension to figure 8.2 and is contained within the Loop

Figure 8.2 shows initialization phase in the beginning, where MATLAT connects to the camera and the robot controller. As described in chapter 4.1, the robot controller is running three continuous task, and these also have an initialization phase, where the robot is readied and moved into its' starting position. Once this phase is completed, the visual servoing systems moves into the main loop, and follows the steps previously described. It should be noted, that figure 8.3 is also within the main loop, this sequence diagram shows how the visual servoing system handles placement of the lid on the housing. Once the systems enters this phase, it is relying on the accuracy of the Kalman filter. If the conveyor speed changes velocity during this phase, the placement of the lid will fail. this is also why this phase is only checking the previous five error vectors, such that, as soon as the robot is converged, it places the lid. Once this part of the sequence diagram is done, the parameter "done" is set to 1, which ensures the systems reenters the first "if" part of the sequence diagram and ensures the end-effector returns to tracking the offset reference.

8.1.1 Simulation Results

Before implementing the visual system om the actual robot, the placement of the lid part of the sequence diagram is tested on the emulated robot to ensure everything is functioning as expected. The position of the end-effector in the plane along with its' reference and the prediction position of the housing is shown on figure 8.4

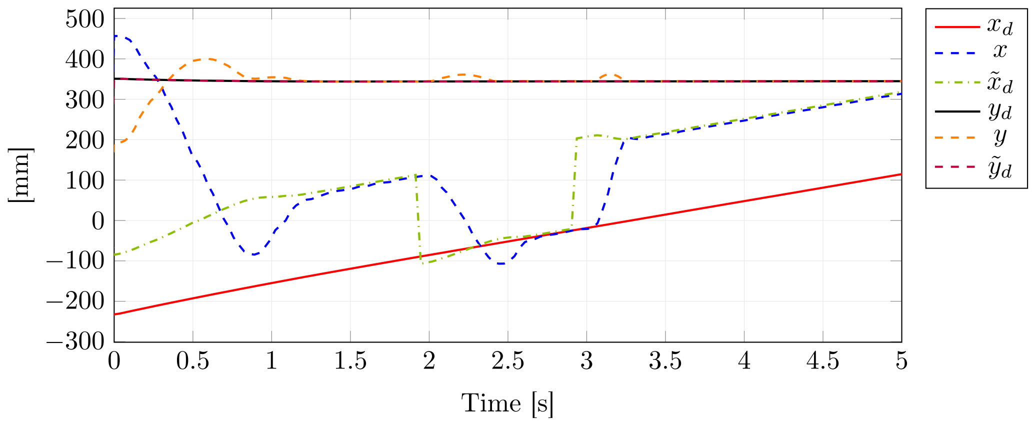


Figure 8.4: The end-effector's location in the plane plotted along with the reference generated by the vision system, and the modified reference

As expected, the robot is tracking a position, which is offset 200 [mm] in the x-direction. Once the tracking error is properly reduced, the end-effector in the span of around 1 [s] moves to the housing, places the lid, and moves away. It is necessary to ensure that the orientation of the end-effector maintains its' reference throughout this process, therefore the orientation is plotted on figure 8.5.

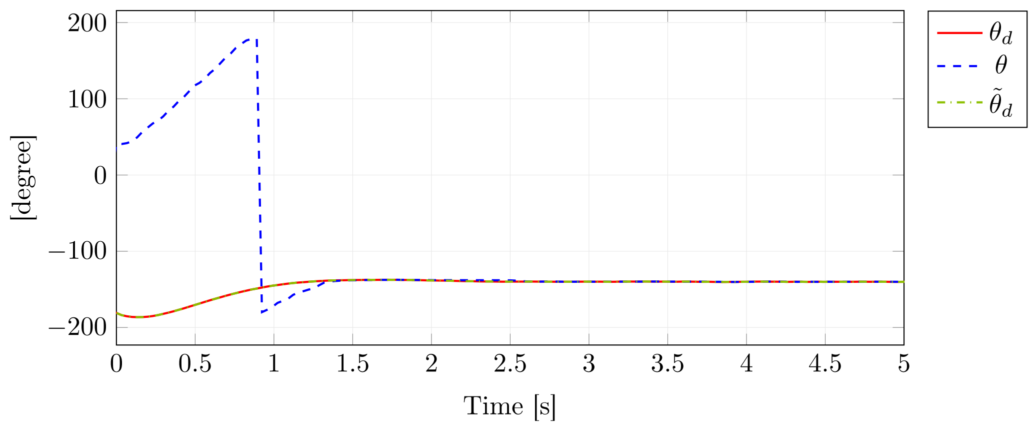


Figure 8.5: The end-effector's orientation plotted along with the reference generated by the vision system, and the modified reference

There is no deviation from the orientation reference, so this concludes that given the conveyor speed remains constant while the end-effector is moving from the offset position and has finished placing the lid. The visual servoing system should be able to place the lid on the housing.

8.2 Implementation results

In all the data used for the simulations, the camera was operating at 30 FPS, and visual servoing system's main loop was therefore previously operating at $T_s = 1/30$. When implementing the system on the robot, it turned out that the visual system was not able to maintain this sample rate, since the time needed for the communication and computation were larger than the established T_s. This is probably due to both the time needed for the image acquisition is larger when operating live rather than getting the frames of a video. Also the communication with the actual robot was slower than when using the emulated robot. These two factors together meant the frame rate was reduced to 10 FPS and thereby $T_s = 1/10$. Additionally the robot was not equipped with a suction cub as an end-effector, when the test performed, therefore it is not possible to verify that the lid is physically placed on the housing, but only if the robot is able to reach it goals. Two tests are carried out on the robot to evaluate the visual servoings performance. Firstly the robot is tracking the housing where the tracking reference is offset to ensure there is visual contact. Secondly a placement test is carried out, where the full visual system in implemented, such that tries to place the lid on the housing. The following figures for position and orientation, also shows the detected values from the vision system and when the values from the vision set to $\begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}^T = \begin{bmatrix} -300\,[\text{mm}] & 443\,[\text{mm}] & 0\,[\text{degree}] \end{bmatrix}^T$. Once the initialization is completed, MATLAB notifies the user, and the housing is placed arbitrary on the conveyor belt. Figure 8.6 shows the end-effector position during the tracking test.

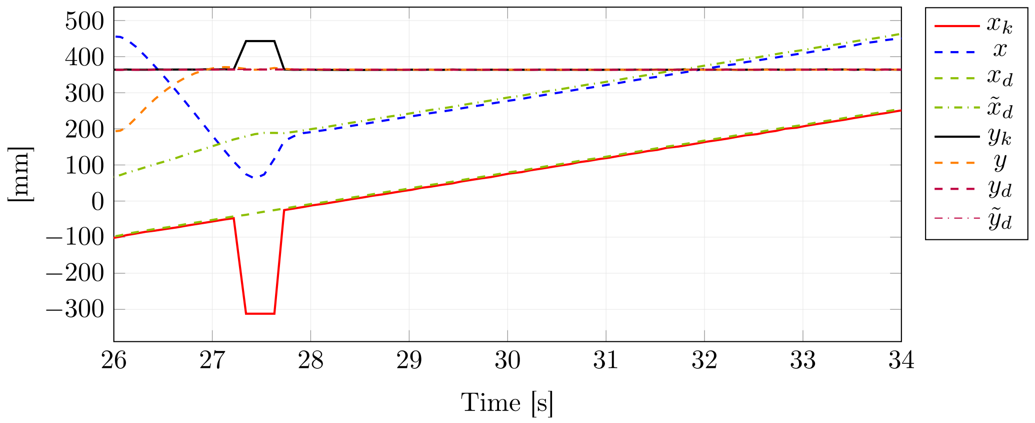


Figure 8.6: The end-effector's location in the plane plotted along with the reference generated by the vision system, Kalman filter, and the modified reference

As with the simulation, the robot has an initial overshoot, before it settles and tracks the offset reference. When the overshoot occurs, the robot is blocking the camera, and it is clear the vision system looses contact with the housing. This also shows the Kalman filter is not affected by this since the prediction continues and when the vision system regains contact with the housing, the Kalman filter is still on track. The orientation of the end-effector is shown on figure 8.7.

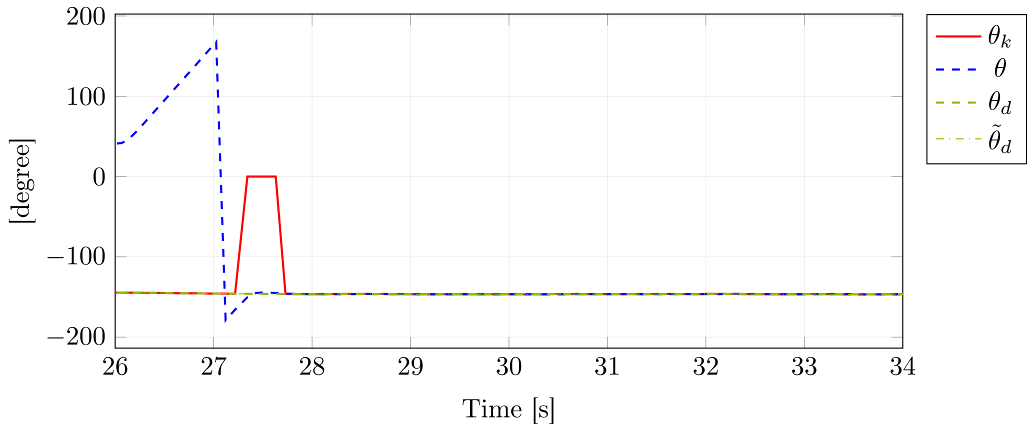


Figure 8.7: The end-effector's orientation plotted along with the reference generated by the vision system, Kalman filter and the modified reference

As in the simulation, there is a an initial increase in the orientation, but the orientation is quick to reach its' reference. As with the position, the Kalman filter is not affected when the robot is blocking the camera, and which ensures the orientation is not changing. The velocity reference is shown on figure 8.8.

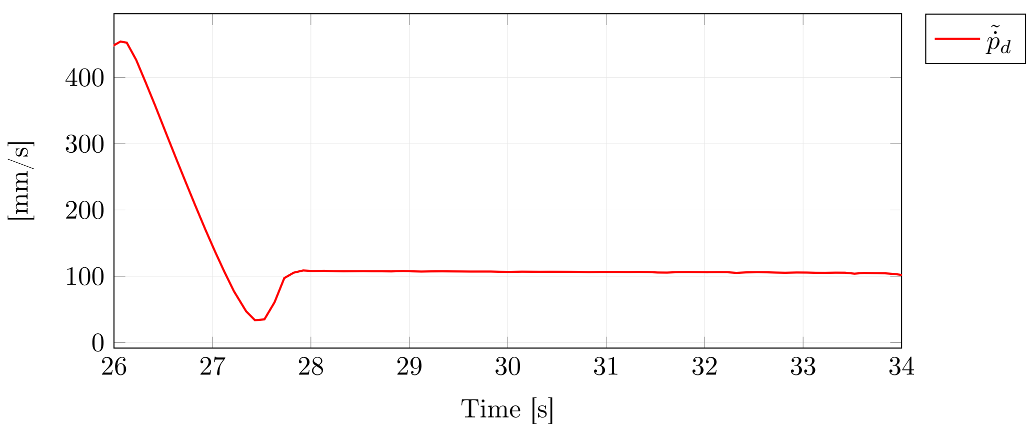


Figure 8.8: The velocity reference for the robot

Once the end-effector settles, the velocity reaches its' steady value, which is also to be expected. Lastly it is necessary to check the tracking error to ensure it is smaller than the maximum allowed.

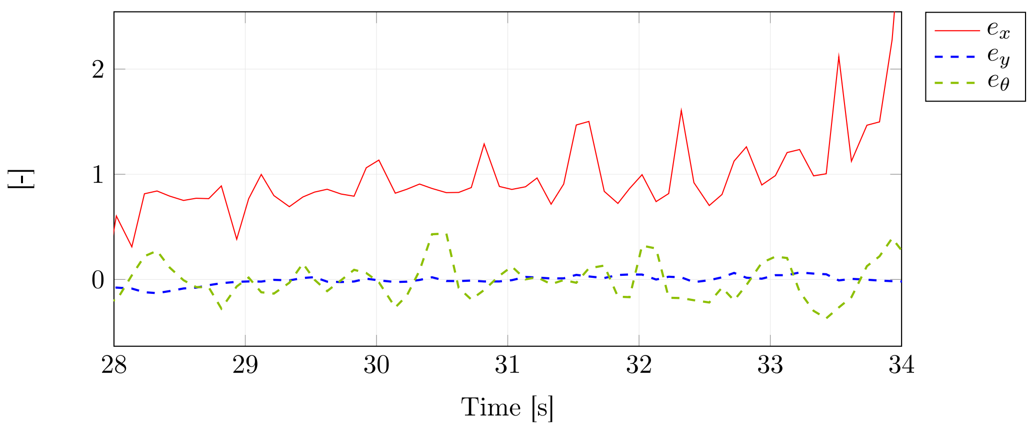


Figure 8.9: The three tracking errors plotted for the time $[28, 34]$, the errors are normalized with respect to their maximum allowed error

The actual robot is not able to keep the tracking error as low as the emulated robot, but it is decided to continue because error in this phase does not affect the placement of the lid. This leads testing the full visual servoing system and check if the lid is placed successfully. Figure 8.10 shows the end-effector position.

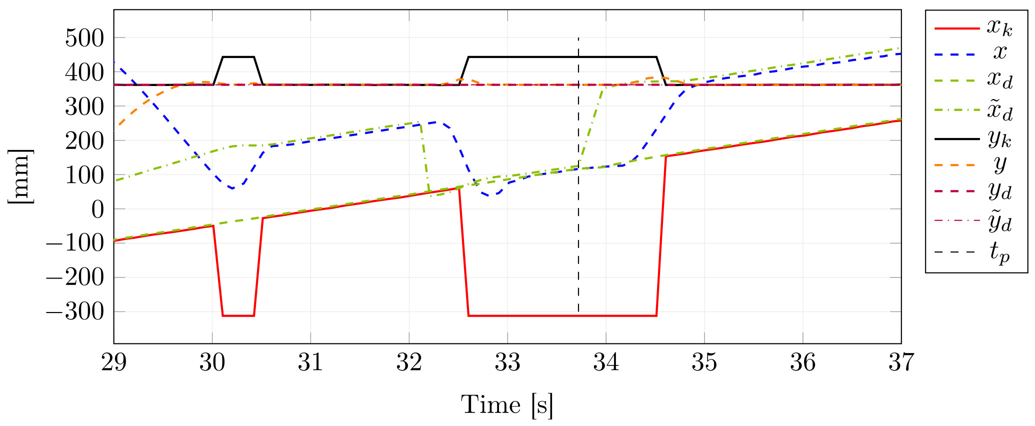


Figure 8.10: The end-effector's location in the plane plotted along with the reference generated by the vision system, Kalman filter and the modified reference

At around 32 [s] the robot enters the second part of the sequence diagram, and the reference is moved from the offset position to the predicted position of the centroid. As mentioned previously, the system depends completely in the Kalman filter for this part of the visual servoing system. Once the vision system again is able to detect the housing, it is clear the Kalman filters prediction is correct. It takes the visual servoing system about 1.5 [s] from the reference is changed to the lid is placed at the time t_p, which is acceptable. Next it is necessary to verify the orientation does not change during the placing of the lid, the orientation of the end-effector is shown on figure 8.11

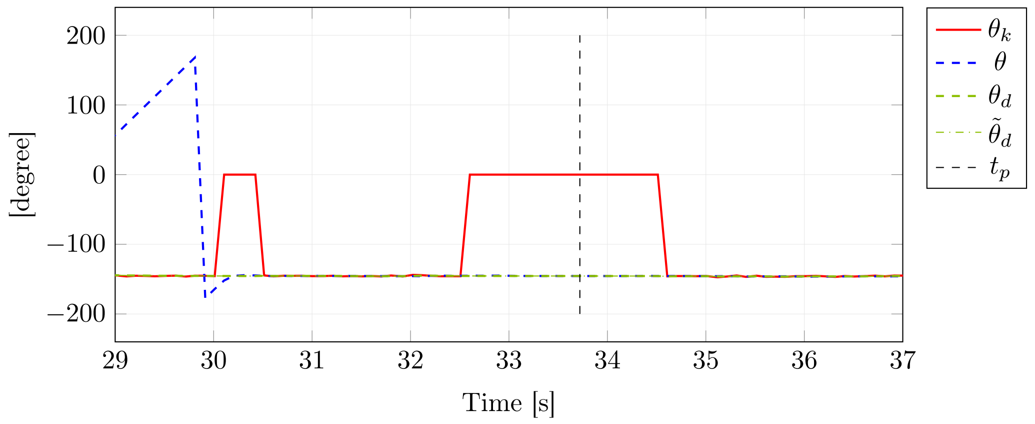


Figure 8.11: The end-effector's orientation plotted along with the reference generated by the vision system, Kalman filter and the modified reference

As before, the Kalman filter ensure the orientation of the end-effector does not change. Since the visual servoing system only places the lid if the error statement is below the threshold, the lid should be placed such that none of the errors are above the maximum allowed errors, but it is still necessary to check the errors. This is done on figure 8.12 where the normalized errors are plotted.

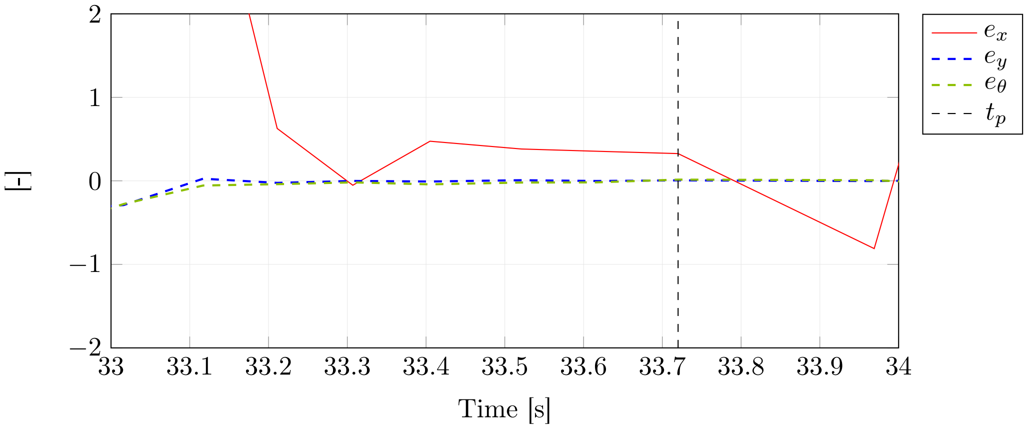


Figure 8.12: The three tracking errors plotted for the time $[28, 34]$, the errors are normalized with respect to their maximum allowed error

As expected, the errors are all below their thresholds and therefore it is the lid is expected to be placed correctly on the housing for this test of the visual servoing system. Lastly the velocity reference is shown on figure 8.13.

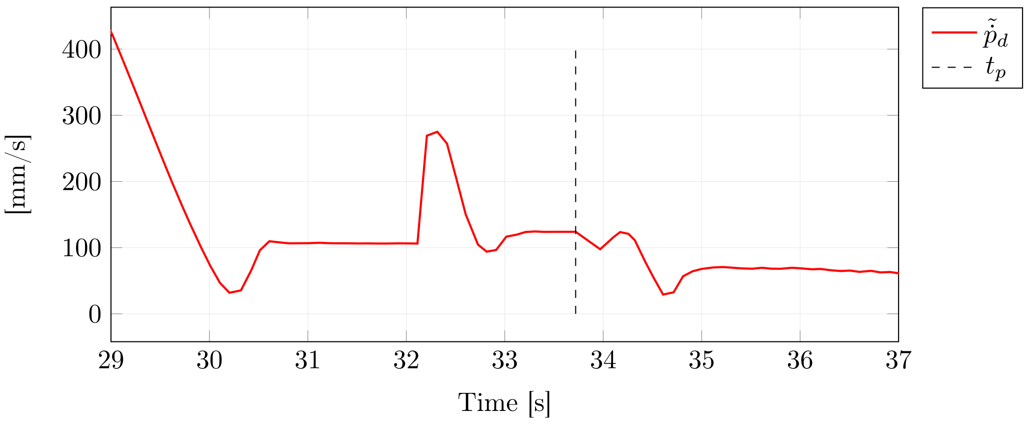


Figure 8.13: The velocity reference for the robot

As expected the velocity reference is increased when the robot has to move from the offset position to the new reference and the velocity settles as the robot reaches its' new reference. Ideally the velocity reference should also have a peak when the robot's reference is changed back to the offset reference, but this is not the case. One explanation could be that the integrator has wound up an large error and when the reference changes, so does the sign of the error, which means the integrator has to unwind, which works against the proportional part of the PI controller.
These two test prove that when all the develop modules are put together as a single visual servoing system, the robot is able to perform task, based on a visual feedback with a precision that is within the requirements for the system. These test are in the mean time not thoroughly enough to prove the robustness of the visual servoing system, which would require that the conveyor belt's speed change during the tracking. This is not done in this report and is left for future work.

Chapter 9

Conclusion

Through an analysis of the Smart Lab and the project delimitation is was determined that the operation which could benefit from visual servoing, was placing the lids on the housing. It was a requirement from Aalborg University that the visual servoing system needs to be both configurable and efficient, which lead to the requirement that the visual servoing system should be able to operate without information about conveyor speed or position and orientation of the housing. This leads to the proposed visual servoing scheme, where the error between the robot's end-effector and the predicted position of the housing should be reduced to zero.
Because the industrial robot controller is not made as an open unit, it limited the available possibilities for the how to control the robot and it was determined the only available approach was to continuously tell the robot controller to move to a point, while in two separate task respectively updating the position and maximum allowed velocity. To determine this position reference for the robot, a vision system was developed, which with a USB camera is able to provide continuous visual feedback. It was found that by converting the frames RGB channels to HSI, the vision is robust both towards changes in illumination and if the robot enters the field of view. To ensure the detected centroid defined in pixels also match the actual centroid of the housing, it is necessary to remove the distortion. This was done by determining the intrinsic parameters of the camera, with which it is possible to compensate for distortion, such that the camera can be treated as a pinhole camera. To transform this centroid to the robot's base frame, such that error between this and the end-effector is determinable, it was not robust to setup a global coordinate system as reference for the vision system and robot's base frame. Instead it was possible to take advantage of the conveyor plane being parallel with the image plane, such that two transformation vectors could be derived. With the detected centroid and the end-effector defined in the same frame, it was then necessary to tune a Kalman filter, which is used as the predictor for the future position of the housing. When comparing the Kalman filter's prediction towards the data from the vision system, it was concluded the prediction was precise and the convergence of the Kalman filter was faster than the requirement of 2 [s]. This prediction was then used as a reference for the robot's end-effector and it was decided to use a combination of a feed-forward and PI controller to determine the Cartesian reference, which ensures the robot's end-effector is able to track the reference and stay within the allowed errors. The simulations showed this control scheme was able to fullfill the requirements and the visual servoing system was therefore implemented on the actual robot. The actual robot was not equipped with a suction cup which limits the verifications to ensuring the robot's end-effector at the time of placing the lid, is within the allowed errors. The tests yielded the same results as the emulated robot, and the visual servoing system was able to track the reference with low enough errors to enter the final placement part of the system. Likewise it was shown that at the time of placing the lid, the normalized errors was well below the maximum allow errors. likewise the robot

converged on its' reference within the allowed time and but there was observed small oscillations of the end-effector. The visual servoing system is functioning, but it is still necessary to further test the robust of it by performing more extensive test, where the conveyor speed and direction is changed. But the problem statement was that by using the proposed control scheme, whether it is possible to turn the robot and a camera into a visual servoing system. From the observed results it is believed that if the robot is equipped with a suction cup, the designed visual seroving system is able to place the lid on the housing and with further tuning the oscillations can be removed.

Chapter 10

Future Work

The purpose of this chapter is to discuss topics for future work, which could improve the performance of the visual servoing system. The chapter will only present ideas and arguments for why the idea might work, but will not work towards solutions

10.1 Adaptive Control

Because the robot controller is a black box and this limits the available methods for controlling the robot, it could make the control scheme more robust, by adding a adaptive controller to the control strategy. This is adaptive controller would be more robust towards changes in the conveyor speed, which in turn will make the visual servoing system more robust.

10.2 Improved Feature detection

The blob detecting in the vision system, fulfilling all the requirements for the vision system, but it does not complete align with the desire for configurability since it still requires test to determine the proper thresholding values for each of the different colors the housing is available in. This need could be removed by instead of blob detecting, the vision system could use edge detection for determining position and orientation. This however would also detect the robot, when it moves into the region of interest, but this could be solved by using the known geometry and joint configuration, the robot could be mapped onto the image plane. Thereby all the pixels which associated with the robot at a given joint configuration, could be set to 0 thereby removing them from the edge detection.

Bibliography

Adept (2017). Adept cobra scara robots. `http://www.adept.com/products/robots/scara/cobra-s600/downloads`. Mar 2017. 8, 9, 13, 87

Caltech (2017). Camera calibration toolbox for matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/parameters.html`. May 2017. 23

Corke, P. (2013). *Robotics, Vision and control*. Springer, 2 edition. 24, 32, 33

Corke, P. I. (1996). *Visual Control of Robots*. Springer, 2 edition. 1, 10, 47, 48

Corke, P. I. (2017). Machine vision toolbox. `http://petercorke.com/wordpress/toolboxes/machine-vision-toolbox`. May 2017. 19

De Luca, P. A. (2017). Robotics i. `http://www.diag.uniroma1.it/~deluca/rob1_en/material_rob1_en.html`. Mar 2017. 15

Logitech (2017). Logitech c920. `https://www.logitech.com/da-dk/product/hd-pro-webcam-c920`. May 2017. 13, 20

Madsen, O. and Møller, C. (2017). The aau smart production laboratory for teaching and research in emerging digital manufacturing technologies. *Science Direct*. May 2017. 3, 7

Mathworks (2017). Matlab. `https://se.mathworks.com/`. May 2017. 13

Moeslund, T. B. (2012). *Introduction to Video and Image Processing*. Springer, 1 edition. 29, 36

SPARC (2015). Multi-annual roadmap. *Strategic Research Agenda*. 1

Welch, G. and Bishop, G. (2006). An introduction to the kalman filter. *UNC-Chapel Hill*. 48

Annex A

Diviation of transformation matrix

From section 6.1.1 the DH table is defined as And the general DH matrix is:

i	α_i	a_i	d_i	θ_i
1	0	a_1	d_1	q_1
2	0	a_2	0	q_2
3	0	0	q_3	0
4	α_4	a_4	d_4	q_4

Table A.1: The David-Hartenberg parameters for the Cobra s600

$$^{i-1}\mathbf{A}_i(\vec{q}) = \begin{bmatrix} c\theta_i & -c\alpha_i\, s\theta_i & s\alpha_i\, s\theta_i & a_i\, c\theta_i \\ s\theta_i & c\alpha_i\, c\theta_i & -s\alpha_i\, c\theta_i & a_i\, s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.0.1}$$

To derive the transformation matrix from the base of the robot to tool tip, it is necessary to substitute into the general DH matrix which yields four transformation matrices. The matrices are written in super compact form, which means $c_1 = \cos(q_1)$ and $s_{12} = \sin(q_1 + q_2)$:

$$^{0}\mathbf{A}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1\, c_1 \\ s_1 & c_1 & 0 & a_1\, s_1 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.0.2}$$

$$^{1}\mathbf{A}_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2\, c_2 \\ s_2 & c_2 & 0 & a_2\, s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.0.3}$$

$$^{2}\mathbf{A}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.0.4}$$

$$^{3}\mathbf{A}_4 = \begin{bmatrix} c_4 & -c_{\alpha_4}\, s_4 & s_{\alpha_4}\, s_4 & a_4\, c_4 \\ s_4 & c_{\alpha_4}\, c_2 & -s_{\alpha_4} c_4 & a_4\, s_4 \\ 0 & s_{\alpha_4} & c_{\alpha_4} & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.0.5}$$

These transformation matrices is now multiplied to derive the desired matrix.

$$ {}^0\mathbf{A}_2 = {}^0\mathbf{A}_1\,{}^1\mathbf{A}_2 = \begin{bmatrix} c_{12} & -s_{12} & 0 & a_1\,c_1 + a_2\,c_{12} \\ s_{12} & c_{12} & 0 & a_1\,s_1 + a_2\,s_{12} \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.0.6} $$

$$ {}^0\mathbf{A}_3 = {}^0\mathbf{A}_2\,{}^2\mathbf{A}_3 = \begin{bmatrix} c_{12} & -s_{12} & 0 & a_1\,c_1 + a_2\,c_{12} \\ s_{12} & c_{12} & 0 & a_1\,s_1 + a_2\,s_{12} \\ 0 & 0 & 1 & d_1 + q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.0.7} $$

$$ {}^0\mathbf{A}_4 = {}^0\mathbf{A}_3\,{}^3\mathbf{A}_4 = \begin{bmatrix} c_{124} & -c_{\alpha_4}\,s_{124} & s_{\alpha_4}\,s_{124} & a_1\,c_1 + a_2\,c_{12} + a_4\,c_{124} \\ s_{124} & c_{\alpha_4}\,c_{124} & -s_{\alpha_4}\,c_{124} & a_1\,s_1 + a_2\,s_{12} + a_4\,s_{124} \\ 0 & s_{\alpha_4} & c_{\alpha_4} & d_1 + q_3 + d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.0.8} $$

Equation (A.0.8) is the transformation matrix from the base to the end-effector.

Annex B

Inverse Kinematics

The purpose of this chapter is to derive the algebraic inverse kinematics for the SCARA robot, which connects a point in the Cartesian space to a set of joint values. Using the final direct kinematics transformation matrix from equation (A.0.8), and defining $\alpha_4 = a_4 = 0$ it reduces to:

$$ {}^{0}\mathbf{A}_4 = \begin{bmatrix} c_{124} & -c_{\alpha_4}\, s_{124} & s_{\alpha_4}\, s_{124} & a_1\, c_1 + a_2\, c_{12} + a_4\, c_{124} \\ s_{124} & c_{\alpha_4}\, c_{124} & -s_{\alpha_4}\, c_{124} & a_1\, s_1 + a_2\, s_{12} + a_4\, s_{124} \\ 0 & s_{\alpha_4} & c_{\alpha_4} & d_1 + q_3 + d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.0.1}$$

The last column of this matrix is the normalised position vector from the base to the end-effector, this is used to define the following expressions together with the orientation of the end-effector:

$$X = a_1\, c_1 + a_2\, c_{12} \tag{B.0.2}$$

$$Y = a_1\, s_1 + a_2\, s_{12} \tag{B.0.3}$$

$$Z = q_3 + d_3 + d_4 \tag{B.0.4}$$

$$\phi = q_1 + q_2 + q_4 \tag{B.0.5}$$

These equations are manipulated such that it is possible to setup an algebraic expression for each q_i defined by the global coordinates (X, Y, Z).

Expression for q_2

To define an expression for q_2 the first step is the following:

$$\begin{aligned} X^2 + Y^2 &= (a_1\, c_1)^2 + (a_2\, c_{12})^2 + 2\, a_1\, a_2\, c_1\, c_{12} + (a_1\, s_1)^2 + (a_2\, s_{12})^2 + 2\, a_1\, a_2\, s_1\, s_{12} \\ &= a_1^2(c_1^2 + s_1^2) + a_2^2(c_{12}^2 + s_{12}^2) + 2\, a_1\, a_2(c_1\, c_{12} + s_1\, s_{12}) \end{aligned} \tag{B.0.6}$$

This expression is reduced using the two following trigonometric relations

$$cos(\Theta)^2 + \sin(\Theta)^2 = 1 \tag{B.0.7}$$

$$\cos(\Theta)\, \cos(\Phi) + \sin(\Theta) \sin(\Phi) = \cos(\Theta - \Phi) \tag{B.0.8}$$

Which leads to

$$X^2 + Y^2 = a_1^2 + a_2^2 + 2\, a_1\, a_2\, c_1$$

$$\Updownarrow$$

$$q_2 = \pm \cos^{-1}\left(\frac{X^2 + Y^2 - a_1^2 - a_2^2}{2\, a_1\, a_2}\right) \tag{B.0.9}$$

It is necessary to define if the positive or negative q_2 is used, this is within the field of robotics referred to if the robot if left- or right arm oriented. Likewise it is necessary to check if $|cos(q_2)| \leq 1$, this ensures the global coordinate is within the robot available work space, where if $|\cos(q_2)| = 1$ the robot is in a singular configuration.

Expression for q_1

The expression for q_1 is derived by firstly rewriting the expressions for X and Y.

$$\begin{aligned} c_{12} &= c_1\,c_2 - s_1\,s_2 \\ s_{12} &= c_1\,s_2 + s_1\,c_2 \\ X &= a_2(c_1\,c_2 - s_1\,s_2) + a_1\,c_1 \qquad \text{(B.0.10)} \\ Y &= a_2(c_1\,s_2 + s_1\,c_2) + a_1\,s_1 \qquad \text{(B.0.11)} \end{aligned}$$

The desire is to bring these expression onto the form:

$$K_1\,\cos(\Theta) + K_2\,\sin(\Theta) + K_3 = 0 \tag{B.0.12}$$

$$K_1\,\sin(\Theta) - K_2\cos(\Theta) + K_4 = 0 \tag{B.0.13}$$

Thereby the equations become

$$0 = \overbrace{(a_2\,c_2 + a_1)}^{K_1}\,c_1 + \overbrace{(-a_2\,s_2)}^{K_2}\,s_1 + \overbrace{(-X)}^{K_3} \tag{B.0.14}$$

$$0 = (a_2\,c_2 + a_1)s_1 - (-a_2\,s_2)c_1 + \overbrace{(-Y)}^{K_4} \tag{B.0.15}$$

From this form, there exist the algebraic solution for the q_1 variable:

$$\tan(q_1) = \frac{-K_1\,K_4 - K_2\,K_3}{K_2\,K_4 - K_1\,K_3} \tag{B.0.16}$$

The function atan2 is used to compute the value of q_1 since this is more numeric robust.

Expression for q_3 and q_4

The expression for q_3 is done by isolating the variable from the expression for Z, likewise q_4 is determined from the expression for ϕ

$$q_3 = Z - d_1 - d_4 \tag{B.0.17}$$

$$q_4 = \phi - q_1 - q_2 \tag{B.0.18}$$

Appendix A

Visual Servoing System

A.1 Main Loop

```
% This visual servoing algorithm uses the toolbox developed by Peter
    Corke
% whic is avaible at his webpage. Likewise the algorithm requires a
    version
% of Adept ACE running on the computer at IP: 172.16.141.110 and port
    1234
% Likewise it is necessary to update the camera definition if not used
% on a Laptop with a USB camera attach or said camera does not support
% resolution of 320x180 at 10 FPS
clear all
clc
close all
imaqreset
%%
load('cameraParams.mat')
load('param_x.mat');
load('param_y.mat');
addpath('./functions/')
addpath('trs/matlab/')
startup_robot
clc

%% Geometri
d1 = 387; %mm
a1 = 325; %mm
a2 = 600-a1; %mm
a4 = 0;
d4 = 0; %mm
alpha4 = 0;
constants = [a1 a2 d1 d4 a4 alpha4];
%% 1D motion model

```

```
 delta_t = 0.1;


F = [1 delta_t 0 0 0 0;
     0 1 0 0 0 0;
     0 0 1 delta_t 0 0;
     0 0 0 1 0 0;
     0 0 0 0 1 delta_t;
     0 0 0 0 0 1];

H = [1 0 0 0 0 0;
     0 0 1 0 0 0;
     0 0 0 0 1 0];

%% Kalman weighting matrices
Vd = [4 0 0 ;
      0 4 0 ;
      0 0 4];

Vn = [1 0 0 0 0 0;
      0 3 0 0 0 0;
      0 0 1 0 0 0;
      0 0 0 3 0 0;
      0 0 0 0 1 0;
      0 0 0 0 0 3];

%% Kalman - Initial

P_k_mark = eye(6);
x_k_mark = zeros(6,1);
x_k_mark(1) = -280;
x_k_mark(3) = 315;
%% Zero position for the robot

 global_zero = [ 600   0   300   0.000   180.000    180];

%% Camera

 vid = videoinput('winvideo', 2, 'MJPG_320x180');
 src = getselectedsource(vid);

 vid.FramesPerTrigger = 1;
 vid.TriggerRepeat = Inf;
 src.FrameRate = '10.0000';
 src.FocusMode = 'manual';
 src.Focus = 0;
 vid.LoggingMode = 'memory';



%% Communication port
```

```

ip='172.16.141.110';
port=1234;

%create socket and open connection
socket=tcpip(ip,port);


%% Control

err_old_y = 0;
int_err_old_y = 0;
err_old_x = 0;
int_err_old_x = 0;

%%
master = 10;
offset = 200; %mm
tracking = false;
finised = false;
detect_count = 1;

%% Vision
posori = [];
count = 1;
start(vid)

%%
SE = strel('rectangle',[3 3]);
roi = [55 320;60 120];
%% Begin loop
fopen(socket);

disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
disp('%                                                              %')
disp('%                  Place housing on conveyor                   %')
disp('%                                                              %')
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')


while true
tic
if mod(toc,delta_t) == 0
        %% Vision System
        [im, time] = getdata(vid,1,'double'); % get image
        time_stamp(count) = time;
        [detect, uc, vc, theta, g] = vision_system(im,roi,SE,
            cameraParams,@iroi,@colorspace,@iblobs);
        uc = uc+roi(1,1);
        vc = vc+roi(2,1);
        posori(count,:) = [uc vc theta];
```

```matlab


        %% Transformation
        p_im = [1 uc vc];
        p_rob = [p_im*param_x p_im*param_y];


        %% Prediction
        z = [p_rob theta]';
        [x_k_1, P_k_1] = kalman_filter(detect,z,F,H,Vn,Vd,x_k_mark,
            P_k_mark);
        if detect == true
                detect_count = detect_count+1;
        end
        x_k_1_save(count,:) = x_k_1';
        T = [x_k_1(1) x_k_1(3) offset x_k_1(5)];
        [q, check] = algebraic_inverse(T,constants);
        %% Check if housing is inside available workspace
        if check == 1
                q_vis(count,:) = q;
        end
        %% Tracking loop
        if check == 1 && detect_count > 40
                %% Get robot position
                clear message
                if master == 10 % Cartesian coordinates
                        message = sprintf('%d,%d,%d,%d,%d,%d,%d,%d',
                            master);
                        fwrite(socket,message);
                        world = fscanf(socket);
                        p = str2num(world(6:end));
                        q_rob(count,:) = algebraic_inverse(p(2:end),
                            constants);
                        p_save(count,:) = p';
                        clear message
                elseif master == 20 % Joint coordinates
                        message = sprintf('%d,%d,%d,%d,%d,%d,%d,%d',
                            master);
                        fwrite(socket,message);
                        joints = fscanf(socket);
                        q_rob(count,:) = str2num(joints(6:end));
                        [p J] = kinematric_transform(q_rob(count,2:end)
                            ,constants);
                        p_save(count,:) = p';
                        clear message
                end

                %% Control
                    %% Y controller
                    err_y(count) = x_k_1(3)-p(3);
                    int_err_y(count) = int_err_old_y+(delta_t/2)*(err_y(
            count)+err_old_y);
        dot_y = 0.8*err_y(count)+ 0.3*int_err_y(count);
        y = x_k_1(3)+x_k_1(4)*delta_t;

        %% X controller
        err_x(count) = (x_k_1(1)+offset)-p(2);
        int_err_x(count) = int_err_old_x+(delta_t/2)*(err_x(
            count)+err_old_x);
        dot_x = (0.8*err_x(count)+ 0.3*int_err_x(count));
        x =((x_k_1(1)+offset)+x_k_1(2)*delta_t)+0.11*err_x(
            count)+int_err_x(count)*(-0.012);

        %% Orientation controller
        err_theta(count) = x_k_1(5)-p(end);
        theta_ref = x_k_1(5)+x_k_1(6)*delta_t;

        err(count,:) = [err_x(count) err_y(count) err_theta(
            count)]
        %% Place lid
        if abs(norm(err(count-5:count))/5) < norm([1 1 1])
            && tracking == true && finised == false
                z_target = 178;
                message = sprintf('%d,%d,%d,%d,%d,%d,%d,%d'
                    ,...
                        666,x,y,z_target,global_zero(4),
                            global_zero(5),theta_ref,22/
                            delta_t);

                fwrite(socket,message);
                pause(delta_t)
                finised = true;
                tracking = false;
                offset = 200;
                z_target = 200;
                message = sprintf('%d,%d,%d,%d,%d,%d,%d,%d'
                    ,...
                        666,x,y,z_target,global_zero(4),
                            global_zero(5),theta_ref,1100);

                fwrite(socket,message);
                pause(delta_t)
        end
        if abs(norm(err(count-15:count))/15) < norm([1.5 1.5
             1]) && finised == false
                offset = 0;
                tracking = true;
        end
    %% Saving

        dot_p_ref(count) = abs(dot_x)+abs(dot_y);
        xy_save(count,:) = [x dot_x y dot_y];
```

```


                %% Send message
                clear message
                message = sprintf('%d,%d,%d,%d,%d,%d,%d,%d',...
                        200,x,y,z_target,global_zero(4),global_zero(5),
                           theta_ref,dot_p_ref(count));

                fwrite(socket,message);


                %% Control parameters
                int_err_old_y = int_err_y(count);
                err_old_y = err_y(count);
                int_err_old_x = int_err_x(count);
                err_old_x = err_x(count);


        end

        %% Update parameters
        x_k_mark = x_k_1;
        P_k_mark = P_k_1;

        count= count+1;
end
end

stop(vid)
fclose(socket);
```

A.2 Functions

A.2.1 Vision System

```
function [detect, uc, vc, theta,g ] = vision_system( im,roi,SE,
   cameraParams,iroi,colorspace,iblobs)
    im = undistortImage(im, cameraParams);
    im = iroi(im,roi);

    %% RGB to HSI
      HSI = colorspace('RGB->HSI',im);

    %% Treshholding

    g = (HSI(:,:,1)>=180 & HSI(:,:,1) <=230 & HSI(:,:,2) >= 0.4 & HSI
        (:,:,2) <= 1  & HSI(:,:,3) < 0.6);
    g = imdilate(g,SE);
    g = imdilate(g,SE);
    g = imerode(g,SE);
    g = imerode(g,SE);
    %% blob detection

    f = iblobs(g,'area',[250 1000]);
%
    [tmp lg] = size(f);

    %% Calculation shit
    if lg >= 1
    detect = true;
    uc = f(1).uc;
    vc = f(1).vc;
    theta = f(1).theta_ * 180/pi;

    else
        detect = false;
        uc = 0;
        vc = 0;
        theta = 0;
    end


end
```

A.2.2 Kalman Filter

```
function [x_k_1, P_k_1] = kalman_filter(detect,z,F,H,Vn,Vd,x_k_mark,
    P_k_mark)
   %% Pure prediction
   if detect == 0
           x_k_1 = F*x_k_mark;

           P_k_1 = F*P_k_mark*F'+Vn;
   else
   %% Correct
   Kk = P_k_mark*H' / (H*P_k_mark*H'+Vd);

   x_k=x_k_mark+Kk*(z-H*x_k_mark);
   P_k = (eye(6)-Kk*H)*P_k_mark;

   %% Predict
   x_k_1 = F*x_k;%+G*u;

   P_k_1 = F*P_k*F'+Vn;
   end

end
```

A.2.3 Inverse Kinematics

```
function [ q, check ] = algebraic_inverse( T,constants )

a1 = constants(1);
a2 = constants(2);
d1 = constants(3);
d4 = constants(4);

X = T(1);
Y = T(2);
Z = T(3);
phi = T(4);

%% q2 calculation
c2 = (X^2+Y^2-a1^2-a2^2)/(2*a1*a2);
if abs(c2) >= 1
    check = 0;
    q = 0;
else
    check = 1;
    q(2) = (acos(c2))*180/pi;

    %% q1 calculation

    K1 = (a2*cosd(q(2))+a1);
    K2 = -a2*sind(q(2));
    K3 = -X;
    K4 = -Y;

    q(1) = (atan2(-K1*K4-K2*K3,K2*K4-K1*K3))*180/pi;

    %% q3
    q(3) = d1-Z;

    %% q4
    q(4) = phi-q(1)-q(2);
end

end
```

Appendix B

Cobra s600 Datasheet

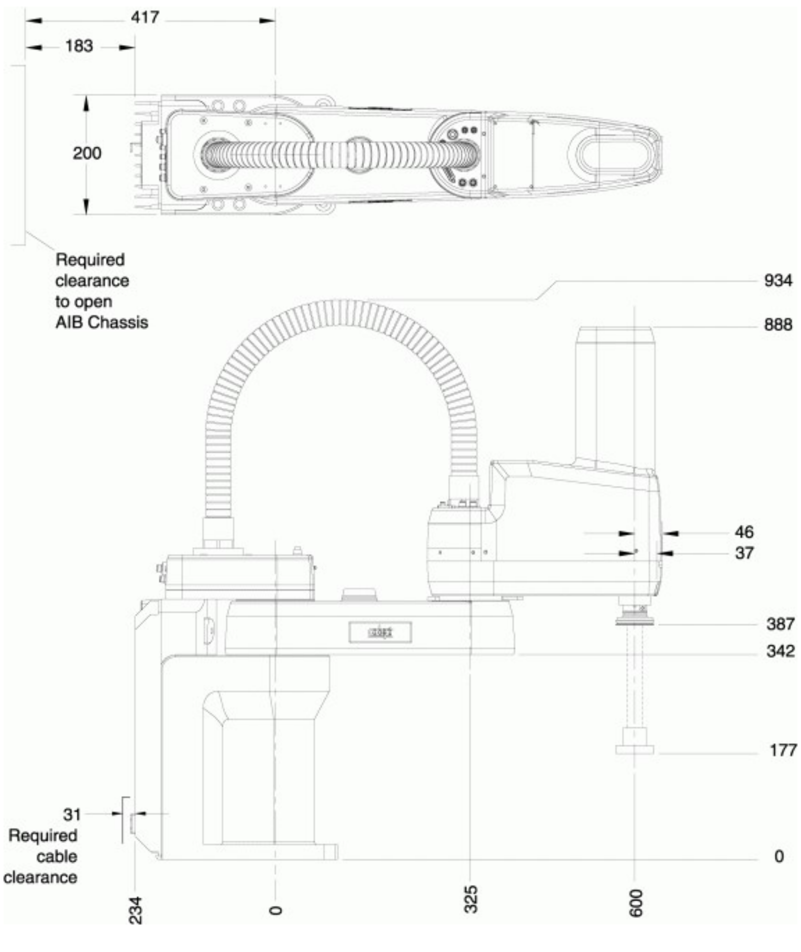


Figure B.1: Datasheet for the Cobra s600 (Adept, 2017)