Learning player abilities based on multimodal data

- MED10 -

Project Report

mta191044

Aalborg University
Media Technology

Media Technology
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Learning player abilities based onmulti-modal data

Theme:
AI, Interaction, Games

Project Period:
Spring Semester 2019

Project Group:
mta191044

Participant(s):
Jorge Villa Yagüe

Supervisor(s):
Matthias Rehm

Copies: 0

Page Numbers: 49

Date of Completion:
September 30, 2019

Abstract:

This project aimed to collaborate with Neurocenter Østerskoven in the development of an Artificial Intelligence for shared control with the members of the institution and their visitors. The aim of the AI was to support the members of Neurocenter Østerskoven on the completion of a task during a multiplayer game, while hindering the performance of their rivals in order to find a common ground in the difficulty and make the experience more engaging for all. The AI proved itself to help when hindering, but the helping part will still need further testing. Further work will be needed in the development and testing of the software, but this project can serve as a starting point for others to come.

Contents

Preface

The following work was developed during a semester as part of a master thesis at Aalborg University, Aalborg, Denmark. I hope that its contents help to clarify some possible solutions to the problem at hand, and that the notes and explanations are clear enough to help those who will come after to progress in the development of this project.

Aalborg University, September 30, 2019

Jorge Villa Yagüe
<jvilla17@student.aau.dk>

Chapter 1

Introduction

Some semesters before the start of this project, the members of Neurocenter Østerskoven received a game made by students from Aalborg University. This game had the purpose of reinforcing learning for people with cognitive and physical disabilities by including gamification of concepts such as pattern recognition and name - image association. The game was based on a radio control car system with two robots (Turtlebot-2 model) which acted as the cars, a joystick with a screen attached for controlling each of them, three poles with screens acting as targets and a computer connecting the whole system (See Figure 1.1).

A problem was detected when the users played against their relatives, who didn't have disabilities of any kind. This generated an imbalance in the game, where some players had direct advantage over others because of their conditions. For this reason, the engagement factor of the game was endangered, and with it, the educational part of the game.

For solving this problem we proposed to create an AI that supported one of the users (the one with the disability) and hindered the control of the other, balancing the experience for both groups and improving their engagement, allowing them to focus on the learning aspect of the game.

The solution proposed was a form of shared / hybrid control.

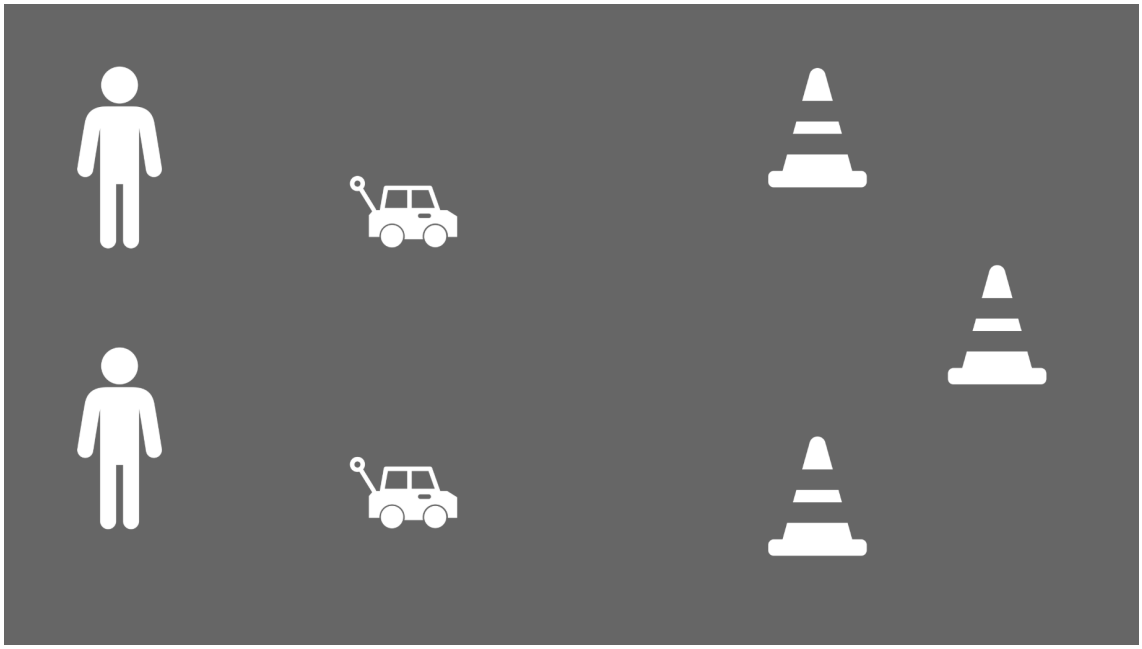

Figure 1.1: A scheme of the game. Both users control an RC car each with a joystick, that they have to drive to one out of three poles.

Chapter 2

Background research

2.0.1 Reference work

The main articles consulted for this project are the H-metaphor written by NASA in 2003 [12] and the posterior study of its application in 2011 by Damböck et al. [10], and the sources that inspired both articles to their conclusions.

2.0.2 Artificial Intelligence

What is an AI?

"AI", stands for "Artificial Intelligence". As defined by Poole et al, AI research is the study of "intelligent agents", which are considered "any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals" [21]. Russel et al. [30] prefer the terminology "rational agent" for it, but the definition stays quite similar, by stating "any device that perceives its environment and takes actions that maximize its chance of success at some goal".

Machine Learning

We call Machine Learning to the subset of artificial intelligence where the algorithms build mathematical models based on sample data (known as "training data) in order to make predictions or decisions without being explicitly programmed to perform the task [16].

The result of running a machine learning algorithm can be expressed as a function y(x) which takes a new digit image x as input and generates an output vector y, encoded in the same way as the target vectors. The precise form of the function y(x) is determined during the training phase, also known as the learning phase, on the basis of the training data. Once the model is trained it can determine the identity of new digit images, which comprise a test set (see Figure 2.1). The ability to categorize correctly new examples that differ from those used for training is known

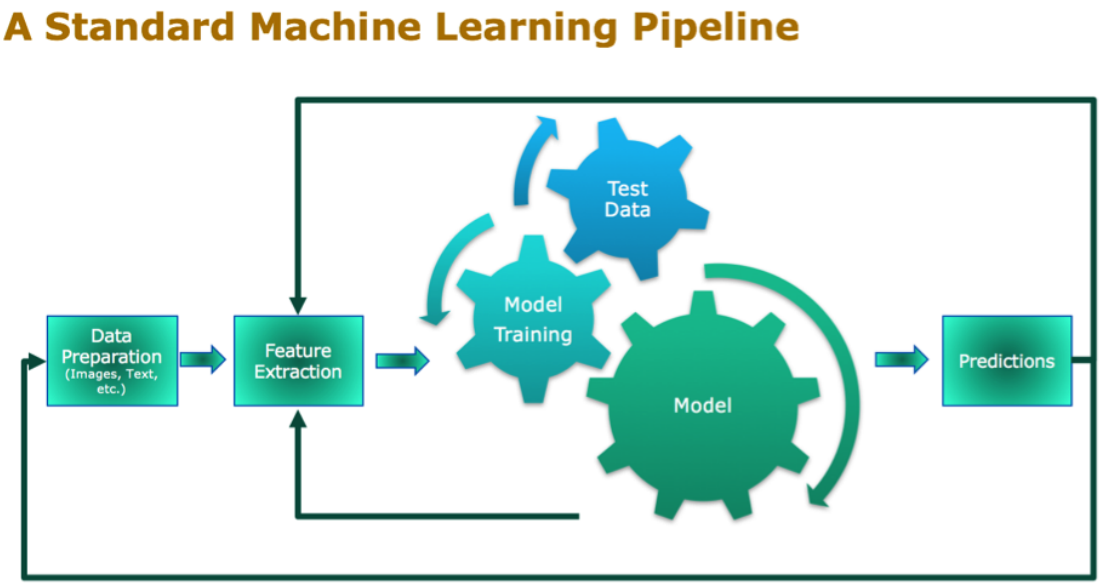

Figure 2.1: Machine learning algorithms workflow.
https://2s7gjr373w3x22jf92z99mgm5w-wpengine.netdna-ssl.com/wp-content/uploads/2018/09/WD_3.png

as generalization. In practical applications, the variability of the input vectors will be such that the training data can comprise only a tiny fraction of all possible input vectors, and so generalization is a central goal in pattern recognition[7].

2.0.3 What is Shared / Hybrid control?

An advanced vehicle driver assistance system (ADAS) is a vehicle control system that aims to improve driving comfort and traffic safety by utilizing various environment perceptive sensors, such as lasers, cameras, and radars, to monitor driving surroundings and then assist the driver in recognizing and reacting to potentially dangerous traffic situations [35] [9] [31]. An ADAS aims to achieve better control effects of vehicles through the combination of assistance control systems and drivers, sharing the control authorities of manoeuvring with the human driver, rather than taking over the driver's authorities [17]. An ADAS can be seen as a two player game [34], with the controller, who can only issue controllable transitions, on one side, and the environment, that can choose the trajectory of the variables and can take uncontrollable transition whenever they are enabled, on the other [6]. A central element of shared control is the allocation of control authority, i.e. the perception of which one of the two elements of the shared control is the one in charge of the driving in a certain moment[4].However, estimating the authority intention in shared control is not a trivial task, due to the vagueness and the lack

of knowledge of how it shapes the driver's behaviour [11].

From all the theories about shared control, the one this project was the most influenced by was "The H-Metaphor as a Guideline for Vehicle Automation and Interaction" [12], in paper published by NASA on December 2003.

2.0.4 Why hybrid control instead of automated systems?

As stated by Bainbridge [2], fully automated systems can create what is defined in his work as "the irony of automation". By this, Bainbridge meant that "by taking away the easy parts of a task, automation can make the difficult parts more difficult". Therefore, a hybrid control system should be focused on the hardest parts of the process rather than the easy ones, allowing the user to have authority during the interaction but facilitating the general manoeuvring of the vehicle.

More complex systems require more complex communication solutions, as explained by Norman [20]: "The solution will require higher levels of automation, some forms of intelligence in the controls, and appreciation for the proper form of human communication". Due to our system focus on its learning possibilities through gamification, it would make sense to make the driving, which is the gamification part of the project, as simple as possible. This would allow the users to focus on the learning part of the experience. A hybrid control system is suitable for this task since it does not eliminate the users' driving authority completely but assists them in the process. While the hindering AI complicates the driving of the vehicle.

2.0.5 Control in hybrid vehicles

It has been observed that experienced operators are able to reduce the required effort of driving by developing precognitive [18] or skill-based routines [23]. However, their sensory and processing resources are still limited [36]. Strategic tasks such as monitoring the remaining fuel on the display have to be kept to a minimum in order not to break the actual control loop [3]. In our problem the users are required to monitor the information displayed oh the screens of their controllers, and matching it with the correct screen in one of the three poles, making them deviate the attention from the actual control of the vehicle. Because the goal of this project is to keep the learning part of the game present, the vehicle control must be assisted in order to allow the users to turn away the attention from it when needed. On top of that, due to the users' only occasional usage of the driving system, precognitive and skill-based routines are difficulty to acquire, making the learning period for driving the vehicles a necessity, and therefore allowing the learning part to be focused on the educational part of it.

However, the hybrid control system should still allow the users to feel authority over the driving. Flight Management Systems (FMS) are commonly used for

the control of diverse tasks related to plane traffic, from flying complete routes to automated landings [12]. However, other automated systems can intervene in the FMS input, appearing unpredictable to the operator and subsequently prone to cause "human error" [37] [32]. If the role of the pilot becomes one of supervising and monitoring the automation without direct involvement, the automation leaves her/him ill-prepared to both recognize an issue and to intervene [33], taking the pilot out of the control loop. Therefore, when coming to our problem formulation, the solution should help the users to control the vehicle and reach the pole, but always on a simple, communicative way. The assistance system should be individual, in order to prevent interference between automation systems as in FMS [12], and allow the users to feel driving authority during the whole experience.

2.0.6 A simile through the horse metaphor

The horse metaphor is a widely spread theory used to explain the inner workings of hybrid control systems. As in the original work published by NASA [12], we will begin by explaining what a metaphor is. Norman [19] gives a detailed description of what a "system image" is, a concept that is needed to fully understand the Horse metaphor:

"The user's model is the mental model developed through interaction with the system. The system image results from the physical structure that has been built (including documentation, instructions, and labels). The designer expects the user's model to be identical to the design model. But the designer doesn't talk directly with the user – all communication takes place through the system image. If the system image does not make the design model clear and consistent, then the user will end up with the wrong mental model."

Metaphors, on the other hand, are meant to transfer meaning from one thing (the source) to another thing (target) [12]. Thus, the horse metaphor is meant to create a proper system image of how a hybrid vehicle works. The authors of the original NASA work describe it as follows:, "If you were riding a horse, you would be able to read your map and be confident that you would not hit any trees or run into people because horses instinctively avoid obstacles. And, using physical feedback through the seat of your pants and your reins, you are constantly aware of what your horse is doing, even while focusing your attention elsewhere. If the horse is unsure about where to go, it may slow down, and seek a new obstacle free path while trying to get the rider back into the loop. The horse might also be aware of how engaged you are and adjust its behavior. If a dangerous situation suddenly pops up, it will try to react before it is too late. You can let your horse choose its path without being completely out-of-the-loop or you can take it on tight reign to reassert a more direct command." "Now apply this image to a new kind of vehicle. Imagine that you could drive or fly through an environment with obstacles and other vehicles, and would be able to focus on other tasks like navigation,

communication, or even enjoying the scenery. You could be confident that your vehicle would not hit anything because it senses and avoids obstacles. Through the physical feedback from your haptic interface, an active joystick for example, you are constantly aware of what your vehicle is doing. If your vehicle senses any danger or is unsure about where to go, it will assume a more cautious and stable configuration, and you can feel where the vehicle is trying to lead you. The vehicle might also be aware of how engaged you are and will adjust its behavior. An extreme example would be if the operator is incapacitated and the vehicle maneuvers to a safe state. If some sudden danger pops up, it will react before it is too late."

Comparing the Tight Rein and Loose Rein modes of controlling horses [5] hybrid systems could operate on a mode continuum, in which the human and the machine would contribute equally to the control of the driving, but in which control would also be delegated mutually between both depending on the situation (tight reins for human control, and loose reins for vehicle control). Therefore, in this project a Tight Reins control should be used whenever the users want to decide to which pole to drive their vehicle (the strategic decision) and the Loose Reins when approaching it so that the vehicle helps the user to reach the pole.

2.0.7 Feedback systems

Damböck et al. [10] explain that on shared control systems both the driver and the automation are simultaneously involved in the driving task, acting parallel to each other (Figure 2.2). Both perceive the environment separately, generate an intention based on their perception and try to put that intention into practice by affecting the vehicle, the driver or accordingly the automation [10]. This communication and negotiation is carried out via the manual haptic channel using active control elements, but can also be supplemented by the presence of acoustic and visual information.

Damböck et al. mention three different kinds of scenarios for a side stick-controlled vehicle in their example:

- In the first scenario, the side stick doesn't support any kind of feedback system (Fig2.3). The operator creates forces on the stick, which adjusts the set point settings of the vehicle. Thus, the dynamics of the stick are autonomous and don't allow conclusions about the state of the vehicle. Therefore the user has no knowledge about its actual state.

- In the second scenario there are position reflective control elements (Fig 2.4), in which, as opposed to the force reflective system (that uses a spring-centered stick), the position point for the vehicle results from the balance of forces in the stick. "The feedback information is returned as position of

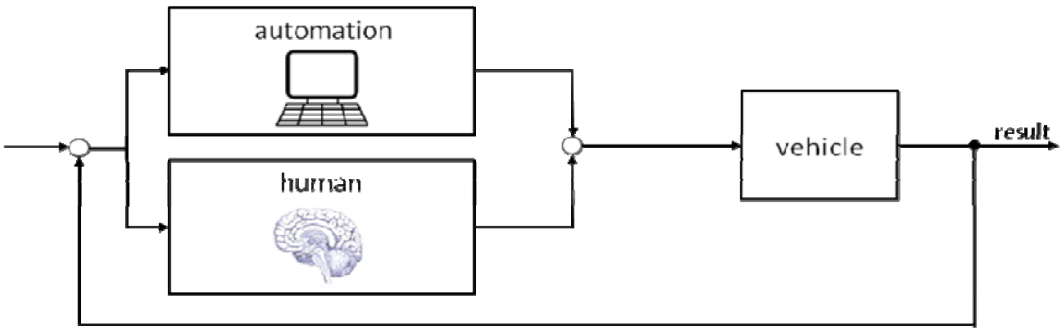

Figure 2.2: Schematic of the communication between driver and automation system. The summation of the parallel work of both is transmitted to the vehicle.
{Dambock2011}

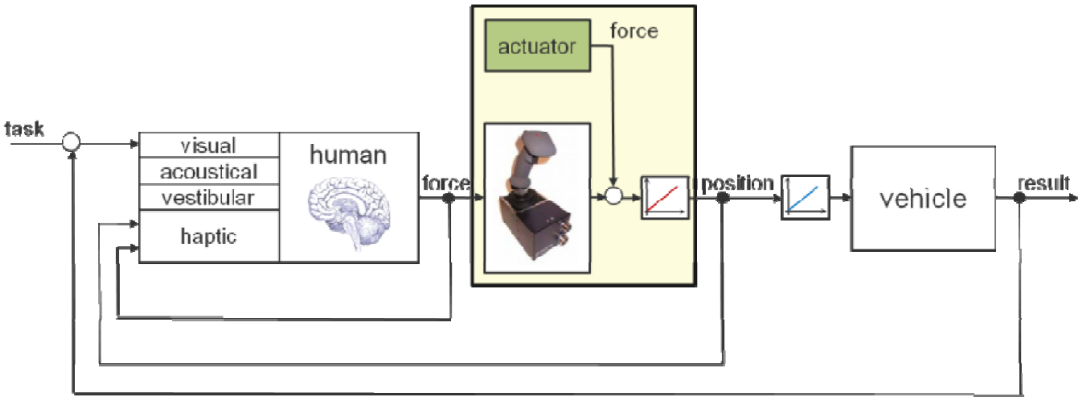

Figure 2.3: A force reflective control element.
{Dambock2011}

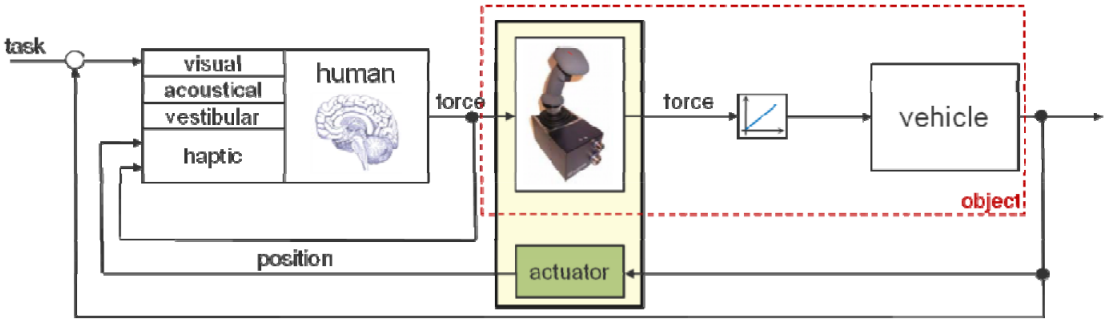

Figure 2.4: A position reflective control element.
{Dambock2011}

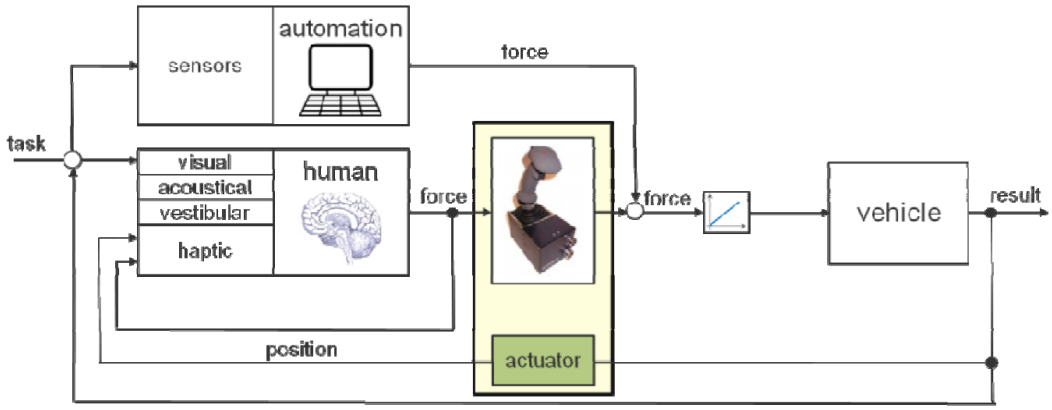

Figure 2.5: A position reflective control element.
{Dambock2011}

the element which thereby represents the actual state while its movement represents the dynamic of the system itself". Thus, the operator senses the behaviour of the system/vehicle.

- The third scenario presents the ideal situation for hybrid control systems. In this, both driver and automation system have influence on a position reflective control element. That way the driver has direct feedback of any action taken by the automation system over the vehicle (By the force exerted by the stick on any direction) and can have direct influence on the control of the vehicle by exerting force on the stick **??**.

2.0.8 Conclussion

From the insights that were made in hybrid / shared control the following points can be seen as given for our scenario:

The automation would be minimal, trying to create a hybrid control, not an automated one. This will prevent the driver from being "out of the loop" [2]. This is also supported by the educational aspect of the game, in which a high amount of automation would suppose a detriment on the challenge, and thus a lost in the educational value. To follow Norman [20],a system as simple as possible will be designed to minimize the amount of feedback that has to be given to the users, as the final users may not pay that much attention to visual or sound feedback.

The control will be done by a force reflective control element (figure 2.3), which is the one already implemented in the physical solution at the start of this project. Ideally, in future steps of the project the control could be adapted to an assisted position reflective control element (figure 2.5).

Due to the nature of this project, in which an AI is designed to hinder one group of users and to help the other group, a simile with the horse metaphor can be made[12], having two different kind of horses: An obedient horse (the helper) and a wild horse (the hinder). Therefore, the control for those users with the helper AI should obey them and assist them in the driving of the vehicle, while the control for the users with the hinder should feel disobedient and opposed to the user intention.

2.0.9 Problem formulation

With the knowledge the background research and by transferring the information to the case this thesis, the problem of the project was reformulated as follows:

What are the challenges of implementing a hybrid control system to an educational RC car game, in which two different AIs are needed that work at the same time, one helping one user, and another hindering the other, while maintaining the educational factor for both of them?

Chapter 3

Design

3.0.1 The game

The game is based on a radio control car system (RC car system from now on) with two robots working as vehicles and three poles working as the possible targets. Each of the robots is controlled by a player, and each player uses a controller for doing so. The controller consists is a force reflective 8 axis joystick with a screen attached to it.

The poles also count with screens attached to them. The objective of each player is to drive their robot to the right pole, which displays the same image on its screen as screen than the one displayed on the on the player's controller screen. The first player who hits the right pole scores a point, and the player with the most points after some rounds (defined by the facilitator) wins the game. Hitting a pole that is not the right one doesn't have any negative effects on the score.

All the elements in the game (the two joysticks, the three poles and the two robots) are connected by a central computer that coordinates all the operations and communications. The game runs on an instance of ROS (see section 3.0.3) supported in Python code (see section 3.0.4)

3.0.2 The robots

The main robots used for the project are TurtleBots Burger. TurtleBot is a low-cost, personal robot kit with open-source software. These TurtleBots were used as the RC cars in the game. The TurtleBots of this project counted with a laser 360 degree sensor, and two small engines for facilitating its movement, one on each wheel. The system was operated by a Raspberry PI attached to it.

The poles where independent robots themselves. They counted with a Raspberry PI inside to coordinate their system, a small display on top to show the goal image that the users had to aim for, and collision sensors on the bottom that were triggered when hitting one of the robots.

The robots were navigated by controller each, made with an 8 axis joystick, a Raspberry PI and a display. The joystick was a switch rather than a continuous system, meaning that the inputs transmitted by it varied from 0 to 1, rather than a progressive value, therefore having a set speed. The display on the joystick was used to show the user to what point the robot had to be driven to, and the Raspberry PI coordinated both systems.

All the robots were coordinated by a central terminal, a stationary computer, that acted as server and coordinator of all the IPs and the game. The terminal kept the score of each player as well as initiated and finished the game. The server sent a signal to both the joystick and the target pole . The joystick received the IP of the target pole, and the pole received the IP of the player's vehicle. On collision, the poles detected the IP of the robot. If the IP of the robot collided against the IP of the target pole, the player scored a point.

3.0.3 ROS

What is ROS?

"ROS" stands for "Robot Operating System", an open-source programming framework developed in the late 2000's at Standford University. As explained by Brian Gerkey [13] ROS can be described in the following equation:

$$ROS = Plumbing + Tools + Capabilities + Ecosystem \tag{3.1}$$

Where each of the variables refer to:

- Plumbing: ROS provides publish-subscribe messaging infrastructure designed to support the quick and easy construction of distributed computing systems.
- Tools: ROS provides an extensive set of tools for configuring, starting, introspecting, debugging, visualizing, logging, testing, and stopping distributed computing systems.
- Capabilities: ROS provides a broad collection of libraries that implement useful robot functionality, with a focus on mobility, manipulation, and perception.
- Ecosystem: ROS is supported and improved by a large community, with a strong focus on integration and documentation. www.ros.org is a one-stop-shop for finding and learning about the thousands of ROS packages that are available from developers around the world.

Gerkey further explain: "In the early days, the plumbing, tools, and capabilities were tightly coupled, which has both advantages and disadvantages. On the one hand, by making strong assumptions about how a particular component will be

used, developers are able to quickly and easily build and test complex integrated systems. On the other hand, users are given an "all or nothing" choice: to use an interesting ROS component, you pretty much had to jump in to using all of ROS."

The main advantage that ROS provides for the robotic industry is the standardization of the "nodes" or groups of functions, which can be shared via internet and used for commercial purposes due to its open software nature [24] [26]

Topics, Services and Actions

The communication in ROS is mainly done through three different kind of publishers: Topics, Services and Actions[1]:

-**Topics:** In Topics a publisher runs a series of actions on the background for an indefinite period of time. Robots can call and stop the call to topics at any given point, using the information provided by them for different purposes. An example for a Topic would be the publisher that tracks the position of one of the robot cars, which sends the information twice per second.

-**Services:** Services are synchronous publishers. Once a robot call a Service it runs the action of the Service until the service itself stops, not running anything else in the meantime. Services are mainly used for blocking every action of the robot until the current action is completed, for example, requesting for user input to start the game and maintaining all the robots on hold until then.

-**Actions:** Actions are asynchronous calls to Services. Actions can be called at any point by a robot, and other actions can be taken during the normal execution of the first one. An example for this would be the system that moves the robots in real time while the sensor is still active.

Programming in ROS

Independently of the coding language used for the development of the apps in ROS, all the systems created on the platform work in the same structure: By initializing, running and stopping different nodes[1].

A node, as defined in ROS wiki [29], is "an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes (See Figure 3.1). Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service." Because the language that was used for programming the original game was Python. For this project we used the corresponding library for Python implementation in ROS, "rospy". [28]

The code for both functionalities of the project (the helper and the hinder) was developed as different executable files, and thus, as independent nodes.

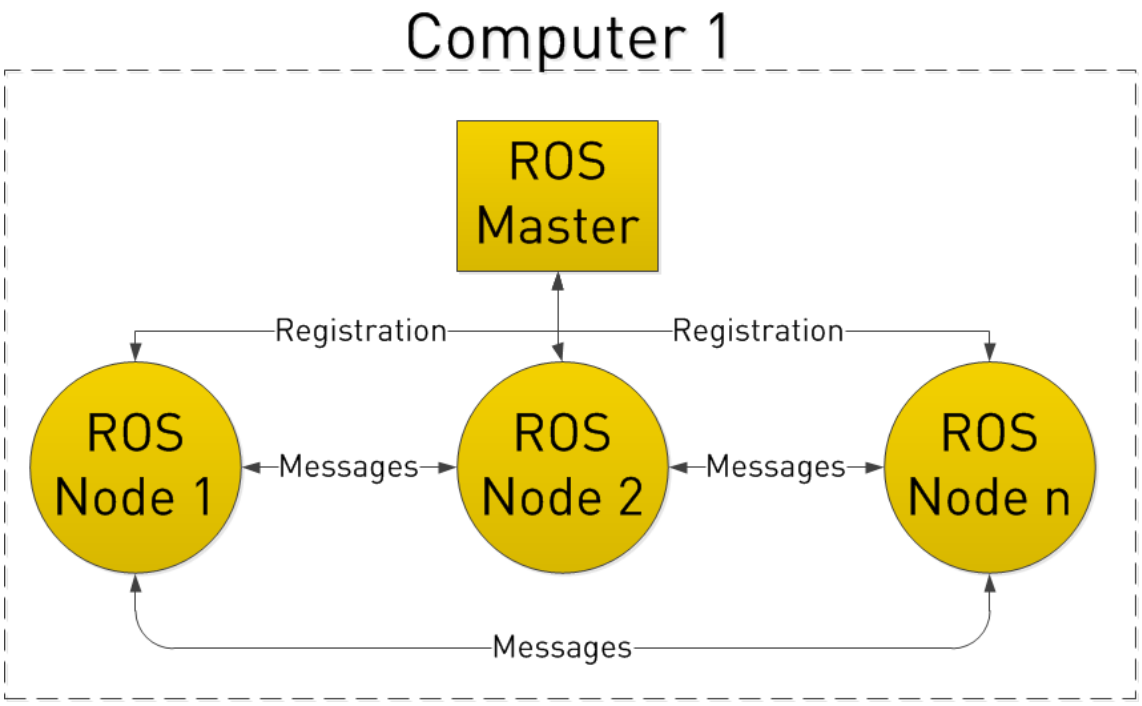

Figure 3.1: Basic node communication structure in ROS.

https://www.clearpathrobotics.com/assets/guides/ros/_images/ros101one.png

ROS Development Studio

One of the main disadvantages of ROS is that it requires to be run on an Ubuntu machine, forcing developers to either use a computer with Ubuntu integrated as the operative system (OS) or through virtual machines. During this project "ROS Development Studio", one of the online solutions for ROS simulation, was used.

As stated on their webpage [8] ROS Development Studio (ROSDS from now on) is an online platform for both learning and development in ROS. Instead of having to install Ubuntu on a computer or run a virtual machine version of the OS, ROSDS allows the users to run an online instance of Ubuntu and ROS for developing directly in the web browser. The only limitation comes with the installation of specific libraries for python, which were limited due to the online nature of the instantiate of ROS.

Robot Ignite Academy

The online learning materials provided by ROSDS is called Robot Ignite Academy. Its content was used during the project for learning the basics of ROS, especially of Topics, Services and Actions, the ROS navigation stack, and the use of ROS debugging tools.

3.0.4 Python

What is Python?

On Python's official website it is stated site[22]: "Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, help to connect existing components together. Python's (...) reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed." Using Python, an open source coding language, combined with ROS, an open source programming framework, allowed this project to be used without commercial licenses.

Python integration in ROS

The Python integration in ROS is handled by the rospy library [28]. The library favours implementation speed over runtime performance, allowing to quickly prototype simple algorithms. ROS also offers the "roscpp" library to use C++ for the development, which is slower to prototype but has a better performance [27].

3.0.5 Initial proposal

For the initial proposal an AI with two main functionalities was defined: Discerning where the users were trying to go and helping them to go there. If the AI was applied to non-disabled users, it should also be able to discern where they were trying to go and make it harder for them to drive the vehicle there. The following proposals focused in one of the three functionalities of this AI (detecting the players intention, helping players and hindering players):

Detecting the player's intention - proposal 1: Creating a virtual grid

This option was based in a machine learning approach, combined with reinforcement learning. If the field are the robots were in play could be divided in a grid of some kind, the behaviour of the player should be traceable by a machine learning algorithm after observing the players behaviours enough times. Therefore, an AI should be able to be implemented with a training set large enough to allow it to approximately predict where the users were trying to go just by the inputs of their controllers (See Figure 3.2). After this the AI would be trained using a Reinforcement learning approach, where it would be rewarded when reaching the target with the minimal number of steps, or punished when exceeding them.

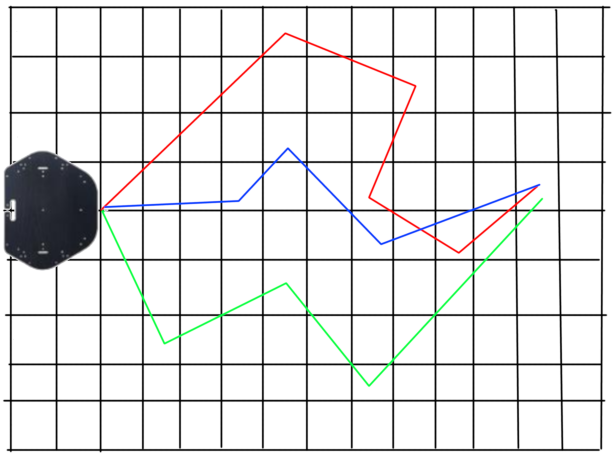

Figure 3.2: A brief explanation of how the machine learning algorithm would work. The training set would include different routes taken by the players with the robot to reach the objective. Those routes (In red and green) would be taken as a training set to develop a model which could predict new routes once the player started playing (in blue) in order to "guide" the robot through it.

However, this option would have generated a "driver out of the loop" kind of situation [3], and thus generate "human error" [37] [32]. Apart from that, this

option could have failed due to the overspecialization of the AI, meaning the assistance system would only be able to help the users in rooms similar to the ones where the training set information was gathered. Furthermore, the original game is of the nature o an educational project, therefore being focused on teaching the users by their interaction with the system. By creating a system with too much automation and no feedback the educational element would have been taken out of it.

Detecting player intention -proposal 2: Using computer vision to detect the trajectory

In the second option a computer vision system proposed, in which the poles an the robots are recognized by a camera. The robot's trajectory would be estimated by the algorithm and modified to face the poles (See Figure 3.3).

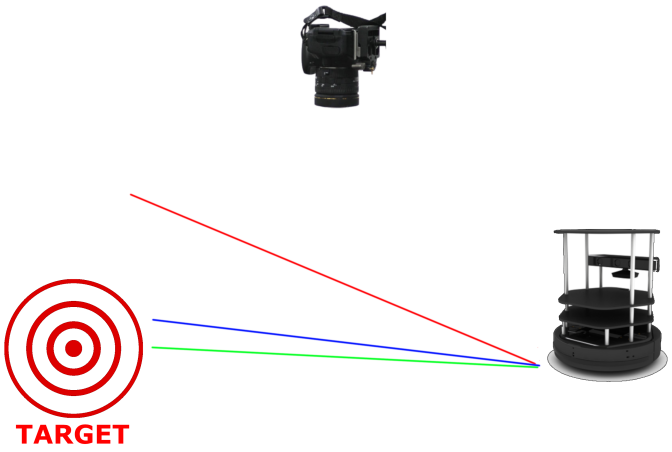

Figure 3.3: The camera system would have an overview of the playground where the robots would move around. The actual trajectory of the robot is represented in red, while the optimal one to the target is represented in green. When the AI is activated, it would modify the route of the TurtleBot to follow the blue trajectory, a mix of both of them.

This option, however, included a series of problems: The use of a computer vision algorithm would mean a new field of study for the project that was not tackled during the background research. Furthermore the presence of the camera would require a set-up of some kind for the functionality of the AI, necessitating the presence of a facilitator during game sessions who had knowledge about the structure. On top of that, if the camera had to be able to have control over the field, a support structure would have had to be developed, restricting where the camera could be used due to the size of the installation, and reducing the application

possibilities of the system. Therefore, the idea was discarded.

Detecting player intention -proposal 3: Detecting the correct pole

In the third solution, the following reasoning was made: Assuming that the educational part of the game was understood, and therefore, that the users were driving their robots to the right pole, an algorithm could be designed that help the users move towards the target (See Figure 3.4.

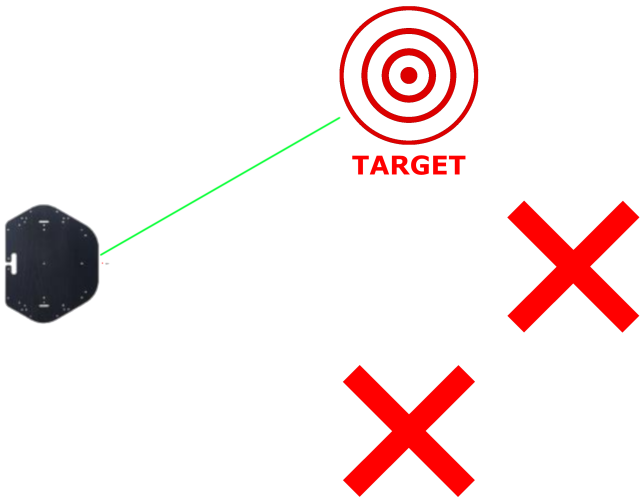

Figure 3.4: A simple scheme of the behaviour of the algorithm. Independently of the trajectory of the robot, the AI would always help the user move towards the pole

However, this option had the same problem as the first proposal: If the users were helped to go towards the right pole, the educational factor of the game would have been completely eliminated.

Detecting player intention - proposal 4: Detecting the closest pole

Therefore, instead of detecting the target pole and helping the users go there, it was decided to take an intermediate step: When the users vehicles entered an area around a pole, the AI would help them reach it, whether the pole was the correct one or not, but always giving margin to the users to change the trajectory at will (See Figure 3.5).

- **Area of detection:** All the poles have an area of detection around them where the AI started operating. As stated before the influence of the AI must not be stronger than the influence of the controller, thus allowing the users to turn the robot around and face another pole. In area intersections between poles the AI should only help the users go towards the closest one.

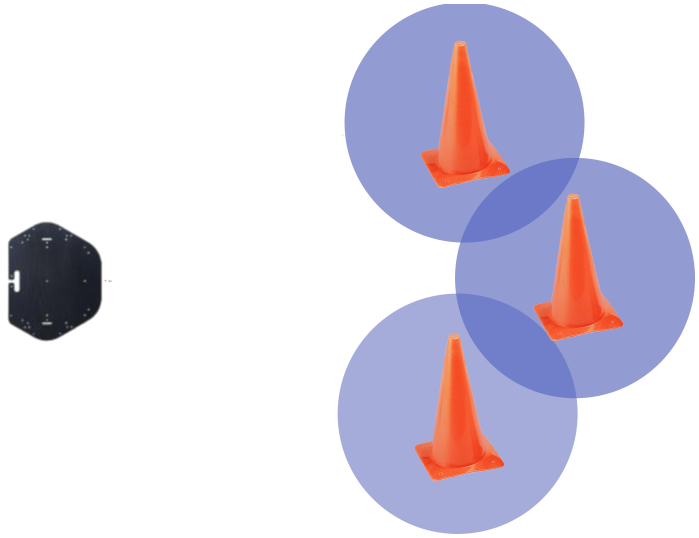

Figure 3.5: Each pole has around an area set by a common threshold. Once the robot enters one of the areas the AI starts working, guiding the robot towards the pole. However, the work of the AI needs to also allow the robot to escape the area of influence of a pole. Only one pole is chosen at a time by the AI, so there are no conflicts in the area in between poles.

- **AI free area:** As a possible option it was proposed to also have a small area free of AI around the poles, making it more challenging for the users once their vehicles were close enough to them. Although it was an interesting option, it was decided to keep it as a possibility instead of implementing it because of not being interesting enough for the problem, building the code while having this in mind to easily update it in the future.

3.0.6 Helping the players to reach the point

In ROS< there are many different was to reach a point. The system includes different tools for the control of robots, from automated navigation via ROS Navigation Stack to modifications in the trajectory of the robot with the cmd_vel topic.

ROSNavigationStack

ROS includes a module (Stack) for automated navigation called ROS Navigation [25]. There are numerous elements that need to work at the same time for ROS Navigation to operate:

- **Mapping of the environment around the robot:** This can be done via the sensors attached to the robot or to an external device that sends information to the machine. In order to activate the Navigation, stack a robot needs mapping of its surroundings.

- **Localization of the robot inside the environment:** Once the mapping has been done the robot can proceed to try to locate itself inside of it. The location of the robot can be modified manually through input of its coordinates towards a point of reference, but normally it is the robot that locates itself by detecting derivations on the information acquired via its different sensors or by comparing its own map to the other robot's map.
- **Path planning:** Once the robot has finished mapping and provides information about its own location in its environment it can begin the path planning. The points to which the robot has to travel can be imputed manually or sent by a script located in an external source (or the robot itself, depending on its purpose). If indicated, once received a point the robot starts doing the path planning to avoid obstacles and also calculates the best route through its map towards the target.

The robots of this project only had a laser sensor on top of them. Through it, the robot registered an array with numbers representing distances, knowing which object was further and which one was closer. Through enough recognition around the playground the robot could have been able to recognize the position of the poles on it, but differentiating between poles and other objects was not doable through such a sensor. Therefore, we opted to use another solution.

ROS cmd_vel

ROS includes a Topic called cmd_vel, where the information about the linear and angular velocity of the robot is published. With this Topic the robot can have a velocity applied to it, in order to make it rotate towards a target, or move towards it. Thus, by applying the adequate speeds on the robot, both linear and angular, it can be ordered to face a certain objective, in our case, the closest pole.

The idea was to send an angular velocity to the robot so it faced the closest pole at any given time (Always only when inside its area of influence). This speed would only be implemented when the robot was stationary, allowing players to rotate the robot and face a new goal at will. Since the target group of this game was not re-active to feedback it was decided to keep it to a minimum, showing the robot intention only by its movement.

3.0.7 Hindering non-disabled player's control of the robot

The same way the AI could help the robots to face the pole by cmd_vel, it could hinder them by applying random angular velocities on activation. To assure that this didn't happen at all times, we decided to design the hinder AI to trigger only when the "forward" inputs from the 8 direction stick were received, allowing the

users to redirection the robot when stationary, but still making it harder to control when advancing.

Chapter 4

Development

The scripts used in ROSDS for activating the functions of the robot were separated in two: Helper.py and Hinder.py. The first one was in charge of the main functions of the AI (detecting the proximity of the pole, rotating the robot, etc.) while the second one had to hinder the actions of the users once triggered. Both scripts were separated in individual elements to be able to trigger each of the functionalities on and off separately. This allowed us to test the behaviour of participants in three conditions: With the hinder, with the helper, and without any of them.

4.0.1 Helper.py

What is a quaternion?

Quaternions were first described by William Rowan Hamilton in 1843. According to his definition, a quaternion is the quotient of two directed lines in a three-dimensional space, or equivalently, the quotient of two vectors [15][14]

A quaternion is generally represented in the form:

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \tag{4.1}$$

Where:

- *a,b,c and d* are real numbers .
- ***i,j and k*** are imaginary numbers which represent the fundamental quaternion units.
  - **i** rotates in the wx and yz planes
  - **j** rotates in the wy and zx planes
  - **k** rotates in the wz and xy planes

Quaternions were used in this project because of their ability to represent three-dimensional rotations. On a quaternion, the three components with imaginary numbers represent the vector around which the 3D body rotates (being **i**, **j** and **k** representatives of the three coordinates that define the distance of the point from the origin). The scalar *a* represents the summation of the sinus and the cosine of the angle that the body is rotating.

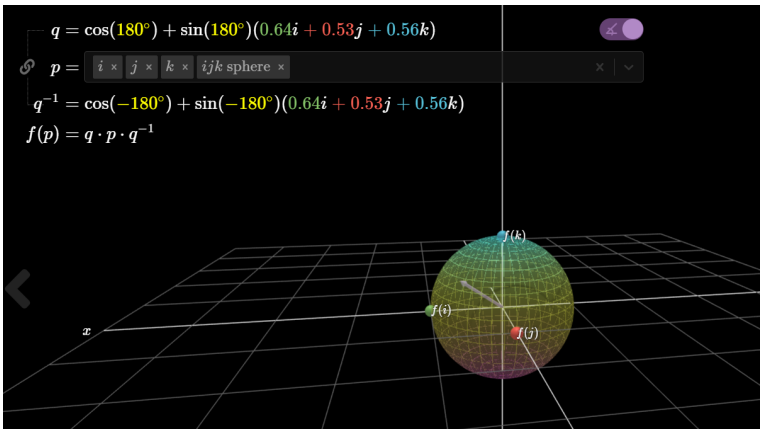

Figure 4.1: The image represents a sphere which rotates around a vector. The grey arrow is the vector that is defined by **i**, **j** and **k**, and around which the sphere rotates. The summation of the sinus and cosine at the beginning of the equation is the same as the scalar *a*

https://2s7gjr373w3x22jf92z99mgm5w-wpengine.netdna-ssl.com/wp-content/uploads/2018/09/WD_3.png

On applied quaternions the following formula comes in play:

$$f(p) = q.p.q^{-}1 \tag{4.2}$$

Quaternions are "double covered" when it comes to rotations in 3D, which means that any given rotation belongs to two separate points on opposite sides of a hypersphere in four dimensions that we can't see. The first quaternion multiplies the sphere p from the left, and the second (Which is the inverse of the first) multiplies the sphere from the right.

How is the distance between two points calculated?

The distance between two points P(x1,y1) and Q(x2,y2) results from the following formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} < R \tag{4.3}$$

The proof for this comes as following: Let's define d as the distance between the point A(x1,y1) and the point B(x2,y2). L equals to (x1,0), M to (x2,0) and C(x2,y1)

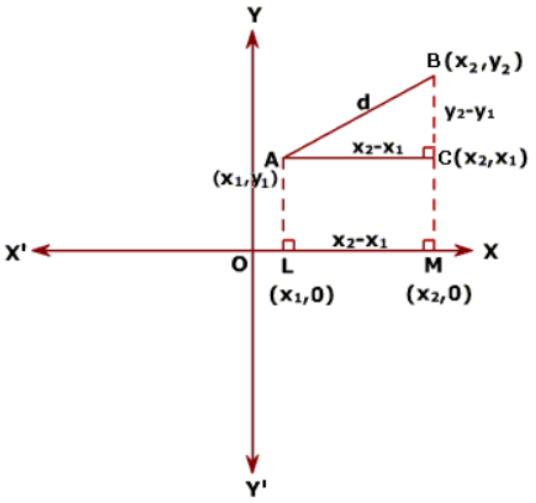

Figure 4.2: Above this lines the graphic with the poins A(x1,y1), B(x2,y2), L(x1,0), M(x2,0) and C(x2,y1).
https://math.tutorvista.com/geometry/distance-formula.html

(This last one forming the triangle ABC with A and B). With the origin in O(0,0), if OL = x1 and OM = x2, it can be said that the length of AC equals the one of LM, and thus:

$$AC = AL = OM - OL = x_2 - x_1 \tag{4.4}$$

If MB = y2 and MC = LA = y1, then:

$$CB = MB - MC = y_2 y_1 \tag{4.5}$$

Then, with Pythagoras theorem:

$$a^2 + b^2 = c^2 \tag{4.6}$$

We can apply to the right-angled triangle ABC:

$$AB^2 = AC^2 + CB^2 d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2 d = sqrt((x_2 - x_1)^2 + (y_2 - y_1)^2) \tag{4.7}$$

What is a quadrant?

Any of the four areas in which a plane is divided by an x and y axis, as shown in Figure 4.3, are called a quadrant.

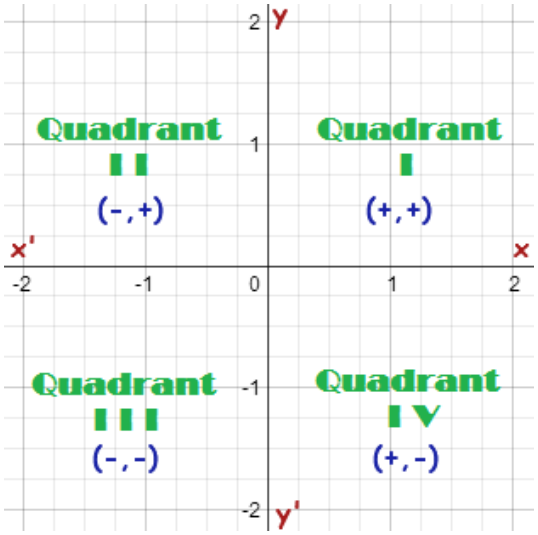

Figure 4.3: In each one of the quadrants the values of x and y change. Therefore, in the first quadrant both values are positive (+,+), for the second x is negative and y is positive (-,+), for the third both are negative (-,-) and for the fourth x is positive and y is negative (+,-).

https://d2jmvrsizmvf4x.cloudfront.net/1qkPaeoESVuB1YQo3ezM_coordinate-system.png

Intersection between lines

Although the robots exist in a 3D space, the intersection between the orientation vector of the robot and any line having the pole as origin will happen in the x and y axis, therefore, the intersections of vectors are only thought in a 2D space. Different solutions for detecting said intersections were developed.

Option 1: "Long enough vector" intersection system

The first proposal was to calculate the intersection between the orientation vector coming out of the robot and an imaginary line traced diagonally from the pole to one out of the four quadrants of the circle around the pole.

After some time of implementing this option the behaviour in ROSDS showed several instabilities. The intersection point was not always detected (i.e. when the orientation vector and the imaginary lines were parallel). On top of that, the intersection algorithm forced to create an imaginary line between the origin of the robot and a far point on its orientation to create "long enough vectors" to allow intersection, creating another margin for error. In the end, the solution proofed itself to be too complicated for what it was tried to be achieved. This is why the calculations were revised resulting in a much more simple solution.

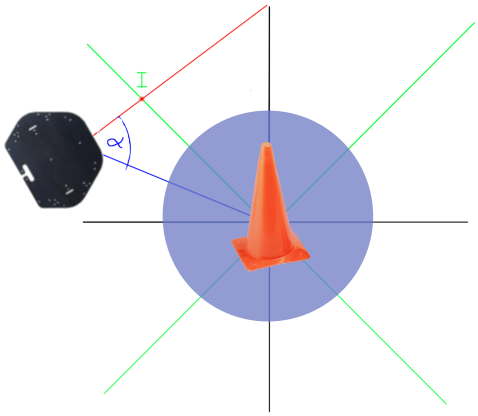

Figure 4.4: The red line represents the trajectory vector of the robot. The green lines are imaginary segments made between two points: The pole(x1,y1) and a distant position (i.e. (x1-400. y1+400) in order to make sure that there is an intersection for the quadrant were the robot is. The angle alpha is calculated through trigonometry, and shows the rotation that the robot need to take in order to face the pole

Option 2: Quadrant system

Knowing the coordinates of the robot on its position point R(xr,yr) and the coordinates of the pole on its position C(xp,yp), it was also known in which quadrant around the pole the robot was by comparing the coordinates of both, as explained in Figure 4.3, such as in:

$$if(xr > xp, yr > yp) : quadrant = 1$$
$$if(xr < xp, yr > yp) : quadrant = 2$$
$$if(xr < xp, yr < yp) : quadrant = 3$$
$$if(xr > xp, yr < yp) : quadrant = 4$$

Therefore, the robot's position to the triangle position could always be connected by a right-angled triangle, with the hypotenuse connecting the robot and the pole, and a pivot point between both with the same x coordinate as the robot and the same y coordinate as the pole (see Figure 4.5). This method could be used because the cmd_vel message "Twist()" rotates the robot in a certain angle, from the origin orientation of the robot if the yaw is subtracted. Therefore, knowing the target angle TA the robot could be rotated at any given point to face towards the pole.

After some testing of this code it was discovered that the orientation for "yaw = 0" in the robot corresponded to a parallel line with the x axis instead of a parallel line with the y axis, changing the system to the one seen in Figure 4.6

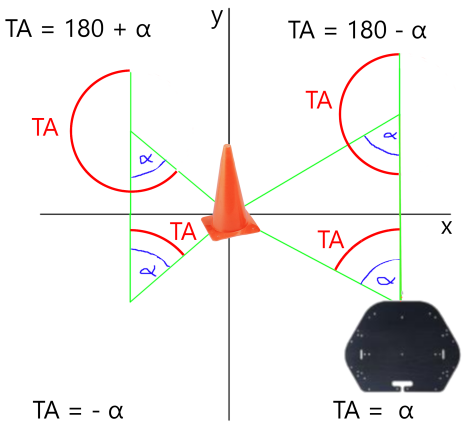

Figure 4.5: On the image the Target Angle (TA) varies depending on the quadrant were the robot stands respecting to the triangle. Therefore, on the 1st quadrant TA equals to 180 degrees minus alpha, in the second one, to 180 degrees + alpha, in the third one to -alpha and in the fourth one to alpha, being alpha the angle that separates the vertical of the robot position to the line formed between the pole position and the robot position. The angle in this model is only dependant on the position of the robot in the quadrant, and not on its orientation.

4.0.2 The Helper algorithm

The algorithm used by the helper can be explained by a flowchart (Figure 4.7). After the start of the script, the positions of the pole and the robot are detected. Then the AI checks if the robot is inside the trigger area of one of the poles. If not, it applies a 5Hz delay and goes back to the beginning of the script, updating the position of the robot checking again.

If the robot is inside the trigger area the AI will calculate the quadrant around the pole in which the robot is. Once that is done, the AI will obtain the target angle, and rotate the robot to face the pole. After these actions have been taken the AI checks if the robot has reached the pole. If not, a 5Hz delay is applied and the position of the robot around the pole is calculated again. If the robot has reached the pole, the script ends.

4.0.3 The Hinder algorithm

The Hinder.py script was created for generating problems in the control. Its basic behaviour was achieved through the use of a random rotation applied to the robot when moving it forwards, by the use of the 'random' Python library and the 'cmd_vel' topic with a Twist() message. The algorithm in Figure 4.8 depicts the working of the Hinder algorithm.

Right after the beginning of the script a 5Hz delay is implemented to give a small advantage to the player aided by the Helper script. The AI then checks if

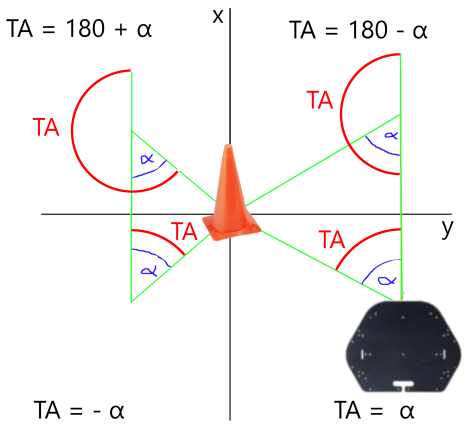

Figure 4.6: The same model as shown in 4.5, but with the x and y fixed to correspond to the real orientation given by the yaw in the /Odom topic.

the robot is making any movement in the Y axis (the equivalent of ROS for going frontwards / backwards). If the answer is negative, another delay is applied, and the check is performed again.

If there is a movement applied to the robot on its Y axis, a random rotation is calculated. After this, the rotation is applied to the robot. The system then checks if the robot has reached the right pole. If the answer is affirmative, the script ends. If negative, a 5Hz delay is applied. After that, the system checks again if the robot is moving on the Y axis, and the script continues.

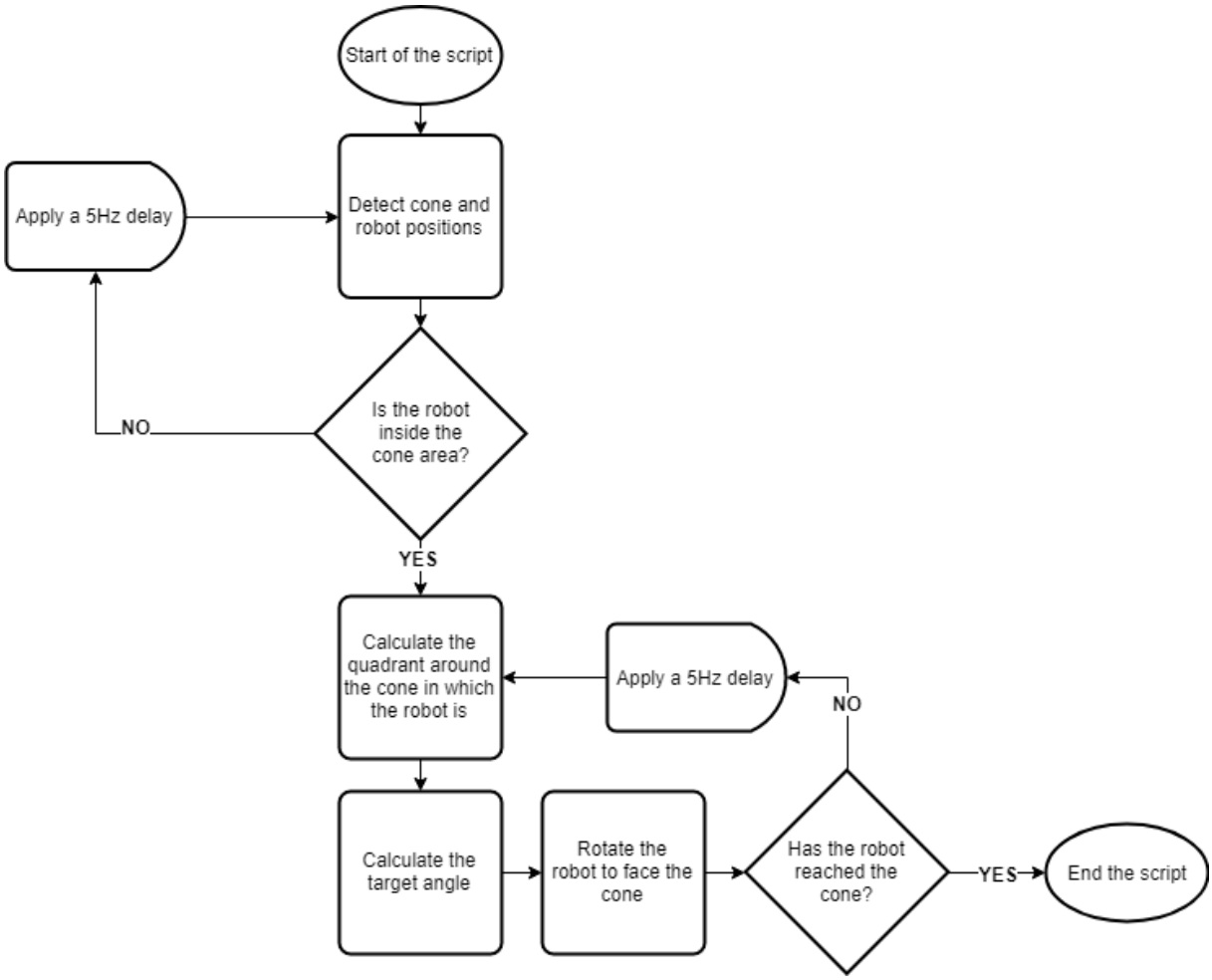

Figure 4.7: Flowchart explaining the helper algorithm

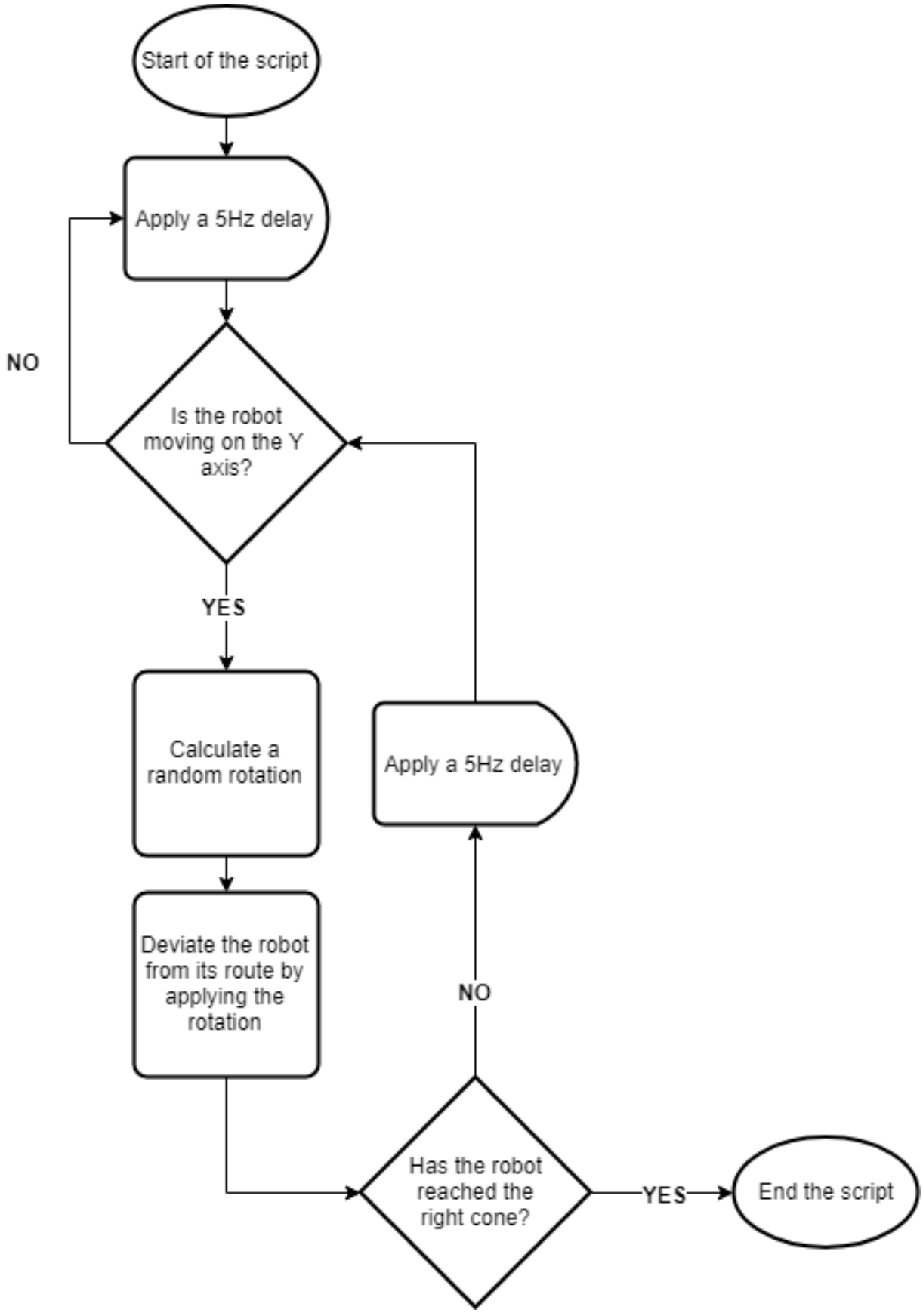

Figure 4.8: A flowchart explaining the working of the Hinder algorithm

Chapter 5

Experiment

5.1 Design

In the experiment the users were asked to complete a simple driving task inside the simulation. This tasks would be performed in three different scenarios: Having the helper active, having the hinder active, and having the AI deactivated. Our dependent variable was the time of completion (in seconds), and our independent variable was the AI mode (1, 2 or 3 depending on which one of the scenarios).

Due to the impossibility of testing with the final users of the RC cars (the members of Neurocenter Østerskoven), it was decided to test on other participants. We couldn't obtain people mentally impaired, so we decided to simulate the impairment by having both Hinder and Helper activated at the same time. This led us to three different, new scenarios: AI deactivated, hinder activated, and hinder and helper activated at the same time.

The experiment was within-subjects since all of the subjects tried all of the three scenarios. To eliminate the systematic effects of behaviour, we decided to randomize the order in which the three scenarios were tested, making sure to have each combination at least three times. This prevented users from learning how to be proficient with the controls during the simulation . Therefore, the six combinations were:

- Normal - Hinder - Hinder + Helper
- Normal - Hinder + Helper - Hinder
- Hinder - Normal - Hinder + Helper
- Hinder - Hinder + Helper - Normal
- Hinder + Helper - Normal - Hinder
- Hinder + Helper - Hinder - Normal

Hypothesis

There should be a difference in the time of completion of the task depending on which one of the three scenarios we are on:

- H1: The hinder condition leads to a decrease in performance compared to the "No AI" condition. Performance is measured in time of completion.
- H2a: The hinder + helper condition leads to an increase in performance compared to the hinder condition.
- H2b: The hinder + helper condition leads to a decrease in performance compared to the "No AI" condition.

Apart from this, it is expected that the users react somehow to the AI control. The sudden moves of the robot in the simulation should be enough to communicate that the AI is also controlling the direction.

5.2 Participants

The participants were 30 university students from Aalborg University, Denmark. All of them were inhabitants of a university dorm.

The participants were equally distributed between females and males. All of them were around 24 years old. They had different nationalities, with participants from Denmark, Spain, Vietnam, China, Netherlands, Kenya, etc. The participants were students of diverse programmes. All of them were computer literates, but only a minority had experience playing computer games or controlling virtual environments.

5.3 Aparatus

For the experiment a laptop running an online simulation in ROS Development Studio was used. For measuring the time a digital stopwatch was used, and for annotating the time and participants information an online spreadsheet was used.

5.4 Procedure

The test was carried out on an isolated room of a university dorm where only the participant and a facilitator were present. Each user was given an explanation of the control scheme at the beginning of the test and answered any questions about the structure of the test or the control of the software. Their completion times were measured by the temporizer and written down by the facilitator in the spreadsheet.

The participants were asked to use only one finger to press the keys, in order to create only one kind of input at a time, emulating the 8-way joystick that the original RC car game had. The simulation consisted of controlling the robot via keyboard controls and making it reach a pole (virtually represented by a cylinder). The pole to reach was pole 1 (as described in the previous chapter, with coordinates [3,3,0]), while the robot started at world origin coordinates, [0,0,0]. The time of completion was registered for each one of the users, and the tasks were done in three different ways:

- **Normal controls:** The robot with the normal controls active, 'i' key for moving forward, 'k' for stopping, 'm' for moving backwards, 'j' for rotating to the left, 'l' for rotating to the right, 'o' for rotating to the right with forward motion at the same time, 'u' for rotating left with forward motion, 'n' to rotate left with backward motion and ',' to rotate right with backward motion. This controls are the ones given by the "teleop" ROS module for TurtleBot.
- **Hinder:** The normal controls, but with the hinder implemented.
- **Hinder and helper:** The version with the hinder and the helper activated.

After completing the full test the facilitator explained in detail what each one of the AIs did.

5.5 Results

The average times of the users gave results different from the expected ones, as seen in Figure 5.1. Despite the average time of completion with the Hinder being higher than the time for completing the task in normal conditions, the time with the Hinder and the Helper was by far worse than just with the Hinder alone.

It was observed during the testing that when both scripts were activated the movement of the robot showed an aberrant behaviour, most likely because of having two different angular velocities being applied to the cmd_vel topic at the same time, one by the Hinder and one by the Helper. This caused the controls to be unresponsive, which generated confusion in the participants because they thought that the controls were defective, having a "driver out of the loop" kind of problem.

On a further analysis through linear regression R^2 showed a value of 0.132, thus showing that the variance generated by the different modes didn't have a significant influence on the general variance of the data (Figure 5.3. Further testing would be needed to discern if the low value of R^2 was due to a lack of significance or a lack of data.

The P-value (0.02) was under 0.05 when running ANOVA, and the F value (3.9) was bigger than our F critical (3), which allows to assume that the null hypothesis

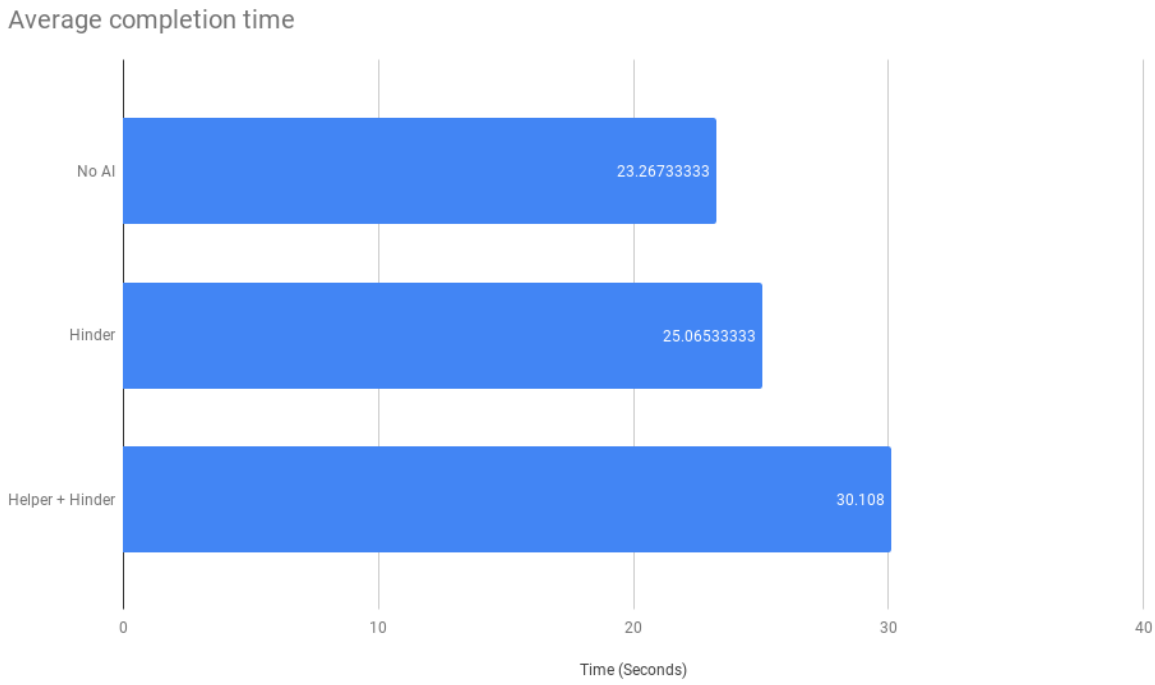

Figure 5.1: The results of the testing with the users. As described in this section, the hinder shows a detriment in the performance of the participants, and the helper together with the hinder show an even greater detriment (Due to the technical complications mentioned in the discussion).

could be rejected (Figure 5.4). Thus, the difference between cases was statistically significant.

After running the ANOVA the data was processed with a Tukey post-hoc test to find out where the difference between the three cases lied (Figure 5.5). As seen in the figure, only the value comparing the Hinder + Helper condition with the No AI condition was higher than the Tukey critical value. Therefore, it can be confirmed that there is a statistically significant difference in the means of both conditions between each other, but more data would be required to make the same assumption about the other two combinations.

5.6 Discussion

More data would be needed to make assumptions about the relation between the three conditions, but a statistically significant difference between them was detected. The results were not overall positive toward the helper script, and its effectiveness was hard to test without the real users. That, added to the problems that the hinder and the helper running at the same time generated, reduced the

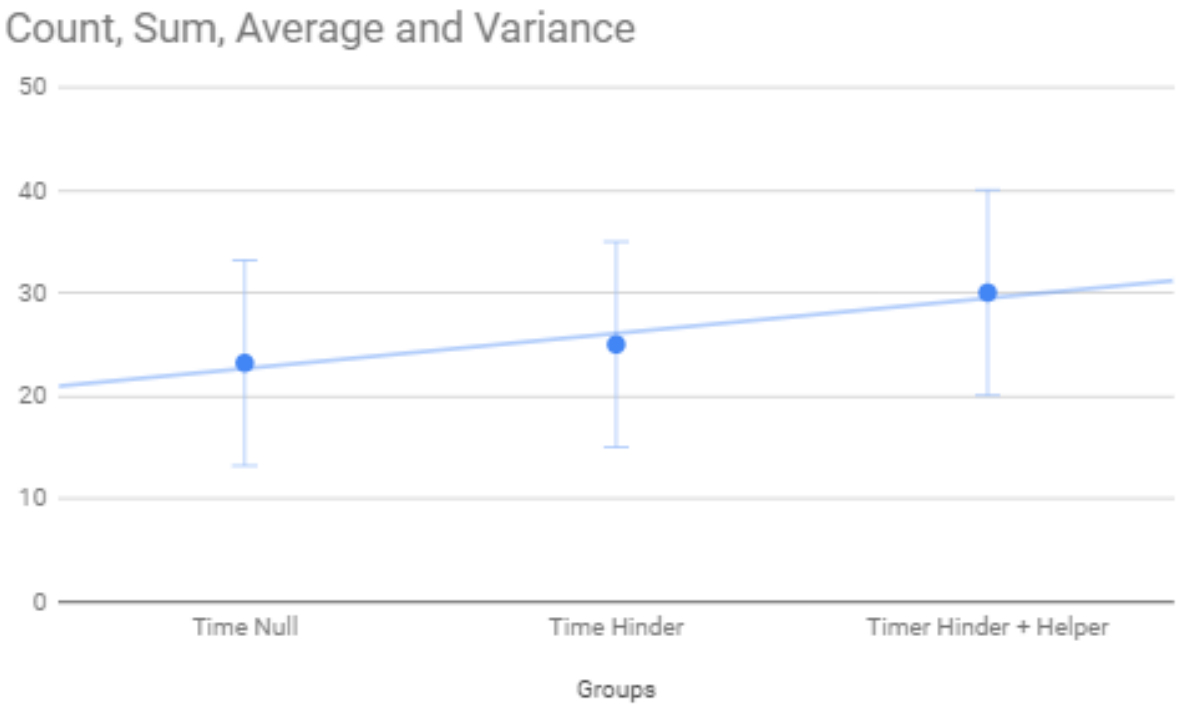

Figure 5.2: Comparison between the group means

testing success. The hinder, however, showed itself to be successful when making it harder for the users to complete the task.

5.6.1 The ideal testing situation

One of the main problems encountered during the testing was the reaction between the helper and hinder scripts. While one of the scripts was trying to help the users face the pole, the other one was trying to make the robot rotate in a random direction. Due to both scripts operating in the cmd_vel topic to do so, in the moments when the robot was receiving input from both scripts at the same time (receiving two angular speeds in opposite directions), its movement stopped. This, added to some kind of interference with the movement of the robot when two keys were pressed at the same time by the participants (which happened sometimes even though the facilitator had told the participants to only use one at a time) made the test conditions not ideal.

However, in the ideal testing situation (In which the members of Neurocenter Østerskoven would be by the helper AI and their relatives would be stopped by the hinder AI) both scripts would never coexist, therefore eliminating the problem of the interference between both of them. On top of that, in an ideal testing situation (once the code is implemented in the already existing physical system) the control

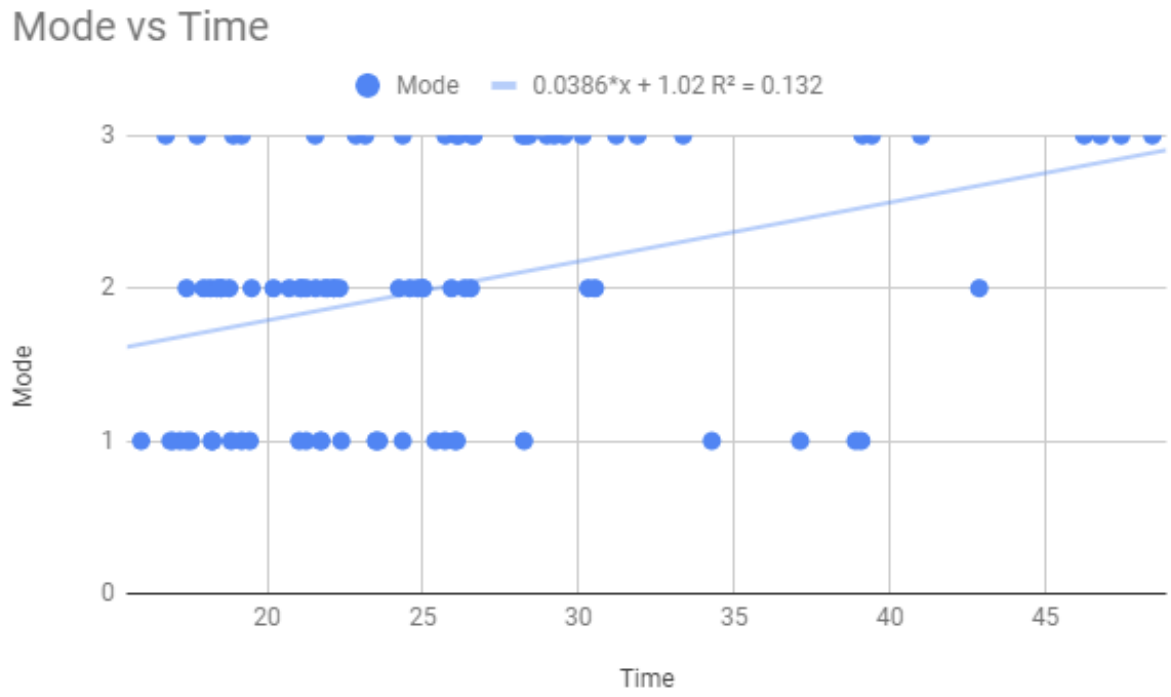

Figure 5.3: The results of the linear regression ran over the data, being 1 the mode without AI, 2 the mode with the Hinder, and 3 the mode with the Hinder+Helper combination.

of the TurtleBots would be done through 8-ways joysticks, whic, in contrast to the keyboard, would prevent having two inputs at the same time.

5.6.2 Implementing a feedback system

One point that generated confusion during the testing was the lack of feedback given by the robot towards the participants when the AI decided to implement a rotation. This affected the performance during the Hinder + Helper scenario. This scenario would not happen with the real users of the game, because of the reasons explained in the previous section, but a physical feedback system could be implemented to reduce confusion. An example of this would be a position reflective control element aided by the AI (Figure 2.5).

This would require a new version of the Helper algorithm, that with a feedback system included would look similar to Figure 5.6. After every time the robot calculates the rotation, a signal would be sent to the joystick that communicates the robot's intention to rotate to the user through its movement. If the users were trying to rotate the robot in another direction, a stronger signal would be sent via the joystick to express the AI intention, but the loop would go back to the initial

ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	*754.5601156*	*2*	*377.2800578*	*3.935074826*	*0.0231180135*	*3.101295757*
Within Groups	8341.230213	87	95.87620935			
Total	9095.790329	89				

Figure 5.4: The ANOVA results

	Hinder + Hinder & Helper	Hinder + Null	Hinder & Helper+ Null
Tukey test	1.421965185	1.005761223	3.826516835
Tukey critical value	3.4	3.4	3.4
Are them significantly different?	NO	NO	YES

Figure 5.5: The Tukey results

calculation. This would ensure that even with the help of the AI the user is free to move the robot in another direction, allowing to keep the educational part of the game.

This implementation would require the design of a new joystick that integrates both the position reflective control element and the screen for the users.

5.6.3 Detecting the poles by their odometry

One of the clear milestones that should come in future development is the detection of the other poles in the system by their odometry. In the current state of the project the poles positions were hardcoded in the script, in other words, they were defined at the beginning, thus not being real, physical positions.

A good first step would be to reach the ability to detect objects in the virtual environment of ROSDS. If the transform of another object could be detected at the beginning (an object existing when the simulation starts), the robot would be able to detect at all points where that object was, just by filling the information about the pole position with the information about that object position.

In a next iteration the robot should be able to detect the object by its position in the map relative to the robot position. This would include the participation of the ROS Navigation Stack, thus including mapping of the surroundings of the robot (done by the laser of the robot, and by extra sensors if needed, like the camera commented on the initial proposals) and understanding where the robot in relation to this positions.

The final iteration would be telling the robot were the poles are by their odom-

etry. Due to all the systems (poles, controllers and robots) being connected to the same server the poles should have an odometry of some kind. If that was the case, it should be fairly easy to fill the information of the pole position with the odometry information about those poles. This would require a full research in multi-robot systems in ROS.

5.6.4 Data gathering with the users for creating a hinder

The hinder proved itself to work during the testing, but its functionality and the degree of hindering it exerted over the controller was selected as a rule of thumb. A proper way of creating a hinder would be to test with the members of Neurocenter Østerskoven until obtaining a proper amount of data, then using that data as a training set for a machine learning algorithm that controlled the robot, and then activating spontaneously that algorithm when the relatives used the robot, mimicking the behaviour of the original users, hindering their use with more similar conditions.

This project could be fairly complex, especially because of the amount of data that would be needed. It would require intense sessions of game play and a proper recording model to guarantee that the behaviour was properly registered and that there was enough information in the data set to train the algorithm.

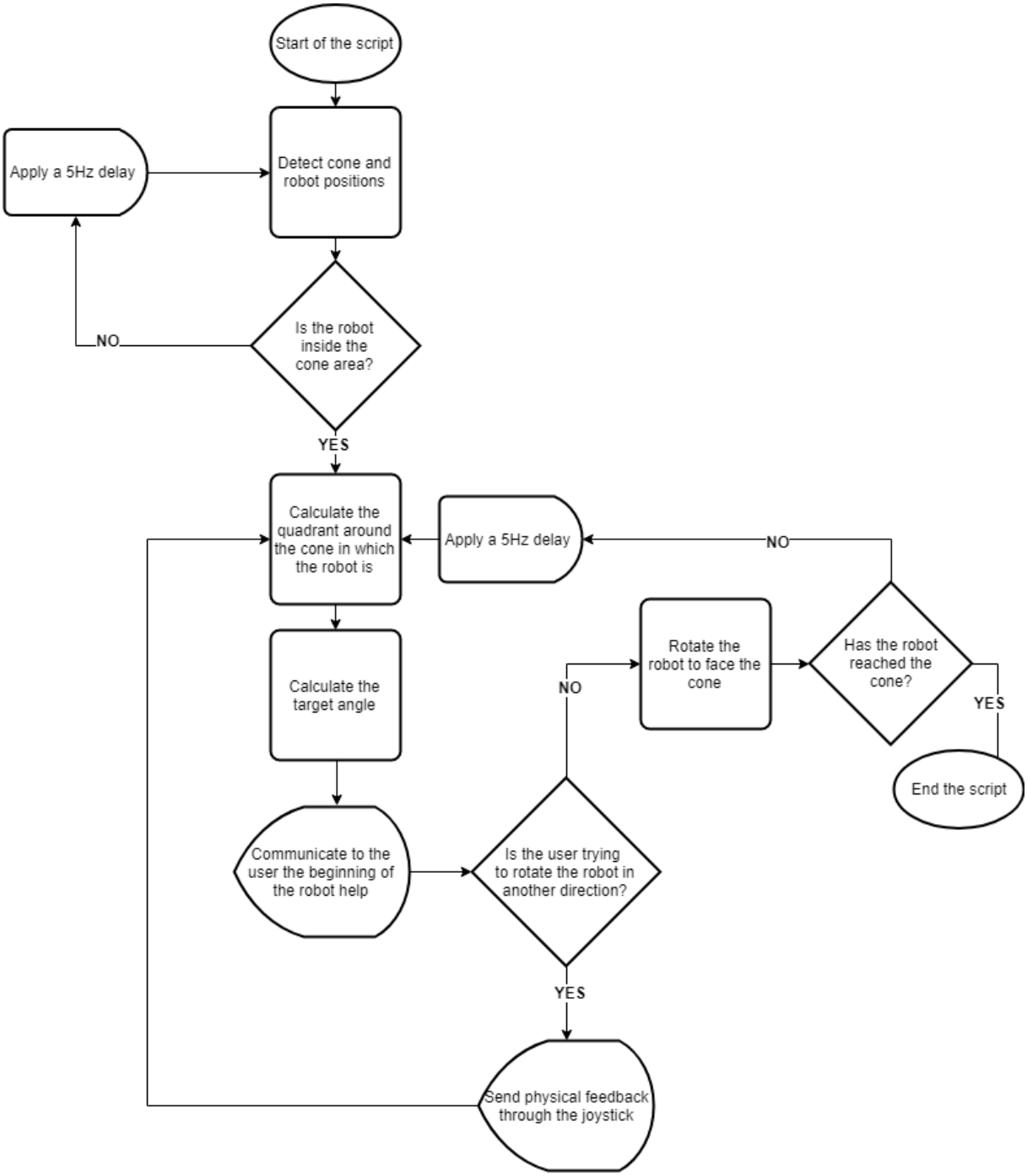

Figure 5.6: Flowchart explaining the helper algorithm with a feedback system integrated

Chapter 6

Conclusions

Further development would be required to obtain a more communicative hybrid control system, including different feedback systems that would have to be tested to select the most efficient mean of communication for the final users. The current solution should be tested on the members of Neurocenter Østerskoven to get a better knowledge of the areas of improvement for the Helper algorithm. At the same time, the data gathered during these testing sessions could be used to improve the Hinder algorithm and knowing how it would replicate the handicap for not-handicapped users. This project, despite not being a final solution, served to set the foundations for future development of a solution, proving that there is an influence of the AI on the driving of the game, and taking a first step towards balancing the game at Neurocenter Østerskoven.

Bibliography

[1] Robot Ignite Academy. *ROS Basics in 5 days*. [Online; accessed 8-May-2019]. URL: https://www.robotigniteacademy.com/en/course/ros-basics-in-5-days_1_0/.

[2] Lisanne Bainbridge. "Ironies of automation". In: *IFAC Proceedings Volumes* 15.6 (1982), pp. 129–135.

[3] Sheldon Baron and William H. Levison. "Display Analysis with the Optimal Control Model of the Human Operator". In: *Human Factors* 19.5 (1977), pp. 437–457. DOI: 10.1177/001872087701900502. eprint: https://doi.org/10.1177/001872087701900502. URL: https://doi.org/10.1177/001872087701900502.

[4] Johannes Beller. "Towards a dynamic balance between humans and automation: authority, ability, responsibility and control in shared and cooperative control situations". In: (Jan. 2012).

[5] C. Belton and Deutsche Reiterliche Vereinigung. *The Principles of Driving*. Complete riding and driving system. Kenilworth, 2002. ISBN: 9781872119458. URL: https://books.google.es/books?id=3i8pAAAACAAJ.

[6] Massimo Benerecetti, Marco Faella, and Stefano Minopoli. "Reachability games for linear hybrid systems". In: *HSCC*. 2012.

[7] Christopher Bishop. *Pattern recognition and machine learning*. New York: Springer, 2006. ISBN: 9780387310732.

[8] The Construct. *ROS Development Studio*. [Online; accessed 8-May-2019]. 2019. URL: http://www.theconstructsim.com/construct-learn-develop-robots-using-ros-2/.

[9] M. Dai et al. "Dynamic Output-feedback robust control for vehicle path tracking considering different human drivers' characteristics". In: *2017 36th Chinese Control Conference (CCC)*. 2017, pp. 9407–9412. DOI: 10.23919/ChiCC.2017.8028857.

[10] Daniel Damböck et al. "The H-Metaphor as an Example for Cooperative Vehicle Driving". In: *Human-Computer Interaction. Towards Mobile and Intelligent Interaction Environments*. Ed. by Julie A. Jacko. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 376–385. ISBN: 978-3-642-21616-9.

[11] M. S. Erden and T. Tomiyama. "Human-Intent Detection and Physically Interactive Control of a Robot Without Force Sensors". In: *IEEE Transactions on Robotics* 26.2 (2010), pp. 370–382. ISSN: 1552-3098. DOI: 10.1109/TRO.2010.2040202.

[12] Frank Flemisch et al. *The H-Metaphor as a Guideline for Vehicle Automation and Interaction*. Feb. 2003.

[13] Brian Gerkey. *What is ROS?* [Online; accessed 8-May-2019]. 2011. URL: https://answers.ros.org/question/12230/what-is-ros-exactly-middleware-framework-operating-system/#18055.

[14] W.R. Hamilton. *Lectures on Quaternions: Containing a Systematic Statement of a New Mathematical Method; of which the Principles Were Communicated in 1843 to the Royal Irish Academy; and which Has Since Formed the Subject of Successive Courses of Lectures, Delivered in 1848 and Subsequent Years, in the Halls of Trinity College, Dublin: with Numerous Illustrative Diagrams, and with Some Geometrical and Physical Applications*. v. 2-4. Hodges and Smith, 1853. URL: https://books.google.dk/books?id=PJIKAAAAYAAJ.

[15] A.S. Hardy. *Elements of Quaternions*. Ginn, Heath, & Company, 1881. URL: https://books.google.dk/books?id=YNE2AAAAMAAJ.

[16] John R. Koza et al. "Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming". In: *Artificial Intelligence in Design '96*. Ed. by John S. Gero and Fay Sudweeks. Dordrecht: Springer Netherlands, 1996, pp. 151–170. ISBN: 978-94-009-0279-4. DOI: 10.1007/978-94-009-0279-4_9. URL: https://doi.org/10.1007/978-94-009-0279-4_9.

[17] L. Li et al. "Cognitive Cars: A New Frontier for ADAS Research". In: *IEEE Transactions on Intelligent Transportation Systems* 13.1 (2012), pp. 395–407. ISSN: 1524-9050. DOI: 10.1109/TITS.2011.2159493.

[18] D.T. McRuer et al. *Human Pilot Dynamics in Compensatory Systems; Theory, Models, and Experiments with Controlled Element and Forcing Function Variation*. 1965. URL: https://books.google.es/books?id=nFncNwAACAAJ.

[19] D.A. Norman. *The Design of Everyday Things*. A currency book n.º 842. Doubleday, 1990. ISBN: 9780385267748. URL: https://books.google.es/books?id=b09jQgAACAAJ.

[20] Donald Norman. "The 'Problem' with Automation: Inappropriate Feedback and Interaction, not 'Over-Automation'". In: *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 327 (May 1990), pp. 585–93. DOI: 10.1098/rstb.1990.0101.

[21] David Poole. *Computational intelligence : a logical approach*. New York: Oxford University Press, 1998. ISBN: 9780195102703.

[22] Python.org. *What is Python? Executive Summary*. [Online; accessed 9-May-2019]. URL: https://www.python.org/doc/essays/blurb/.

[23] J. Rasmussen. "Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.3 (1983), pp. 257–266. ISSN: 0018-9472. DOI: 10.1109/TSMC.1983.6313160.

[24] Robotics and IoT. *What is ROS (Short description)*. [Online; accessed 8-May-2019]. 2016. URL: https://www.youtube.com/watch?v=UL1_Ue4rUWs.

[25] ROS.org. *ROS Navigation Stack summary*. [Online; accessed 10-May-2019]. URL: http://wiki.ros.org/navigation.

[26] ROS.org. *ROS website*. [Online; accessed 8-May-2019]. 2016. URL: https://www.ros.org/.

[27] ROS.org. *Roscpp package summar*. [Online; accessed 9-May-2019]. URL: http://wiki.ros.org/roscpp.

[28] ROS.org. *Rospy package summary*. [Online; accessed 8-May-2019]. URL: http://wiki.ros.org/rospy.

[29] ROS.org. *Understanding Nodes*. [Online; accessed 8-May-2019]. URL: http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes.

[30] Stuart Russell. *Artificial intelligence : a modern approach*. Upper Saddle River, N.J: Prentice Hall/Pearson Education, 2003. ISBN: 0137903952.

[31] L. Saleh et al. "Shared Steering Control Between a Driver and an Automation: Stability in the Presence of Driver Behavior Uncertainty". In: *IEEE Transactions on Intelligent Transportation Systems* 14.2 (2013), pp. 974–983. ISSN: 1524-9050. DOI: 10.1109/TITS.2013.2248363.

[32] Nadine B. Sarter and David D. Woods. "How in the World Did We Ever Get into That Mode? Mode Error and Awareness in Supervisory Control". In: *Human Factors* 37.1 (1995), pp. 5–19. DOI: 10.1518/001872095779049516. eprint: https://doi.org/10.1518/001872095779049516. URL: https://doi.org/10.1518/001872095779049516.

[33] Paul M Satchell. *Cockpit monitoring and alerting systems*. Routledge, 2016.

[34] C. J. Tomlin, J. Lygeros, and S. Shankar Sastry. "A game theoretic approach to controller design for hybrid systems". In: *Proceedings of the IEEE* 88.7 (2000), pp. 949–970. ISSN: 0018-9219. DOI: 10.1109/5.871303.

[35] Michel Verhaegen et al. "Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations". In: *Vehicle System Dynamics* 44 (Aug. 2006), pp. 569–590. DOI: 10.1080/00423110600563338.

[36] Christopher D Wickens et al. *Engineering psychology and human performance*. Psychology Press, 2015.

[37] Earl L Wiener. "Human factors of advanced technology (glass cockpit) transport aircraft". In: (1989).

Appendix A

Helper and Hinder code

The following pages contain the Hinder and Helper scripts.

```python
#!/usr/bin/env python

import random  # module for generating random numbers
import math
import rospy
import math  # importing the math module for calculating square roots
import pdb
# importing the module for knowing the odometry of the robot
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
from tf.transformations import euler_from_quaternion

speed = 0  # Declaration of the variable that will read the linear speed


def readOdometry(msg):
    global roll, pitch, yaw
    # declaring of the variables that will be override by the script to track the position of the robot
    x_robot = msg.pose.pose.position.x
    y_robot = msg.pose.pose.position.y
    z_robot = msg.pose.pose.position.z
    # Creating a list with the 4 values that compose the orientation quaternion of the position message
    orientation_list = [msg.pose.pose.orientation.x, msg.pose.pose.orientation.y,
                        msg.pose.pose.orientation.z, msg.pose.pose.orientation.w]
    # Assigning their values to roll, pitch and yaw variables through the euler_from_quaternion conversion
    (roll, pitch, yaw) = euler_from_quaternion(orientation_list)
    print yaw


def main(geometry_msgs):
    global speed
    speed = geometry_msgs.linear.x


rospy.init_node('hinder')

odom_sub = rospy.Subscriber('/odom', Odometry, readOdometry)
odom_sub = rospy.Subscriber('cmd_vel', Twist, main)
pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
r = rospy.Rate(5)
command = Twist()

while not rospy.is_shutdown():  # making a loop
    if speed != 0.0:
```

```
45          print('You Pressed A Key!')
46          target_rad = yaw + random.uniform(-0.4,0.4)
47          command.angular.z = target_rad
48          pub.publish(command)
49          print ("target={} current:{}", target_rad)
50          r.sleep()
51
```

```python
#!/usr/bin/env python

# importing math package for square roots, sinus and other calculations.
import math
# importing the rospy module to initialize the node.
import rospy
# importing the module for knowing the odometry of the robot.
from nav_msgs.msg import Odometry
# importing the transform converter between eulers and quaternions
from tf.transformations import euler_from_quaternion, quaternion_from_euler
# Import Twist() function to make the robot rotate once obtained the proper angle.
from geometry_msgs.msg import Twist

# declaration of the hardcoded positions of both cones (z coordinate is not declared because of not being used by the script).
cone1_x = 3.0
cone1_y = 3.0
cone1_point = [cone1_x, cone1_y]
cone2_x = 3.0
cone2_y = -3.0
cone2_point = [cone2_x, cone2_y]
# threshold which points out the radius around the cones where the script starts having effect on the robot.
threshold = 3
# Initialization of the global variables
alpha = 0  # used in calculateAngle(...)
kp = 0.5  # variable to control rotation speed
yaw = 0  # Measuarement of the yaw component of the rotation of the robot
target_angle = 0  # Initialization of the target angle for the robot


def calculateAngle(robot_point, cone_point, pivot_point):
    global alpha
    CP = math.sqrt(abs((pivot_point[0]-cone_point[0]) **
                       2 - (pivot_point[1]-cone_point[1])**2))
    CR = math.sqrt(abs((robot_point[0]-cone_point[0]) **
                       2 - (robot_point[1]-cone_point[1])**2))
    if CR < 0.6:
        print "Target reached"
    if CP < CR:
        radAlpha = math.asin(CP/CR)
    else:
        radAlpha = math.asin(CR/CP)
    alpha = radAlpha * 180 / math.pi
    return alpha


def helper(msg):
```

```
47      global target_angle
48      global yaw
49      # declaring of the variables that will be override by the script to track
          the position of the robot
50      x_robot = msg.pose.pose.position.x
51      y_robot = msg.pose.pose.position.y
52      # declaring of the Vector2 which stores the position of the robot as point
          on a 2D plane
53      robot_point = [x_robot, y_robot]
54      # Creating a list with the 4 values that compose the orientation quaternion
           of the position message
55      orientation_list = [msg.pose.pose.orientation.x,
          msg.pose.pose.orientation.y,
56                          msg.pose.pose.orientation.z,
                    msg.pose.pose.orientation.w]
57      # Assigning their values to roll, pitch and yaw variables through the
          euler_from_quaternion conversion
58      (roll, pitch, yaw) = euler_from_quaternion(orientation_list)
59
60      # Code for calculating which cone is the closest one
61      if(math.sqrt((cone1_point[0]-robot_point[0])**2 + (cone1_point[1]-
          robot_point[1])**2)) < threshold:
62          pivot_point = [cone1_point[0], robot_point[1]]
63          # Calculation of in which quadrant around the cone is the robot if the
              cone is cone number 1
64          if x_robot > cone1_x and y_robot > cone1_y:
65              # Definition of the line to cross diagonally through quadrant
                  number 1
66              alpha = calculateAngle(robot_point, cone1_point, pivot_point)
67              target_angle = alpha - 180
68          elif x_robot < cone1_x and y_robot > cone1_y:
69              # Definition of the line to cross diagonally through quadrant
                  number 2
70              alpha = calculateAngle(robot_point, cone1_point, pivot_point)
71              target_angle = -alpha
72
73          elif x_robot < cone1_x and y_robot < cone1_y:
74              # Definition of the line to cross diagonally through quadrant
                  number 3
75              alpha = calculateAngle(robot_point, cone1_point, pivot_point)
76              target_angle = alpha
77
78          elif x_robot > cone1_x and y_robot < cone1_y:
79              alpha = calculateAngle(robot_point, cone1_point, pivot_point)
80              target_angle = 180 - alpha
81
82      elif(math.sqrt((cone2_point[0]-robot_point[0])**2 + (cone2_point[1]-
          robot_point[1])**2)) < threshold:
83              # Calculation of in which quadrant around the cone is the robot if
```

```
                the cone is cone number 2
        pivot_point = [cone2_point[0], robot_point[1]]
        if x_robot > cone2_x and y_robot > cone2_y:
            alpha = calculateAngle(robot_point, cone2_point, pivot_point)
            target_angle = alpha - 180
        elif x_robot < cone2_x and y_robot > cone2_y:
            # Definition of the line to cross diagonally through quadrant
              number 2
            alpha = calculateAngle(robot_point, cone2_point, pivot_point)
            target_angle = -alpha
        elif x_robot < cone2_x and y_robot < cone2_y:
            # Definition of the line to cross diagonally through quadrant
              number 3
            alpha = calculateAngle(robot_point, cone2_point, pivot_point)
            target_angle = alpha
        elif x_robot > cone2_x and y_robot < cone2_y:
            # Definition of the line to cross diagonally through quadrant
              number 4
            alpha = calculateAngle(robot_point, cone2_point, pivot_point)
            target_angle = 180 - alpha
    else:
        print ("No cone in range")


################################################################################
##############


rospy.init_node('helper')

odom_sub = rospy.Subscriber('/odom', Odometry, helper)
pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
r = rospy.Rate(10)
command = Twist()


while not rospy.is_shutdown():
    target_rad = target_angle * math.pi/180
    #print command.angular.z
    command.angular.z = kp * (target_rad-yaw)
    pub.publish(command)
    #print ("target={} current:{}", target_angle, yaw)
    r.sleep()


'''
def testIntersection(pt1, pt2, ptA, ptB):
    """ prints out a test for checking by hand... """
    print "Line segment #1 runs from", pt1, "to", pt2
```

```
127     print "Line segment #2 runs from", ptA, "to", ptB
128
129     result = intersectLines(pt1, pt2, ptA, ptB)
130     print "    Intersection result =", result
131     print
132
133 '''
134
135 '''
136
137 # Code for finding the intersection between two lines
138 def findslope(point1, point2):
139     return (point2[1] - point1[1]) / (point2[0] - point1[0])
140
141
142 def calculateIntersection(point1, point2, pointA, pointB):
143     # Calculating the slope of the line bebtween the robot and the pivot
144     slope1 = findslope(point1, point2)
145     # Calculating the slope of the line between the cone and the diagonal
146     slope2 = findslope(pointA, pointB)
147     x = (slope2 * pointA[0] - pointA[1] + point1[1] - slope1 * point1[0]
148          ) / (slope2 - slope1)  # Calculating x of the intersection point
149     y = point1[1] - slope1*point1[0] + slope1 * \
150         x  # Calculating y of intersection point
151     return [x, y]  # Returns the intersection point
152
153
154 '''
155
156 '''
157
158 def eulerToDegree(euler):
159     return ((euler) / (2 * math.pi)) * 360
160
161 # Code for transforming the quaternion values of the orientation message from
      the /odom topic to euler angles
162
163
164 def get_rotation(msg):
165     # Creating a list with the 4 values that compose the orientation quaternion
         of the position message
166     global roll, pitch, yaw
167     orientation_list = [msg.pose.pose.orientation.x,
        msg.pose.pose.orientation.y,
168                         msg.pose.pose.orientation.z,
                    msg.pose.pose.orientation.w]
169     # Assigning their values to roll, pitch and yaw variables through the
        euler_from_quaternion conversion
170     (roll, pitch, yaw) = euler_from_quaternion(orientation_list)
```

```
171     print yaw
172     # Function made for getting to which quadrant the robot is looking to
173     '''
174 '''
175    if 0 > yaw and yaw >= -1.5:
176       print ("The robot is looking to quadrant number 1")
177    elif -1.5 > yaw and yaw >= -3.2:
178       print ("The robot is looking to quadrant number 2")
179    elif 1.5 < yaw and yaw <= 3.2:
180       print ("The robot is looking to quadrant number 3")
181    elif 0 < yaw and yaw <= 1.5:
182       print ("The robot is looking to quadrant number 4")
183       '''
184 '''
185
186     # Conversion of the yaw to a rotation anglee in degrees
187     rotation_angle = eulerToDegree(yaw)
188     # Calculation of the y coordinate of the point for the line of the vector
189     point_y = (((math.sin(180 - (math.radians(rotation_angle))*2))
190                 * (400 / math.radians(rotation_angle)))**2) / 800
191     # Defining the points that we will use for creating a line in the
          orientation vector
192     # Calculation of the x coordinate of the point for the line of the vector
193     point_x = math.sqrt((((math.sin(180 - (math.radians(rotation_angle))*2))
194                          * (400 / math.radians(rotation_angle)))**2) -
                       (point_y) ** 2)
195     # Defining the line with the point inside the orientation vector of the
          robot
196     # orientation_line=LineString([(x_robot, y_robot), (point_x, point_y)])
197
198 # function called every frame that updates the values of the robot_point
      variables
199
200
201 def calculateRotation(robot_point, cone1_point, intersection_point):
202     #wait = rospy.Rate(10)
203     if cone == 1:
204         PC = math.sqrt((intersection_point[0] - cone1_point[0])
205                        ** 2 + (intersection_point[1] - cone1_point[1])**2)
206         RC = math.sqrt((robot_point[0] - cone1_point[0])
207                        ** 2 + (robot_point[1] - cone1_point[1])**2)
208         PR = math.sqrt((intersection_point[0] - robot_point[0])
209                        ** 2 + (intersection_point[1] - robot_point[1])**2)
210         global target_angle
211         target_angle = yaw + math.acos(
212             (PR**2 + RC**2 - PC**2) / (2 * PR**2 * RC**2))
213         # wait.sleep()
214         #print target_angle / (math.pi/180)
215     else:
```

```
216         PC = math.sqrt((intersection_point[0] - cone2_point[0])
217                        ** 2 + (intersection_point[1] - cone2_point[1])**2)
218         RC = math.sqrt((robot_point[0] - cone2_point[0])
219                        ** 2 + (robot_point[1] - cone2_point[1])**2)
220         PR = math.sqrt((intersection_point[0] - robot_point[0])
221                        ** 2 + (intersection_point[1] - robot_point[1])**2)
222         global target_angle
223         target_angle = yaw + math.acos(
224             (PR**2 + RC**2 - PC**2) / (2 * PR**2 * RC**2))
225         # wait.sleep()
226         #print target_angle / (math.pi/180)
227
228
229 '''
230
```