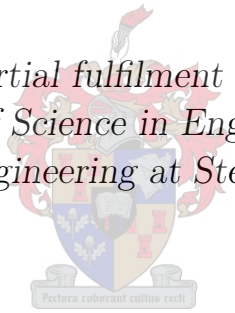
Development of distributed control system for SSL soccer robots

by

David Schalk Holtzhausen

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering (Mechatronic) in the Faculty of Engineering at Stellenbosch University



Supervisors:

Prof. K. Schreve
Dr. M. Blanckenberg

March 2013

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 2012/12/01 .....

Abstract

This thesis describes the development of a distributed control system for SSL soccer robots. The project continues on work done to develop a robotics research platform at Stellenbosch University. The wireless communication system is implemented using Player middleware. This enables high level programming of the robot drivers and communication clients, resulting in an easily modifiable system. The system is developed to be used as either a centralised or decentralised control system. The software of the robot's motor controller unit is updated to ensure optimal movement. Slippage of the robot's wheels restricts the robot's movement capabilities. Trajectory tracking software is developed to ensure that the robot follows the desired trajectory while operating within its physical limits.

The distributed control architecture reduces the robots dependency on the wireless network and the off-field computer. The robots are given some autonomy by integrating the navigation and control on the robot self. Kalman filters are designed to estimate the robots translational and rotational velocities. The Kalman filters fuse vision data from an overhead vision system with inertial measurements of an on-board IMU. This ensures reliable and accurate position, orientation and velocity information on the robot. Test results show an improvement in the controller performance as a result of the proposed system.

Uittreksel

Hierdie tesis beskryf die ontwikkeling van 'n verspreidebeheerstelsel vir SSL sokker robotte. Die projek gaan voort op vorige werk wat gedoen is om 'n robotika navorsingsplatform aan die Universiteit van Stellenbosch te ontwikkel. Die kommunikasiestelsel is geïmplementeer met behulp van Player middelware. Dit stel die robotbeheerders en kommunikasiekliënte in staat om in hoë vlak tale geprogrameer te word. Dit lei tot 'n maklik veranderbare stelsel. Die stelsel is so ontwikkel dat dit gebruik kan word as óf 'n gesentraliseerde of verspreidebeheerstelsel. Die sagteware van die motorbeheer eenheid is opgedateer om optimale robot beweging te verseker. As die robot se wiele gly beperk dit die robot se bewegingsvermoëns. Trajekvolgings sagteware is ontwikkel om te verseker dat die robot die gewenste pad volg, terwyl dit binne sy fisiese operasionele grense bly.

Die verspreibeheerargitektuur verminder die robot se afhanklikheid op die kommunikasienetwerk en die sentrale rekenaar. Die robot is 'n mate van outonomie gegee deur die integrasie van die navigasie en beheer op die robot self te doen. Kalman filters is ontwerp om die robot se translasie en rotasie snelhede te beraam. Die Kalman filters kombineer visuele data van 'n oorhoofse visiestelsel met inertia metings van 'n IMU op die robot. Dit verseker betroubare en akkurate posisie, oriëntasie en snelheids inligting. Toetsresultate toon 'n verbetering in die beheervermoë as 'n gevolg van die voorgestelde stelsel.

Dedications

To the Lord my Saviour
There, but for the grace of God, go I

Acknowledgements

I would like to express my sincere gratitude to the following people and organizations:

- The CSIR for providing the financial means by funding the research and providing me with this opportunity
- Eskom for providing for my living expenses throughout my studies
- Lorita Jacobs for assisting me with understanding the work she had done on the project.
- Mr. Ferdie Zietsman and the workshop personnel for their assistance in providing me with a robot to work on.
- My friends Stephan Heunis and Andre Smith for providing me with the necessary motivation to keep on giving my best.
- Reynaldo Rodriguez without whom I would not have made it this far, for his immediate availability and technical assistance.
- My supervisors, Prof. Kristiaan Schreve and Dr. Mike Blanckenberg for their continual support and unwavering faith in my abilities.
- Lastly my family who always loved and supported me and taught me... *"I count all things but loss for the excellency of the knowledge of Christ Jesus my Lord"*.

Contents

List of Figures

List of Tables

Nomenclature

Motor Control

e	Error in the motors speed	[rpm]
U	PID controller output	[rpm]
K_i	Integral constant of PID controller	[ ]
K_p	Proportional constant of PID controller	[ ]

Motion Control

a_{max}	Maximum resultant acceleration	[m/s^2]
$a_{x,max}$	Maximum acceleration in the x direction	[m/s^2]
$a_{y,max}$	Maximum acceleration in the y direction	[m/s^2]
v_{max}	Maximum resultant velocity	[m/s]
$v_{x,max}$	Maximum velocity in the x direction	[m/s]
$v_{y,max}$	Maximum velocity in the y direction	[m/s]
$\ddot{x}$	Acceleration in the x direction	[m/s^2]
$\ddot{y}$	Acceleration in the y direction	[m/s^2]
α	Synchronisation scaling factor	[ ]
$\ddot{\theta}$	Angular acceleration	[rad/s^2]

State Estimator

F	Fundamental matrix	[ ]
G	Discrete input matrix	[ ]
H	Discrete output matrix	[ ]
I	Identity matrix	[ ]
R	Measurement noise matrix	[ ]
u	Input control vector	[ ]
v	Measurement noise vector	[ ]
w	Process noise vector	[ ]
x	State vector	[ ]
y	Measurement vector	[ ]

F_c	System dynamics matrix	[ ]
G_c	Input matrix	[ ]
H_c	Output matrix	[ ]
K^k	Kalman gain	[ ]
M^k	Predicted covariance matrix	[ ]
P^k	Covariance matrix	[ ]
S^k	Measurement predicted covariance matrix	[ ]
Q^k	Discrete process noise matrix	[ ]
T_s	Sampling time	[s]
u^k	Input vector	[ ]
v^k	Measurement noise vector	[ ]
w^k	Process noise vector	[ ]
x^k	State vector	[ ]
y	Output vector	[ ]
x_p^k	Predicted state vector	[ ]
θ	Estimated orientation of robot	[rad]
τ	Filter time constant	[1/s]
θ_c	Orientation of robot obtained from vision system	[rad]
θ_{hp}	High-pass filtered orientation component	[rad]
θ_{lp}	Low-pass filtered orientation component	[rad]
Φ_s	Continuous process noise spectral density	[ ]
$\dot{\theta}_g$	Angular velocity of robot obtained from gyroscope	[rad/s]

Appendix B

l	Gear ratio	[ ]
m	Mass of the robot	[kg]
g	Gravity of earth	[m/s^2]
J	Inertia moment around rotation axis	[kgm^2]
Bx	Viscous friction coefficient for the x direction	[N/m/s]
By	Viscous friction coefficient for the y direction	[N/m/s]
$B\omega$	Viscous friction coefficient around rotation axis	[Nm/rad/s]
Cx	Coulomb friction coefficient for the x direction	[N]
Cy	Coulomb friction coefficient for the y direction	[N]
$C\omega$	Coulomb friction coefficient around rotation axis	[Nm]
E_n	Motor voltage	[V]
F_{Bx}	Viscous friction forces in the x direction	[N]
F_{By}	Viscous friction forces in the y direction	[N]

F_{Cx}	Coulomb friction forces in the x direction	[N]
F_{Cy}	Coulomb friction forces in the y direction	[N]
F_R	Resultant force acting on robot	[N]
i_n	Motor current	[A]
K_v	EMF Motor constant	[V/rad/s]
K_t	Motor torque constant	[Nm/A]
P_m	Maximum output power of the motor	[W]
R_m	Motor resistance	[V/A]
R_r	Robot radius	[m]
r_w	Wheel radius	[m]
$T_{B\omega}$	Viscous friction torque around rotation axis	[Nm]
$T_{C\omega}$	Coulomb friction torque around rotation axis	[Nm]
T_m	Motor torque	[Nm]
T_n	Torque of wheel n	[Nm]
v_n	Wheel velocity	[m/s]
v_x	Robot x velocity	[m/s]
v_y	Robot y velocity	[m/s]
μ_f	Friction between the robot's wheels and the field	[ ]
θ_n	Angle of wheel with respect to x-axis of robot	[rad]
ω_r	Angular rotation velocity of robot	[rad/s]
ω_m	Angular velocity of motor	[rad/s]
ω_w	Angular velocity of wheel	[rad/s]

Acronyms

ACK Acknowledge

AI Artificial Intelligence

API Application Programming Interface

CLI Command-Line Interface

CF Complimentary Filter

CO Controller Output

DOF Degrees of Freedom

EKF Extended Kalman Filter

I^{2}C Inter-Integrated Circuit

IMU Inertial Measurement Unit

GPL General Public License

GUI Graphical User Interface

HDL Hardware Description Language

HLC High Level Controller

FPGA Field Programmable Gate Array

KF Kalman Filter

LLC Low Level Controller

LUT Lookup Table

MSL Middle Size League

NACK Negatively Acknowledge

OFC Off-field Computer

PID Proportional Integral and Derivative

RDE Robotic Development Environment

rpm Revolutions per Minute

SBC Single Board Computer

SP Set Point

SPI Serial Peripheral Interface Bus

SSL Small Size League

TCP Transmission Control Protocol

TLC Trajectory Linearisation Control

VCSEL Vertical-Cavity Surface-Emitting Laser

WLAN Wireless Local Area Network

XDR External Data Representation

1. Introduction

1.1. Background

Robotics is an exciting innovative technology that is becoming an integral part of society. Robotics has a large influence on peoples lives. The magnitude of this influence varies from robots building the cars and appliances that are used every day; to the use of robots in military applications. This importance of robotic systems necessitates that research be done in this field to further improve this technology and the impact thereof on our society.

This project continuous on work done towards creating a robotic research platform at Stellenbosch University. The goal of the research platform is to create a modular robotic system, which is adaptable and can be modified using high level programming languages. In order to facilitate this goal it was decided to develop a team of robots that could participate in a division of the RoboCup™.

RoboCup™ is an international research and educational initiative that was created to promote robotics research. This initiative attempts to promote research in Artificial Intelligence (AI) and robotics by creating a platform for competitive and attractive challenges in the fields of robotics and AI. The long term goal of this initiative is:

> "*By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply* [sic] *with the official rules of the FIFA, against the winner of the most recent World Cup.*" (**?**).

In order to accomplish this goal different leagues were created. The league that was focussed on in this project is RoboCup Soccer. The aim of RoboCup Soccer is to promote research in the fields of cooperative multi-robot and multi-agents systems in dynamic competitive environments. This is done through five soccer leagues in which teams of autonomous robots compete against each other. The leagues are Simulation League, Small Size League (SSL), Middle Size League (MSL), Standard Platform and Humanoid League. These leagues all differ in complexity and in their research goals. This project will focus on developing robots for the SSL, because there is some overlap between the SSL and the MSL, some of the research done on the MSL will be discussed.

The SSL focusses on developing a multi-agent robotics team. This league uses either a centralised architecture or distributed architecture. A distributed architecture is a combination of centralized as well as decentralised control. However, the SSL is usually implemented using a control architecture that is mostly centralised, with the robots having little processing power and only performing basic tasks. This architecture is used to enable the robots to play soccer without any on-board sensors and rely on an Off-field Computer (OFC) for their commands. Due to the fact that no on-board sensors are required the robots are smaller and thus the name Small Size League. The robots are restricted to a maximum diameter of 180 mm and a maximum height of 150 mm. Some of the robots are able to move at speeds of up to 3.5 m/s and kick the ball up to 15 m/s (**?**, **?**). A team consists of a maximum of six robots. They play on a green carpeted field that is 4.05 m by 6.05 m in size.

All the teams in the SSL use the same open-source vision system called SSL-Vision. The vision system uses two overhead cameras situated 4.0 m above the field. Information regarding the position of all the robots and the ball is sent to the OFC. This data is processed on the computer and intelligent decisions are made accordingly. The OFC communicate with the robots using a wireless network. Figure 1.1 shows the typical layout of the SSL.

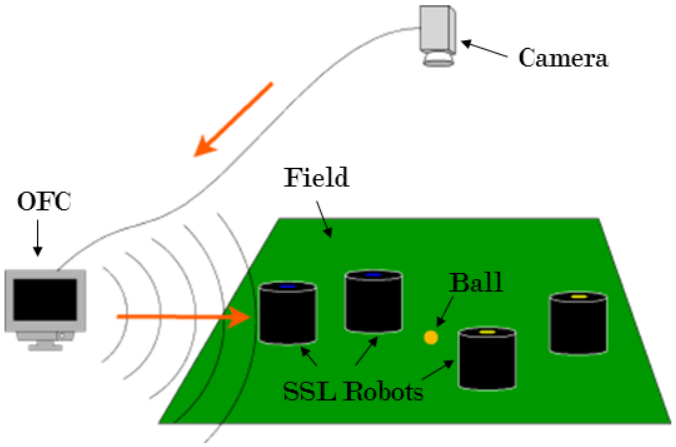


Figure 1.1: Control architecture of SSL (**?**)

The SSL game is almost fully autonomous. The only human intervention is that of a referee and an assistant referee. They communicate with the OFC using software called Referee Box. Other than that there is no human intervention, the robots play on their own and the AI software controls every aspect of the gameplay. The AI software runs on the OFC where most of the processing is done. From here the velocity commands are sent to the robots.

The MSL focusses on full autonomy and cooperation between the different robots. These robots have a maximum diameter of 50 cm and the teams consist of teams of up to 6 members. All sensors are mounted on the robots themselves

and there is no OFC. Thus the robots have to work together to plan the game strategies and to facilitate good perception of the playing field. These robots communicate with each other using a wireless network. The MSL robots use various on-board sensors to facilitate localization and navigation.

1.2. Objectives

The project's main objectives are listed below. The ultimate goal is to develop and test a distributed control system.

1. Develop and test a centralised control system using SSL-Vision system.
    - Develop a driver for the soccer robots based on Player
    - Develop communication client using Player which implements the SSL-Vision system.
    - Have a soccer robot execute basic movement commands.
2. Develop a distributed control system using the SSL-Vision system combined with on-board sensors.
    - Implement on-board sensors.
    - Implement sensor fusion using estimator algorithms.
    - Implement trajectory following to test the control system.
3. Test and evaluate the developed distributed control system using SSL soccer robots.
    - Compare the performance when different sensors are used.
    - Evaluate the performance of the distributed control architecture.

1.3. Motivation

SSL soccer robots are very suitable for research purposes. This is due to the versatility of these robots. By modifying the robots' hardware or the system architecture research can be done in a wide variety of the fields such as image recognition, AI, motion control, behaviour strategy and communication. The research suitability of the robots is increased by increasing the modifiability and versatility of the robots.

Most SSL teams use a Field Programmable Gate Array (FPGA) as a main controller on the robot. FPGA have to be programmed in Hardware Description Language (HDL) which is complicated and a lot more difficult than programming in high-level languages such as C++ or Java. Furthermore by designing the robots to be modular and semi-autonomous enables them to

be used in both a centralised and distributed control system. Thus enabling robotic research in both control architectures, each yielding different challenges and opportunities. However, although the soccer robots might be versatile and provide the functionality enabling them to be used by either a centralised or distributed control system, it places a challenge on the communications system. A rapidly changeable and modifiable communication system is required in order to communicate effectively with the robots while enabling research and development in both control architectures. This means that a communication system which can be rapidly changed and tested is required.

The main sensor input in the SSL is that of the overhead cameras. As a result most SSL teams use a centralised control architecture, where an OFC uses the vision information to control the soccer robots. This system can be used as a platform to test AI and behaviour strategies. Mostly the robots used in a centralised control system have no or very limited autonomy. In order to use these robots in a distributed control system they need to be semi-autonomous. In such a control system the OFC will still act as a central controller, deciding on the strategy and game plan. However the OFC will be able to allocate tasks to the robots. The robots can implement these tasks using on-board sensors and control algorithms.

The distributed control architecture enables the robots to perform some autonomous tasks, however the successful execution of these tasks is dependent on the robot's ability to effectively utilise on-board sensors. Various sensors could be used to measure different aspects of the robot's state. These include gyroscopes, accelerometers, compasses, encoders and many more. The sensor that will be used must be chosen such that they optimise the usage of the limited space on the SSL robots, while enabling the estimation of all the required robot attributes. Therefore research will be done to investigate which is the optimal sensor to use with the soccer robots.

The correct choice of sensors will ensure that the robot's behaviour can be observed, however sensor data is never without noise or biased errors. **?** did real-time experimental testing and found that accurate position and orientation measurements are essential for controller performance. Cumulative errors are introduced when individual sensors are used to obtain an estimate of the robot's position. Various algorithms and techniques have been developed to estimate the actual movement of the robot based on multiple sensor inputs. Sensor fusion is when data obtained from various sensors is used to generate information that is more accurate than the information obtained from the data of the sensors individually.

There are numerous techniques for doing this, such as Markov Localization Monte Carlo Localization and various types of Kalman filters (**?**). These techniques can be used to enable the robot to accurately determine its own state. This information regarding the state of the robot can be used as feedback for the controllers on the robot, enabling the robot to execute desired commands. Therefore it is clear that in order to effectively implement a distributed control

system an accurate state estimator is required.

Optimally implemented distributed control architecture will be advantageous to a team of soccer robots. Therefore research will be done to develop such a control system. This involves the implementation of an OFC with intelligent soccer robots. Sensor fusion will be performed to estimate the robot state to enable the robots to move from one position to another.

1.4. Methodology

This section explains the research methods that were followed during this project. This project continuous on work done by **?** and **?**. **?** built a SSL robot and designed and implemented a motor controller with a Low Level Controller (LLC). **?** did very little work on developing a communication system, but did do some research regarding middleware. **?** designed a second iteration robot based on the design of **?**. This included redesigning the motor controller and the motor controller software.

The implementation of the motor controller was not tested and this project started by completing and improving the motor controller software. This was done because an optimal functioning robot was required for this project. This required that the software for the motor controllers had to be modified and tested. In order to do research on the control of the soccer robots a communication system was required. Research was done on different types of middleware, which is the communication platform between the robots and the central controller on the OFC. The middleware was tested and developed throughout the project. The middleware was continuously changed to fulfil the requirements of the project.

The majority of SSL teams use a centralised control system. The first objective was therefore to develop a centralised control system using overhead cameras with the SSL-vision system. This required that the SSL-vision system be implemented at Stellenbosch University. Furthermore in order to move a robot from one position to another, a velocity profile is required that describes the desired movement of the robot with respect to time. An existing algorithm was implemented to generate velocity profiles for the robots. This centralised control system was used to test the middleware and improve on the overall system performance.

Control architectures were researched and the distributed control architecture was selected for further development. The distributed control architecture is an extension of the centralised control system; the difference is that the robots are semi-autonomous. It was decided to focus on developing one specific robot, because the goal of this project is not to develop a team of soccer robots, but rather develop the capabilities of the individual robots. The distributed control system has the ability to control multiple robots. However, as the interaction and behaviour of multiple robots are outside the scope of this

project only the ability of individual robots to accurately execute movement commands by using on-board sensors will be developed.

Sensors were added to the robot to enable it to execute commands semi-autonomously. Estimators are used to better determine the state of the robot than the individual sensors would have. Estimator algorithms were investigated and the implementation and design is discussed in Chapter 6. Estimators were developed and implemented to estimate the velocity and position of the robot in the x and y directions as well as the angular speed and angular rotation of the robot. Once these estimators were implemented and thoroughly tested they were used as the input to the Proportional Integral and Derivative (PID) control system to keep the robot on its desired trajectory. The ability of the robots to follow the desired trajectory is measured. This is done for different robot setups as well as for different estimator setups. The deviation from the specified trajectory is quantified and evaluated.

2. Control Architecture

2.1. Introduction

The SSL RoboCup teams use two types of control architectures: centralised or distributed. Each control system consists of various agents that need to fulfil a task. The agents in SSL can be categorised as either a simple robot or a semi-autonomous robot (**?**). Simple robots are used in a centralised control system, they are basic and primarily rely on an OFC for commands and computational power. Whereas semi-autonomous robots are used in a distributed control system and have the ability to perform certain tasks and make basic decisions, while the bulk of the processing still occurs on an OFC.

The SSL usually implements a centralized control system using simple agents. A global vision system provides the OFCs with position and orientation information regarding the soccer robots. This information is used by the strategy system to determine what every robot should do. The instructions are sent to each robot by the OFC. Thus the OFC does almost all the work, requiring it to be computationally very strong in order to effectively preform all its tasks and communicate them to the robots. This software architecture has certain advantages and disadvantages. The advantage of this system is that the robots are not required to have significant control capabilities and thus require less processing power and sensors. As a result of the size of the SSL robots, space is an issue and thus smaller, simpler robots are beneficial. The drawback with this architecture is that the communication system is burdened with sending large amounts of commands between the robots and the OFC (**?**, **?**).

Instead of using a centralised control system, some SSL teams are using distributed control architectures in order to minimize the amount of commands sent by the OFC. These teams commonly use feedback control, typically PID, on the robots themselves (**?**). Thus, some of the control capabilities are placed on the robots themselves yielding semi-autonomous robots. These robots differ from simple robots in that they have on-board sensors and stronger computational power. This distributed control architecture has the advantage of requiring a less powerful OFC, although it requires hardware and software development for the soccer robots. The hardware development involves implementing various on-board sensors such as position sensors, cameras, infrared

sensors and Inertial Measurement Unit (IMU)s. These robots are required to be computationally stronger than simple robots in order to process the sensor data and perform allocated tasks.

?, **?** and **?** used the RoboCup SSL as a platform for research on developing distributed control systems. **?** stated that the present availability of a wide variety of sensors with embedded controllers justifies the development of distributed control in the SSL. They developed a modular robot using microcontrollers to control the robot and its various on-board sensors. The system was adaptable although the microcontroller has only limited processing power.

? used a FPGA as the central controller to eliminate the delay between the time that each motor receives a new command. The system is seen in figure 2.1. The FPGA yields an increase in speed although the difficulty with programming an FPGA greatly reduces the modifiability of this system, thus restricting the systems use as a research platform. They used an IMU returning acceleration and rotation information as feedback for their PID control. The simulation results obtained indicated a reduction in positional error.

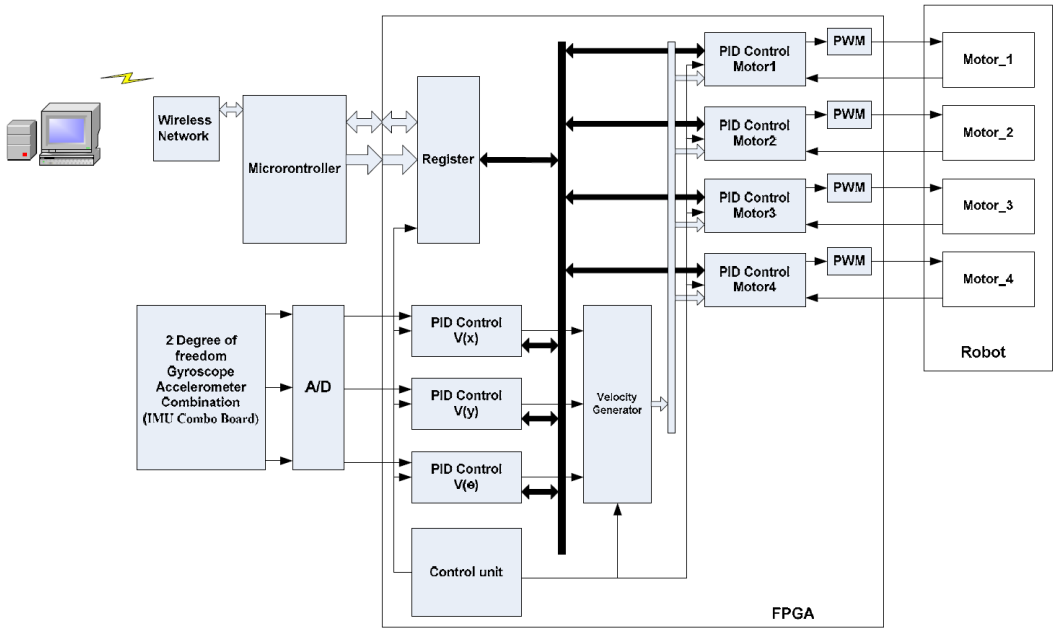


Figure 2.1: Distributed SSL robot's system architecture using a FPGA (**?**)

? discuss the development of a hybrid distributed control architecture. The hybrid refers to the fact that the system is able to effectively perform both reactive and deliberate tasks. When using a centralised control strategy the main computer processes all the information giving it the ability to have a clear understanding of the whole environment and enabling it to make decisions based on control strategies. Deliberative tasks are tasks such as stated in

?: "*generation of strategies, creation and manipulation of the environmental model, and control of robot global navigation*".

A centralised control strategy results in deliberative tasks being performed very effectively (**?**). When using decentralised strategies the data is processed on the different robots, enabling them to react fast on immediate requirements, such as getting control of a ball close by. This results in decentralised control being very effective in preforming reactive tasks; these are tasks that require rapid decisions (**?**).

A distributed control architecture is able to effectively perform both reactive and deliberate tasks. The deliberative tasks can be performed on the OFC and the reactive tasks on the individual robots in order to improve the overall performance of the system. The distributed control system has elements of both decentralized and centralized control strategies. The intelligence of the system is distributed between the various agents in the system. However, the intelligence is not distributed equally. The OFC will act as a central controller, deciding on the strategy and game plan. It will act as the communication server and allocate tasks to the robots on the field. The robots will have on-board sensors and some form of autonomy. They will be able to perform these tasks using the on-board sensors, reducing their dependency on the OFC.

The distributed control architecture has clear benefits. Utilizing this architecture the burden on the wireless communication would be reduced, because the OFC would not need to send as many commands. Furthermore the implementation of on-board sensors on the robot will provide additional information which could improve the robot's positional control and overall performance.

2.2. Proposed Architecture

The architecture for the soccer robots at Stellenbosch are developed to yield a system which is easily changeable. These robots can be used for further research and act as a testbed for other fields of robotic research. The main goal of this project is to develop centralised and distributed control architectures, which allow the robots to reach a specific destination as accurately as possible. Trajectory control is used to control the movement of soccer robots to ensure that they follow the intended path. This involves generating a geometric path that a robot should follow and using some kind of feedback control to ensure that it executes this manoeuvre (**?**).

The software architecture shown in figure 2.2 is that of a typical SSL team. Most of the SSL teams have a High Level Controller (HLC) running on the OFC and a LLC running on the robot. The HLC is used to control the robot's skills, such as `goTo()` or `setVelocity()`. These commands are generated by the Strategy Module and a desired trajectory is generated by the navigation part of the Control Module. The desired path is then sent to the Motion Control algorithm. This module sends the robots velocity commands based

on the vision feedback to ensure that they follow their specified paths. The Motor Control Module on the robot transforms this velocity command into desired wheel velocities. The Motor Control module is part of the LLC which uses the feedback from the motor encoders to ensure that the wheels maintain their desired velocities.

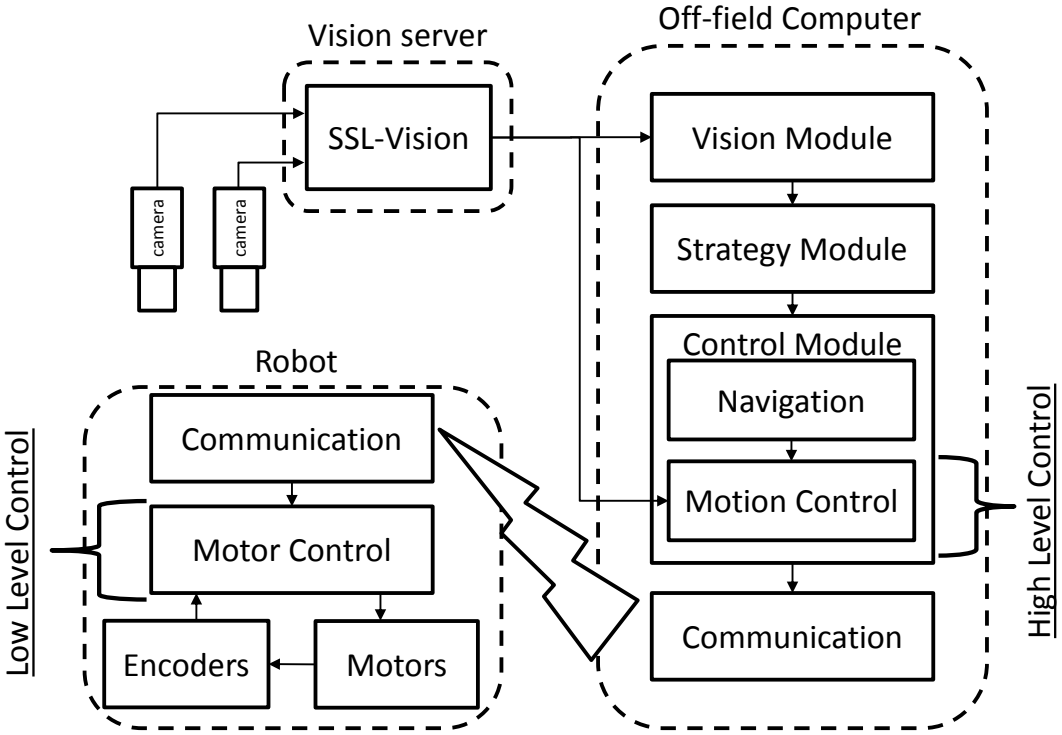


Figure 2.2: Typical SSL software architecture

The disadvantage is that the speed at which new vision information is available is slow. Therefore if the HLC only uses vision data it will run at a lower rate than the dynamics of the soccer robots. Some teams use Kalman filters to predict the robots' behaviour to eliminate the latency issue, although the communication is another bottleneck. The velocity commands need to be sent to every robot in the team resulting in a reduction in the rate at which the Motion Control loop is executed. The fast pace of the SSL games requires that the Motion Control module has to send a great deal of information using the wireless communication.

The high volume of commands that are required to be sent using the wireless communication is as a result of the HLC being run on the OFC as seen in figure 2.2. For example when a robot is required to move from one point on the field to another. With centralized control each robot will have to be sent a range of velocities to enable it to reach a desired destination. This will

place a heavy burden on the communication system. Alternatively when a purely decentralized control system is used the robot will move to a specific point based solely on on-board sensors, as in the MSL. However decentralised control has the drawback of no robot having an accurate global picture of what is happening on the field.

When designing a distributed control architecture it must be decided which of the tasks are reactive and which are deliberate. Then the decision must be made to decide which of the reactive tasks will be done on the robots and which will be left to the OFC. Due to the limited space on the robots only certain reactive tasks can be done by the robot itself. The task that this project is focussing on is getting the robot to a specified destination. The calculation of the geometric obstacle free path is deliberate as it requires an overview of the whole system and therefore will be done by the OFC. However, a major restriction on the movement accuracy is slippage and is discussed in section 5.2.1. Motion control is required to compensate for slippage due to factors such as varying friction.

Slippage is a rapid disturbance in the robot's motion and thus motion control is a reactive task. As a result it is beneficial to have the HLC on the robot instead of on the OFC as is done in figure 2.2. The robot requires on-board sensors to act as feedback for the HLC. The various sensors are discussed in section 4.2.3. It was decided to use an IMU and thus the system will be similar to the one illustrated in figure 2.1. The proposed system is discussed in the rest of this section.

The proposed distributed control architecture is illustrated in figure 2.3. The central controller, situated on the OFC, knows the location of the various robots. The Navigation module on the OFC can then send either a position or velocity command to the HLC which is now situated on the robot. The robot is then able to use on-board sensors to accurately execute the command. The HLC will send velocity commands to the LLC using communication busses on the robot. The communication speed of the robot's busses is much higher than that of the wireless communication which results in a much faster control loop.

Vision data can still be sent to the robot to compensate for biased errors on the on-board sensors. The OFC can send an updated position command if the robot is veering of its desired trajectory or does not reach its desired destination. The goal of this proposed architecture is to reduce the delay between the HLC and LLC by placing both on the robot. The reliance of the robot on the vision data will be reduced as a result of both controllers being situated on the robot.

The proposed system that is built and tested is seen in figure 2.4. The figure shows the control architecture of the proposed system in which both the LLC and HLC are situated on the robot. The robot receives a position command from the OFC, this command specifies the desired x and y position of the robot. This command is used to generate a velocity profile and a geometric

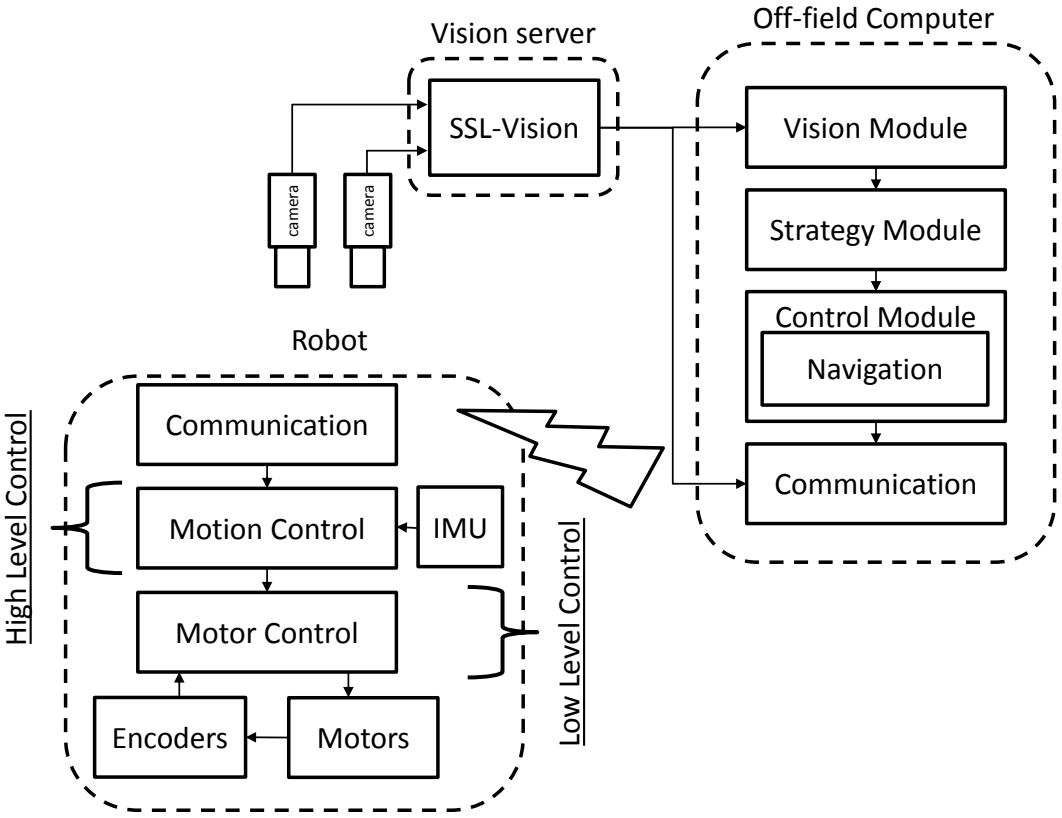


Figure 2.3: Proposed SSL software architecture

trajectory which describes the robot's movement with respect to time.

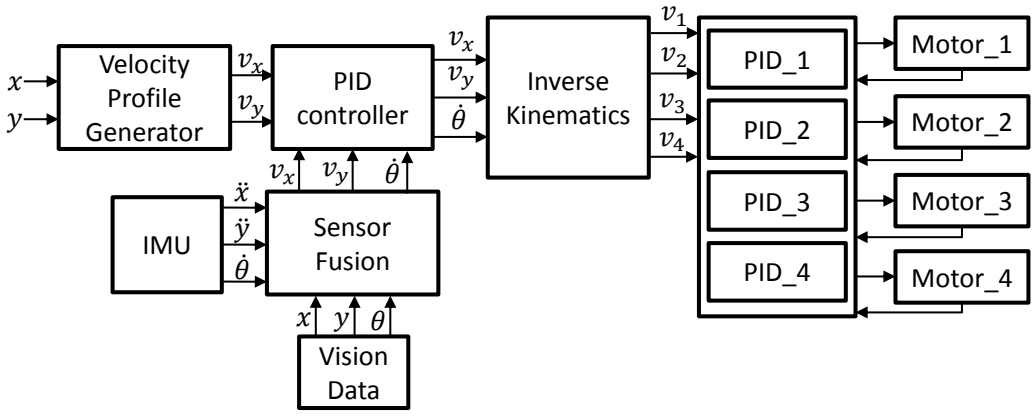


Figure 2.4: The proposed control system

The x and y velocities obtained from the velocity profile are used as set

points in the HLC. The measured output is obtained by performing sensor fusion on the information obtained from the on-board IMU and the vision system. This yields the translational and angular velocity estimate of the robot. The HLC then produces the x, y and angular velocity commands based on the velocity profile and the sensor fusion estimates.

Through inverse kinematics the velocities of the individual wheels are calculated using the outputs of the HLC. These wheel velocities are sent to the LLC. The LLC consists out of four separate PI controllers, which use encoders on the motors as feedback. The design and evaluation of this system is explained in detail in the rest of this report. The next chapter starts by explaining the design of the communication between the robots and the OFC.

3. Communication Design

3.1. Introduction

Communication is a very import aspect of the SSL software architecture. Typically SSL teams use dedicated commercial radio transmitter/receiver units. These systems have high communication rates although they are expensive and need to be integrated into the robot's hardware system. An alternative is middleware software, which is used to provide an adaptable software solution. It runs on standard operating systems such as Linux. **?** gives the following definition for middleware: "*Middleware is a software layer that provides the infrastructure for integration of applications and data in robotics system*". Different middleware solutions will briefly be discussed and the one best suited for this project will be selected. The chosen middleware solution will then be further elaborated on.

3.2. Middleware

For a distributed control system to preform successfully effective communication between components on the robot, robots in a group and across a network is required (**?**). Modern robots consists of various hardware components, these components may be made by different manufacturers and may use different software and communication mechanisms. Middleware is used to provide a layer between the modular components such as sensors and the different robots in the network.

? discusses the requirements that middleware should fulfil and then gives an overview of the various middleware solutions that are currently available. **?** states that there is no specific middleware solution that solves all problems and that no specific middleware solution is perfect for all robotic systems.

Thus there is more than one middleware solution that will work for this application. A decision has to be made based on some criteria to determine which middleware will be best for the project. **?** did a survey of different middleware solutions. Middleware is called a Robotic Development Environment (RDE), in their study. They evaluated nine different RDEs based on their features, usability and the influence that an RDE has had on the field of

robotics. Based on the performance of each criteria the RDE was awarded a score. The results can be seen in table 3.1. From these results it can be seen that the Player/Stage RDE achieved the highest overall score. The value of the Player/Stage RDE is further validated by **?** which state that Player/Stage is a "*de facto standard in the robotics research community*". **?** evaluated and compared freely available middleware and found that Player was an very good middleware solution. They found that although Player has some latency issues, it is easy to use and supports a great deal of sensors. **?** evaluated middleware solutions for a SSL team based on literature available, he concluded that Player was the most used middleware and thus has great support which aids the development of projects. **?** developed a very basic Player driver and was able to establish a connection over the wireless network, however this was never completely implemented and not used for testing.

Table 3.1: Comparison scores (%) for each RDE adapted from (**?**)

RDE	Features	Usability	Influence	Total Score
TeamBots	28	35	10	27
ARIA	45	53	10	41
Player/Stage	59	82	80	70
Pyro	64	88	15	63
CARMEN	47	59	25	46
MissionLab	60	71	35	59
ADE	74	76	30	67
Miro	53	29	30	42
MARIE	76	50	35	61

? did a study to compare Player/Stage to Microsoft Robotics Studio. They selected these two RDEs because of Player's wide use and Microsoft Robotics Studio's commercial association. They found that both RDEs performed almost equally well, and that the specific application will determine which one is best to be used. They very briefly referred to Robot Operating System (ROS) which is seen as an heir to Player.

ROS has not been extensively reviewed by any of the aforementioned authors. **?** briefly discussed ROS, but still considered Player as a better alternative for the soccer robots due to its wide usage in the robotics community. Brian Gerkey, which is one of the main developers of Player, has been working on ROS since 2009 (**?**). ROS was developed to fulfil perceived gaps in existing middleware. ROS re-uses code from numerous other open-source projects, such as the drivers, navigation system, and simulators from the Player project (**?**). The differences between Player and ROS are explained by Brian Gerkey in **?**. Gerkey explains that Player is designed for non-articulated mobile platforms, whereas ROS is designed for complex mobile manipulation platforms. Player

provides better hardware driver support and ROS offers more implementation of algorithms. It is evident that ROS is more powerful and flexible than Player, however this results in it being more complex.

Considering all the above Player will be used as the middleware for this project. Player is one of the most wildly used and best supported middleware solutions available and because soccer robots are non-articulated Player is ideal for this project.

3.3. Player

3.3.1. Player/Stage

Player is a multi-threaded robot device server (**?**). Player runs on UNIX-like platforms and is released under the GNU General Public License. It provides an interface for robots to access sensors and devices over an IP network using Client/Server communication. Player is the network server, and various clients can communicate with Player over Transmission Control Protocol (TCP) sockets. Stage is a 2D robot simulation software that simulates robots and devices that can be seen by Player as real hardware (**?**).

The newest version is Player 3.0.2 which was released on 28 June 2010, thus when **?** was working developing the first iteration robot Player was still being developed. A lot of literature is available on Player, **?** gives an overview of the original Player and then discusses Player 2.0. **?** develops a new communication architecture based on Player/Stage and tests its performance on the cooperative guidance of robotic units. **?** used Player to develop a method for localizing a team of mobile robots relative to each other. Player allows designers freedom of choice on which programming language they want to use for development of each component. Furthermore, because Player can accommodate such a great deal of clients situated anywhere, an off-field client can be used to do all the computational expensive calculations. Player/Stage is ideal for comparing different controllers due to the fact that it is simple to implement, freely available and actively being developed. Nubotics™ is a range of robotics products designed and manufactured by Noetic Design, Inc (**?**). WheelCommander™ is a closed loop differential drive controller produced by Nubotics™ which supports Player/Stage.

The University of Auckland has a Robotics Research Group uses Player as middleware on all their robots with on-board computers (**?**). The University of Tennessee use Player/Stage as an aid with some of their Control System courses (**?**). Player is also a great platform to test different communication strategies. **?** presents a new communication architecture which was developed using Player.

As mentioned above, Player provides interfaces for devices that are connected over a network. Clients that are connected to Player communicate using

messages over TCP sockets. Player supports multiple clients simultaneously each with their own socket (**?**). An important feature of Player is that because its external interface is TCP sockets, client programs can be written in any language that supports sockets. Client-side libraries are already available for C, C++, Tcl, Java and Python (**?**). Devices are treated as files. Thus in order to read from a sensor device the appropriate read access is acquired to open the device. Similarly in order to control an actuator write access is required.

Player is divided in two sections, the core and the transport layer (**?**). The core layer consists of two libraries, the driver library and the core library. These libraries provide the core Application Programming Interface (API) and functionality for the Player system. The core is a queue-based message passing system; each driver has a single incoming queue and can publish to the queues of other drivers (**?**). The function of the core library is to coordinate the passing of messages and to define the syntax for these messages. The TCP transport layer in Player is currently provided by two libraries, however other alternatives can be used (**?**). This TCP layer consists of two libraries, a library that handles the TCP communication and an External Data Representation (XDR) library that handles the data representation.

The Player architecture that will be used for this project is depicted in figure 3.1. The central controller uses a client to communicate with the robots which have Player drivers on them. In Player a distinction is made between device *drivers* and device *interfaces*. Interfaces specify how to interact with a device and which messages could be sent to the driver. Whereas a driver executes the command that is given in the format the interface specifies. A commonly used interface is the `position2d` interface which is used to control the movement of ground based robots. Drivers are device specific and different types of robots will each have their own drivers to support the required interfaces. The driver and interfaces used for this project will be discussed in section 3.3.2.

The clients are application specific, they are written to fulfil a variety of tasks. The clients use Player proxies to communicate with devices running player servers. The `Position2dProxy` is a class that is used to communicate with robots which support the `position2d` interfaces. The clients and proxies used for this project are explained in section 3.3.3.

3.3.2. Player Drivers

There are two different types of drivers in Player, Static and Plugin. Static drivers are drivers that are distributed with the Player code. These are standard drivers, for example drivers for sensors such as an IMU or a camera. Plugin drivers are custom drivers that are written for a specific robot. They are loaded at runtime and thus the Player package does not need to be recompiled. This section will explain how Player drivers operate.

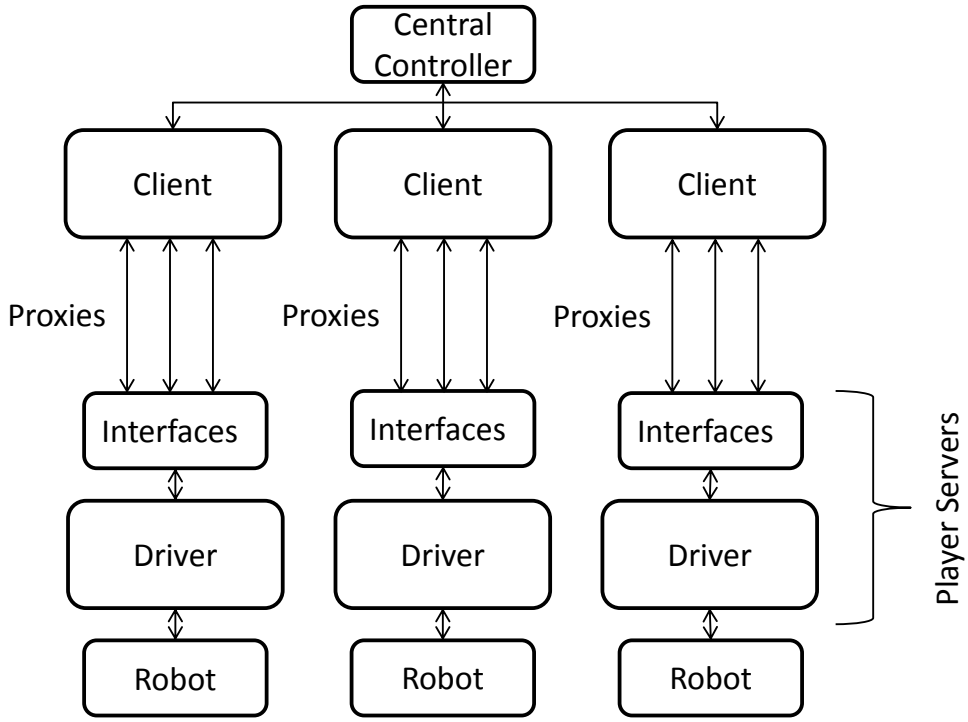


Figure 3.1: Player architecture configuration

Player is a queue based message passing system. All the messages received by the driver is put in a queue from where they are categorized and processed. Therefore to achieve an efficient functioning system the message handling of the drivers is of utmost importance. Player drivers consist of three successive processes that repeat in a loop. They are *ProcessMessages*, *ReadSensors* and *PublishData*, and can be seen in figure 3.2.

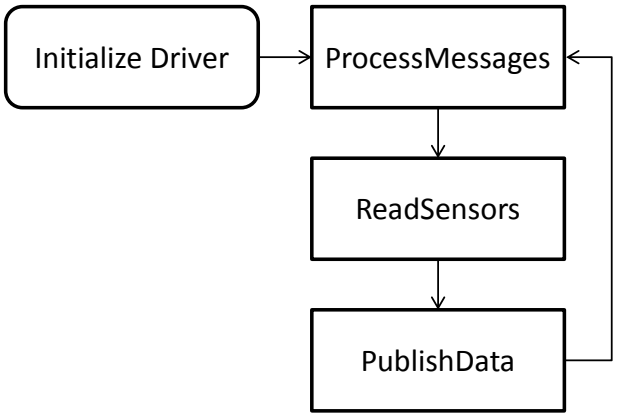


Figure 3.2: The main flow of the Player driver

During the *ProcessMessage* step all incoming messages are processed based

on their message type. Player utilizes five messages types. Only the three main types will be discussed, Data, Command, and Request Messages. Data messages are published asynchronously to the devices. They usually contain data regarding the geometry or the state of the robot. Command messages are also sent asynchronously and are used to change the state of the robot, for example they are used to set the velocity of the robot. Request Messages are different from the previous messages in that they are published synchronously. For every request message sent a response, either Acknowledge (ACK) or a Negatively Acknowledge (NACK) is returned. The Request messages are slower due to the fact that they are sent synchronously and because they require a response from the robot. When a new message is recieved it is processed by die *ProcessMessages* process as seen in figure 3.2. First it is checked to see which interface this message relates to. When the interface is established it is checked to see if it is a Request or Command message. After the message type is established the process correlating to the specific message is executed.

During the *ReadSensor* step all the sensors on the robot are read. The data obtained from all the sensors are stored in variables to be used in calculations or in the *PublishData* step. In the distributed control architecture all the necessary calculations and algorithms are executed in the *ReadSensor* step. The *PublishData* step is only done after the required calculations have been completed. The data that is required by the central controller is published and the whole process start from the beginning again.

Interfaces specify how to interact with a certain type of device. The interfaces define the set of messages that the specific robot can receive. The `position2d` interface specifies that a device which implements this interface can amongst others receive velocity commands and return odometry and geometry data. Player has a large variety of interfaces that can be implemented by a driver. Unfortunately Player does not have all the interfaces required by the soccer robots. There is no interface for the kicker mechanism in the Player codebase. However the interfaces do not have to be used for their intended purposes. The `position2d` interface can receive a command message that gives a command for velocity/heading control mode. This command sends two variables, one specifying the velocity and one specifying the turning angle. The driver can be programmed to use this to set the intensity and angle at which the kicker should kick the ball.

Three interfaces were used in the driver that was written for the soccer robots. Two `position2d` interfaces were used to control and transmit data related to the motor controller circuit and the kicker circuit. The third interface used, is the power interface. The power interface is used to transmit the state of the batteries on the robot. The control of the kicker is beyond the scope of this project, however the functionality required to implement the kicker was written into the driver.

This section discussed the functionality of Player drivers and interfaces. This will be used to implement a new driver for the soccer robots. The imple-

mentation of the motor control and sensor data transmission will be discussed in section 5.4.

3.3.3. PlayerClient

The SSL robot team will consist of six robots each running a Player server consisting of a driver with multiple interfaces. A `PlayerClient` is an object in a program that connects to a Player server on a robot over a TCP socket. The client program is the program that contains a `PlayerClient` object for each of the robots that it wishes to connect to. The program connects to the Player servers using these client objects. Once the connections to all the robots are established, the player proxies can be connected to the interfaces to control the robots. The main program will use these `PlayerClients` to control the team of soccer robots.

The proxies are used to connect to the corresponding interfaces of the drivers on the robot. The proxy sends and receives messages from the interface. The `power` proxy corresponds to the `power` interface and is used to monitor the battery levels. The `position2d` proxy corresponds to the `position2d` interfaces. Thus for every robot the client program needs to contain three proxies. There are client libraries available to access these proxies. The client program for this project was written in C++. Therefore the C++ client libraries was used, the classes in these libraries that were used are the `PlayerCc::ClientProxy`, the `PlayerCc::Position2dProxy` and the `PlayerCc::PowerProxy`. The structure of the client programs will be discussed in detail in section 5.5.

4. Hardware Design

This section describes the hardware that was used for this project. This project continuous on work done by **?** and **?**. The focus of this project was not centred on the hardware and therefore many of the hardware decisions and designs of **?** and **?** were used and will be briefly discussed in this chapter.

4.1. Mechanical Design

? designed the first iteration SSL robot at Stellenbosch University. However, the was not within the specified size limit required by the SSL and did not have a kicking mechanism. Therefore a second iteration SSL robot was designed by a technician at Stellenbosch University. This robot was within the size limits and had a kicker mechanism, the robot can be seen in figure 4.1. The mechanics of the physical robot has a major effect on the performance of the soccer robots. In the production of the soccer robots defects and irregularities are introduced, which result in unpredictable motion when the robot is driving. This is why control software is needed, the software compensates for the non-ideal mechanics. Due to the large effect of the mechanical design on the performance of the robots the mechanical design will briefly be discussed in this section.

The robot has two main mechanical units. The motor unit, which consists of the gears, motors and wheels. The kicker unit which consist of the capacitors, solenoids, dribbler and kicking mechanisms. The kicker unit is used to kick, chip or dribble the ball. This unit will not be used in this project and will not be discussed. The motor unit is used to drive the robot. The SSL robots are omnidirectional, which means that they can drive in any direction without having to turn. Figure 4.2 shows one of the wheels, the robot has four big wheels each with numerous small wheels all around the circumference. The big wheels are driven by the motors and the small wheels rotate freely, enabling the robot to move in any direction. Two spur gears are used to reduce the speed of the motors and increase the torque on the wheels.

Most teams use brushless DC motors which are more powerful (**?**, **?**), however there are teams who used brushed motors (**?**, **?**). The motors that are used are Faulhaber 2342 012 CR brushed DC motors. The motors operate at

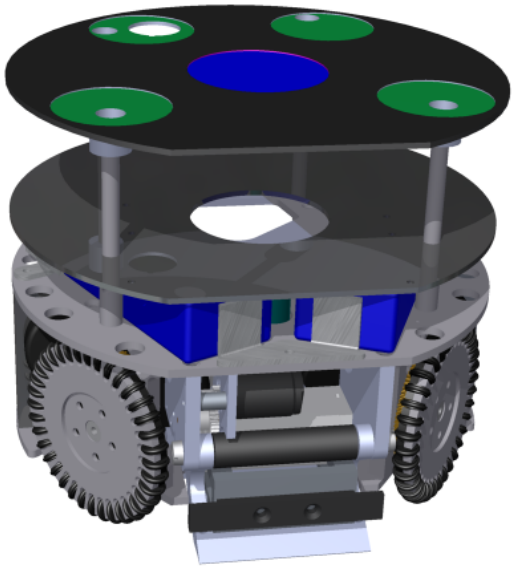


Figure 4.1: CAD design of second iteration soccer robot

Figure 4.2: Omnidirectional wheel

a nominal voltage of 12 V with a maximum output of 17 W, a maximum speed of 7000 rpm and maximum torque of up to 16 mNm. The motors come with Faulhaber IE2-512 magnetic encoders, with 512 lines per revolution, which produce quadrature signal output. A large gear ratio of nine was chosen to ensure that the wheels receive adequate torque. The large gear ratio required that custom gears had to be manufactured for the robots.

The design of the current SSL robot made it susceptible to manufacturing imperfections and irregularities. These manufacturing inaccuracies of the motor unit and the gears result in erratic behaviour of the robot. This resulted in a complication in the control of the robot. The initial axils of the robot's

wheels were not made to the correct dimensions. This resulted in the wheels buckling, which caused damage to the gears. Some of the wheels centres were off-centre thus the contact force between the gears varied constantly, this had a negative influence on the performance of the LLC. Once the wheel hubs had been replaced the performance of the robot improved, although the robust design of the first iteration robot was still better than that of the second iteration robot.

Figure 4.3 illustrates the position of the robots wheels and placement of the the robots coordinate frame. The robot is symmetric with respect to the y-axis. The axes of the two back wheels, in the positive y direction, are 45° from the y-axis. However axes of the wheels on the kicker side are placed at a slightly wider angle to make room for the kicker mechanism. This has a effect on the control of the robot, as it complicates the dynamic and kinematic model of the robot as seen in appendix B.

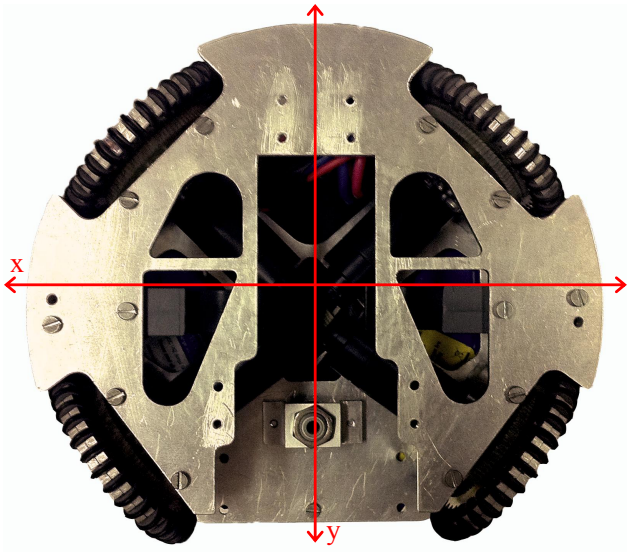


Figure 4.3: Axis of robot and wheel placement

4.2. Electronic Design

The electronic subsystems of the robot are described in the following section. The robot consists of four electronic subsystems; the main controller, the kicker system, the sensor system and the motor control system.

4.2.1. Main Controller

The main controller is the central part of a soccer robot. The main controller does all the higher level calculations and communicates with all the subsys-

tems. **?** found that most SSL teams use FPGA as main controllers. However, **?** stated that a viable alternative is to use a Single Board Computer (SBC). SBC is a small computer on one electronic board, it has most of the functionality of a normal computer such as USB and Ethernet interfaces.

? decided to use a SBC for the soccer robots at Stellenbosch University. This is because the SBC provide higher processing power, enables the use of standard networking communication and provides the ability to program using high-level programming languages such as Java, Python or C++. Five different SBC alternatives were reviewed, all of them were Linux compatible. The selected SBC will need to communicate with the other electronic subsystems, and therefore needs to have Serial Peripheral Interface Bus (SPI) and Inter-Integrated Circuit (I^2C) communication interfaces. The Roboard RB-100 was selected as the best solution due to it fulfilling all the necessary requirements and being small enough to fit in the space constraints of of SSL soccer robots. The Roboard RB-100 can be seen in figure 4.4.

Figure 4.4: Roboard RB-100 for the main controller (**?**)

The Roboard provides much more resources and interface capabilities. It is able to run either Windows or Linux and can be programmed in any high-level programming language. Since the project was started a new version of the Roboard was released, the Roboard RB-110. The only major difference between the RB-100 and RB-110 is that the RB-100 has SPI where the RB-110 has High-Speed serial up to 12 Mbps. SPI communication is required for this project and thus the RB-100 will be used.

Communication between the robot and the central controller is an important task of the main controller, and this is another aspect were this project's soccer robots greatly differ from most SSL teams. The majority of SSL teams use custom radio frequency (RF) communications because it is the easy to implement with a microcontroller or FPGA (**?**). However, because a SBC is being used, Wi-Fi was utilised for the wireless communication. Wi-Fi is a Wireless Local Area Network (WLAN) that uses the IEEE 802.11 standard. A Mini-PCI-WLAN card is used in combination with the Roboard enabling Wi-Fi communication between the robot and any Wi-Fi device. This means

that no special communication units are needed; any computer or device with Wi-Fi is able to communicate with the robots. The Wi-Fi has only one drawback and that is that it is not as reliable as custom RF communication.

The Roboard will communicate with the electronic subsytems on the robot as well as the OFC. The Roboard is able to run Linux and communicates to the central controller using standard Wi-Fi. This makes the system very versatile and well suited for this project.

4.2.2. Motor Controller

The motor controller is used to drive the motors at the desired speed. **?** designed and built four separate motor controllers, one for each wheel of the robot. They used closed-loop control to drive a brushed DC motor at a desired speed. The encoders were used as feedback for the motor controllers. These motor controllers used a PIC 18F2431 microcontroller and VNH2SP30 H-bridge. The magnetic encoder of the motor connects to the motor controller and provides feedback to the PIC. The motor controllers communicate with the main controller using the SPI interface. This design experienced some problems. The motors caused the PIC on the controllers to reset (**?**).

The motor controllers were redesigned by (**?**). The redesigned circuit used the same components as the previous design. More connectors were added to enable additional sensors to be connected to the board. Furthermore instead of using four separate motor controllers the new design uses a single board. All four motor controllers were placed on one circuit board, as seen in figure 4.5. This reduced the physical connections on the robot. The I^2C connection on the Roboard goes directly to the new motor controller and sensors that require the I^2C interface needs to access it from the new board.

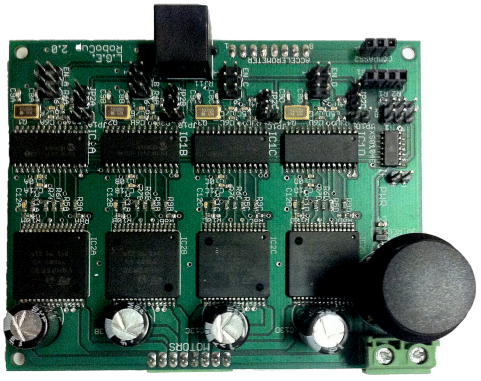


Figure 4.5: Motor controller circuit

4.2.3. Translation and Rotation Sensors

The robots require on-board sensors to derive information regarding their environment. Various sensors have been used in the RoboCup SSL. The most common are rotary encoders which determine the motors' rotational speeds. However, the current availability of sensors with embedded controllers has led to SSL teams using additional sensors. This project focusses on achieving accurate motion control of a soccer robot. Therefore only sensors relating to the robot's motion will be discussed.

The translation and rotation of a device can be measured using combinations of various sensors. These include accelerometers, gyroscopes, electronic compasses and mouse sensors. A mouse sensor directly provides the in-plane velocity. The disadvantage with a mouse sensor is that they have very small positional tolerances. This complicates the mechanical design of the sensor unit. A mechanical and electronic design was done to implement a mouse sensor on the robots at Stellenbosch University. The design was never implemented and is discussed in appendix C. This was because the sensor was deemed unsuitable for its purpose. **?** implemented a mouse sensor with SSL robots. They found that the robot's rotation had an influence on the measured velocity, which resulted in inaccurate measurements.

Another sensor which can be used to derive information regarding the translational movement of a robot is an accelerometer. Accelerometers measure the in-plane acceleration, which can be integrated over time to determine the velocity and displacement. **?** and **?** used accelerometers in their SSL soccer robots, although they did not present sufficient test data. Most MSL teams use accelerometers and an accelerometer will be used in this project.

? suggested that a compass be used to determine the orientation of the robot. However, compasses are sensitive to electromagnetic noise. **?** implemented an electronic compass on their robot. The information obtained from the electronic compass was not correct as a result of electromagnetic interference from other devices such as the solenoid. Therefore it was decided not to use a compass due to the significant electromagnetic forces induced by the kicker system. The encoders on the wheels could be used to determine the rotation of the robot, however due to wheel slippage this is not accurate. Instead a gyroscope was chosen to determine the orientation of the robot. **?**, **?** and **?** used gyroscopes to determine the rotation rate of their soccer robots. Due to the wide usage and suitability a gyroscope will be incorporated into the robots.

An IMU is a sensor that consists of accelerometers, gyroscopes and/or a compass. An IMU was used instead of separate acceleration and gyroscope sensors to conserve space and simplify the design. **?** used an IMU consisting of a single degree of freedom gyroscope and two Degrees of Freedom (DOF) accelerometer. A wide variety of IMU sensors are available, and for this project an IMU with a three-axis accelerometer and one-axis gyroscope was required.

The accelerometer's two horizontal axes will is used to measure the translational movement, while the vertical axis can be used to calibrate and position the IMU. The most suitable IMU available was a six DOF IMU and can be seen in figure 4.6. It consists out of a three-axis accelerometer (ADXL345) and a three-axis gyroscope (ITG-3200). The IMU uses an I^2C interface and an interrupt pin for the accelerometer and gyroscope is available. The IMU is very small and was installed on the robot so that the axis of the robot aligns with the axis of the IMU.

Figure 4.6: 6 degrees of freedom IMU (**?**)

5. Software Design

The largest portion of the author's time was spent on developing and optimising software for this project. The chapter starts by showing the modifications made to the motor controller's software. Then software is developed which implements motion control algorithms on the robot. This is divided in two sections; velocity profile generation and trajectory tracking. The software developed to implement the Player drivers and the Player clients are shown. Finally, the software aspects relating to the on-board sensors are discussed.

5.1. Motor Control

The motor controller was developed concurrently with this project. All four of the motors are driven by the same motor controller circuit. The performance of the robot is dependent on effective and correct implementation of the motor controller. The controller needs to accept a velocity command for a specific wheel and ensure the proper execution of that command using PID control. The LLC will be implemented using the motor controller, therefore the encoder data should also be available to enable testing and development.

? designed the motor controller to operate in a daisy chain. This means all the individual motor controllers are connected in series and a command is sent to the first one which passes it on to the next controller and so on till it reaches the last controller from where it is sent to the main controller again. This reduces the number of connections between the main controller and the motor controllers, but has the disadvantage that the main controller needs to send four commands every time it wants to do something. The motor controllers are programmed so that each command is sent with an identifier to indicate which motor it is intended for. The main controller can also send an encoder data request after which the motor controllers return the desired information.

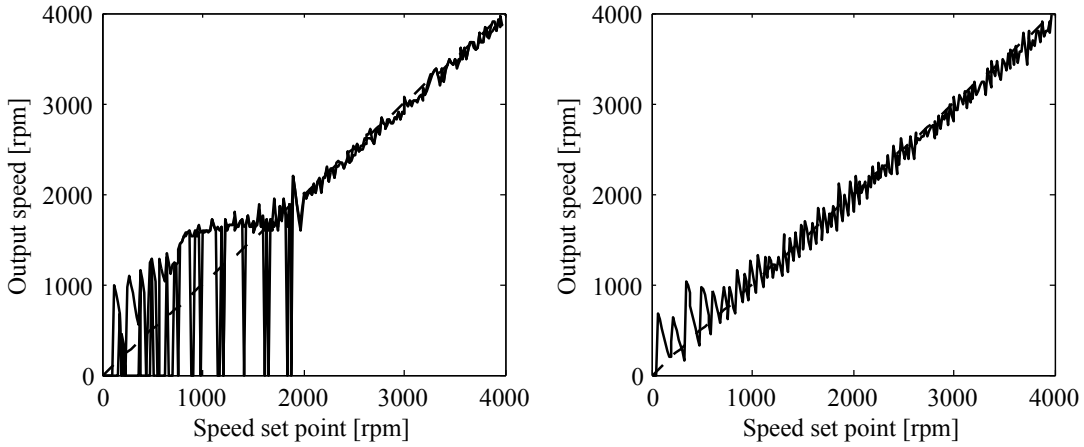
Unfortunately **?** did not have sufficient time to test or finish the software for the controller. There were still minor errors in the software that needed to be developed further and tested to verify that the controller would perform adequately to enable it to be used for this project. Only major changes in the

structure of the code will be mentioned in this section, as well as the calibration of the PID control.

PID control is necessary to ensure that the wheels turn at the desired speed. The job of the PID is to compensate for disturbances in the system, due to varying torque on the wheels and thus ensuring that the correct speed is maintained. This is done by using closed-loop feedback where the magnetic encoders of the motors are used as feedback.

The first problem encountered was as a result of the encoders no longer being triggered when the wheel stopped abruptly. Due to a sharp increase in torque or due to mechanical imperfections at low speeds the wheel stopped turning. This resulted in that the speed at which it was running at before it was stopped will stay in the speed register. This results in the PID controller thinking that the motor is running at the correct speed while it is actually stationary.

? programmed the motor controllers with the intention of avoiding such situations. However, this did not work as seen in figure 5.1a. For any set speed below 2000 Revolutions per Minute (rpm) the output assumed any arbitrary value. This meant when the robot was accelerating from standstill it behaved in an unpredictable manner.



(a) Initial velocity profile

(b) Improved velocity profile

Figure 5.1: Velocity profile of motor controller

The motors are required to operate at speeds between 0 and 4000 rpm. As a result of the fast paced nature of the SSL games the robots are constantly accelerating from standstill. Therefore the low velocities are especially important. This meant that another means of detecting whether or not the motor has stopped was required, in order to avoid the arbitrary behaviour illustrated in 5.1a. It was decided to use a timer to determine how long it has been since

an encoder interrupt had occurred. The encoders are set up to trigger an interrupt 16 times a revolution. If the time since an encoder interrupt has occurred reaches a certain value the PID controller is notified that the wheel is no longer rotating and the PID can compensate for this. The new and improved velocity profile is seen in figure 5.1b and, although the controller is badly, tuned the controller behaves as expected. This graph illustrates that the controller is able to detect a static motor and compensate for this.

The oscillation in 5.1b is as a result of the poor performance of the PID algorithm. If only proportional and integral gain is used as tuning parameters the control equation reduces to:

$$U(t) = K_p e(t) + K_i \int e(\tau)d\tau \tag{5.1}$$

The error has to be integrated to calculate the correct output of the controller as seen in the second term in equation 5.1. The PID controller of **?**, calculated the integral term by adding the error each time the controller output was calculated as:

$$\int e(\tau)d\tau = \sum e \ . \tag{5.2}$$

This is sufficient for high speeds when the dynamics of the mechanical system is fast. However, at lower motor speeds the integral term increases at a much higher rate when compared to the mechanical system's dynamics. This results in oscillations in the system as seen with PI_1 in figure 5.2.

An improved way to calculate the integral term is to use a crude form of numerical integration, as seen in this equation,

$$\int e(\tau)d\tau = \sum e\Delta t \tag{5.3}$$

Here Δt is a fixed time step at which the integration occurs. Using equation 5.3 the integration happens at a fixed rate and represents the continuous integration better than equation 5.2. The result can be seen in figure 5.2, the set point is labelled SP.

The previous step response is labelled as PI_1 in figure 5.2, the controller had a large overshoot reaching the desired value only after about 375 ms. This causes a problem when a dynamic velocity profile needs to be followed. If a new velocity command is given every $\Delta t_{command}$ seconds with $\Delta t_{command} << 375\ ms$ there will not be enough time between consecutive velocity commands for the controller to reach the set point velocity. The result is that the controller will not be able to follow the desired velocity profile.

Therefore the PID of the motor controller was reprogrammed and recalibrated which yielded the step response labelled PI_2, the derivative parameter was selected as zero and thus it became a PI controller. The optimal PI parameters were selected as P = 1.1 and I = 2. The performance of the new motor

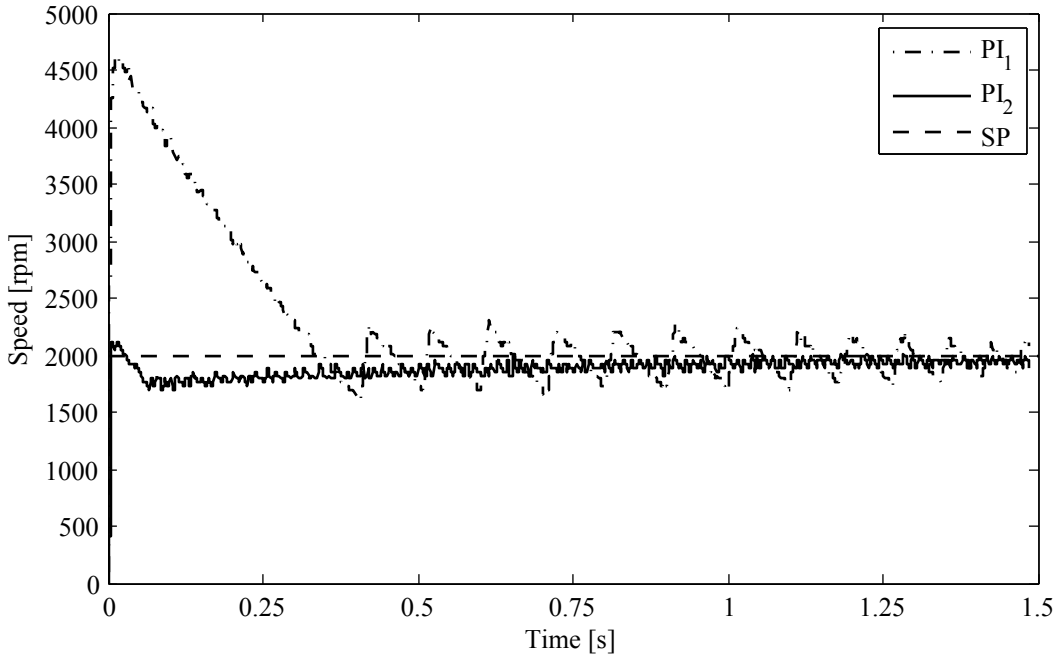


Figure 5.2: PID step response, with derivative parameter D = 0

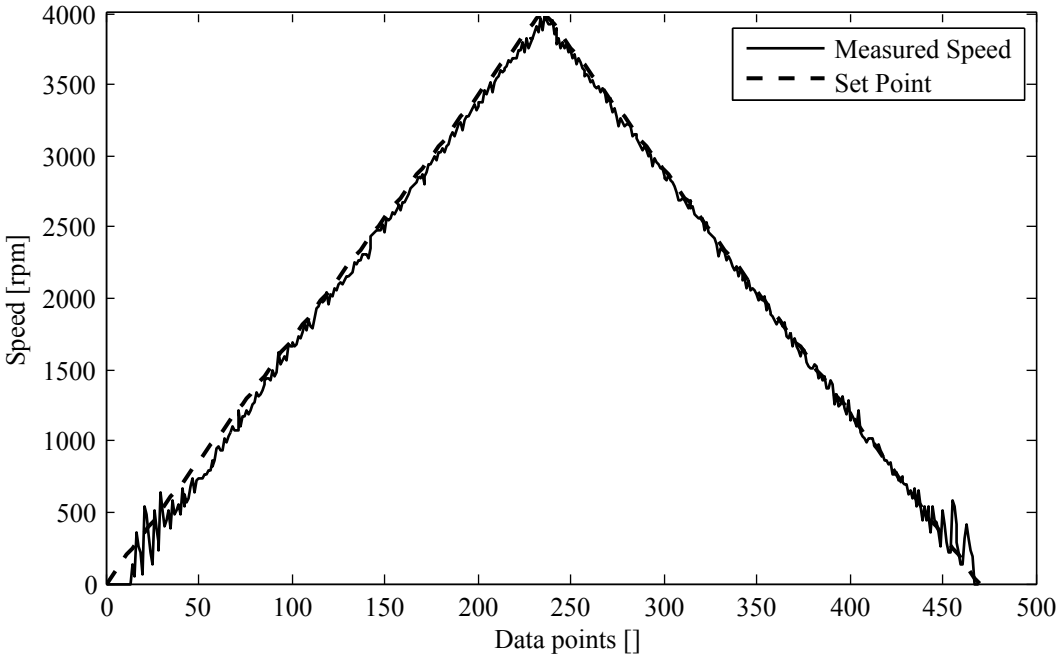


Figure 5.3: Velocity profile of motor controller after the PID controller was tuned

control system is demonstrated in figure 5.3. The controller was sent a series of speed commands and the output was recorded. The straight triangular line is the desired set point speed. The controller follows the desired speed with a slight error at the beginning. This is as a result of the controller not having time to reach the steady state speed before a new command is issued. The integrated error is not reset to zero with each new command, as one would do with a new step input. Instead the integrated error is only set to zero if the new velocity command is not within 30% of the previous command. This is done to ensure that the integral term is only reset for new step inputs.

It can be seen from figure 5.3 that there is still some oscillation below 500 rpm. This is as a result of the test being done while there was no load on the wheels. The motors are not designed to operate at such low voltages that result in those speeds when there is no load present.

The motor controller was debugged and tested and performs well enough to drive a specified motor at a desired speed. The robots are now able to receive and execute velocity commands. However as a result of the way on which the daisy chain is implemented the delay between when the motors receive their commands is four times longer than it could be be changing the software implementation (**?**). This was beyond the scope of this project and thus it was decided not to completely reprogram the motor controller. The performance of the motor controller regarding the implementation of a LLC will be discussed later in this project.

5.2. Motion Control

5.2.1. Background

This section discusses the motion control of omni-directional robots. Motion control is the combined task of the HLC and the LLC. Trajectory tracking control is required in a dynamic environment in order to get a robot from a certain point to a specified destination while avoiding obstacles. Trajectory tracking comprises of two tasks, path planning and trajectory following (**?**). Path planning consists of planning a geometrical path and velocity profile which the robot should follow to ensure that it avoids any obstacles, while operating within its physical constraints (**?**).

In order to perform different tests with robots a trajectory plan is required. Because this project focusses only on one robot, instead of the whole team, the obstacle avoidance is not required and only the velocity profile is needed. The section will first look at the task of generating a velocity profile which ensures that the robot stays within its physical constraints. After this the task of controlling the robot to follow the desired trajectory will be discussed. Although the location of the HLC is a fundamental part of this research, the type of controller used is not a major focus and it will only be discussed briefly.

The SSL robots are very manoeuvrable. As a result of advances in technology the robots are not restricted by the amount of power they are able to produce, but by limited friction at the wheels. This can be seen in that robots have enough power for slippage to occur even at high velocities (**?**) (**?**). A motion control method is required that ensures that slippage of the wheels does not occur.

Most motion control techniques are based either on the robot's dynamic model (**?**) or the kinematic model (**?**) (**?**). The kinematics of a robot is purely related to the geometry of a robot and the influence thereof upon the motion of the robot whereas the dynamics relates the forces acting upon the robot to its motion (**?**).

? and **?** decoupled the translational and rotational motions, enabling the use of a separate controller for each degree of freedom. **?** developed a system which use two identical PID controllers to control the robots position and orientation. Control strategies have been developed and tested that do position control without orientation control (**?**). **?** looked at kinematic and dynamic modelling of omni-directional robots in combination with velocity and acceleration filtering to ensure that slippage does not occur. **?** proposed a new control method based on the inverse input-output linearized kinematic of the robot. The control method of **?** took the translation and rotation of the robot into account.

? discusses four different control strategies for omnidirectional robots with three lateral orthogonal-wheels. These strategies are resolved acceleration control method, PID method, fuzzy model method and stochastic fuzzy servo method. The PID method is modified to create an adaptive PI controller. The only way to select the appropriate PI values is experimentally by trial and error. Thus **?** suggest using an adaptive method where the P and I gains are adjusted according to the error signal. **?** developed a LLC using a discrete-time linear quadratic tracking approach, while the robot's dynamics is incorporated in combination with fuzzy logic as a HLC .

? designed a control using a Trajectory Linearisation Control (TLC) method based on a nonlinear dynamic model of the robots. **?** was the only authors found to implement sensor fusion, using a Kalman Filter (KF), as part of the robot control. The motor's encoder data was combined with data from the vision system to formulate and accurate estimate of the robot's state. The LLC of **?** consisted of two separate parts. The first part controlled each wheel separately, using the encoder information as feedback. **?** used an on-board IMU sensor to provide additional information regarding the motion of the robot. This information was fed into the second part of the LLC. This system is illustrated in figure 2.1.

This section starts by discussing the algorithm implemented to perform the path planning. After that the implementation of the trajectory following will be discussed.

5.2.2. Velocity Profile

For the robots to accurately perform basic movements without slippage, velocity profiles that restrict the robot's acceleration are required. No new research was done on this field and one of the existing methods was selected and used in this project.

A great deal of research has been done in this field regarding omni-directional robots as mentioned in section 5.2.1. The algorithm developed by **?** was developed and tested using SSL soccer robots. The algorithm is intuitive and easy to understand. Example code was available in MATLAB®. Therefore, because there was sample code available and the algorithm was easy to understand it was decided to use this algorithm for the project.

The algorithm has two limitations; the first is that the final velocity of the robot is always zero to avoid discontinuities in the solution. However, this is not necessarily a problem due to the fact that the end destination will change constantly (**?**). The second limitation is that the algorithm does not compute the rotational velocity profile of the robot. This was deemed acceptable because this aspect will not be required for the tests that will be performed for the current project, but will be needed in future development.

There are two physical limiting factors for the soccer robots. The first is the limited friction between the wheels and the operating surface and the second is the top speed of the robots. The goal of the algorithm of **?** is to compute a trajectory to move a robot from an initial state to another in the shortest amount of time, while taking limited friction and weight transfer into consideration.

The acceleration envelope needs to be determined to implement this algorithm. The acceleration envelope is defined as the boundary of the set of feasible combinations of acceleration ($\ddot{x}, \ddot{y}$ and $\ddot{\theta}$) (**?**). Any combination of $\ddot{x}, \ddot{y}$ and $\ddot{\theta}$ that lie within the acceleration envelope can be performed without slippage occurring. The acceleration envelope is not rotationally symmetric and simplification needs to be done to make it computationally easy to solve. The acceleration envelope is derived from the dynamic model of the soccer robots. The envelope is restricted to make it independent of the robot's orientation. The envelope is further simplified by decoupling the translational and rotational DOF. An example acceleration envelope is seen in figure 5.4. It is a cylinder with a radius a_{max}.

The robot's maximum acceleration a_{max} is the radius of the cylindrical envelope in figure 5.4 and its maximum velocity v_{max} is defined by the motor specifications. The algorithm is based on the fact that in order for the robot to reach its destination in the minimum time it will either be accelerating at a_{max} or moving at v_{max}. All the possible optimal control scenarios are discussed in appendix F.

The algorithm starts with the initial conditions and based on how far the robot is from the final destination and the initial velocity it determines which

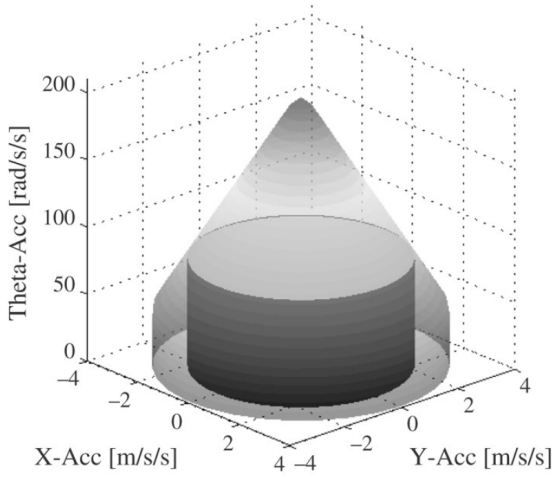


Figure 5.4: Cylindrical acceleration envelope (**?**)

control scenarios should be followed. This is done for movement in both the x and y direction. Both x and y trajectories are calculated, and they will both have different execution times. This is because either the x or y distances might be shorter and thus require a shorter time to reach the destination. However, these two solutions need to be synchronised in order to calculate the minimum time required to reach the destination.

The synchronisation is done by using a scaling factor (α) and trigonometric identities. The value of α is determined using the iterative bisection algorithm. This factor scales a_{max} and v_{max} in order to minimize the the required time in the one direction, while prolonging the time in the other direction. This is done to ensure velocity profile for the x and y directions take exactly the same time to complete. Using the scaling factor the new values for a_{max} and v_{max} are,

$$\begin{aligned} a_{x,max} &= a_{max}cos(\alpha), \quad v_{x,max} = v_{max}cos(\alpha) \\ a_{y,max} &= a_{max}sin(\alpha), \quad v_{y,max} = v_{max}sin(\alpha) \ . \end{aligned} \tag{5.4}$$

Because $cos(\alpha)^2 + sin(\alpha)^2 = 1$ the maximum velocity and acceleration constraints are not violated, as seen in the following equations:

$$\begin{aligned} \sqrt{a_{x,max}^2 + a_{y,max}^2} &\leq a_{max} \\ \sqrt{v_{x,max}^2 + v_{y,max}^2} &\leq v_{max} \ . \end{aligned} \tag{5.5}$$

The algorithm of **?** used a fixed 16.67 ms time step to store the velocity values and the corresponding times in two arrays. Thus if a manoeuvre took three seconds there would be two arrays of 180 elements each. It was decided that an alternative method should be used as these large arrays would result in degradation in the performance of the algorithm. The velocity profiles are

always linear, this means the velocity profile can be described by the three linear equations and three time values seen in figure 5.5. Therefore only the nine variables are required no matter the duration of the manoeuvre, resulting in a significant reduction in memory requirements.

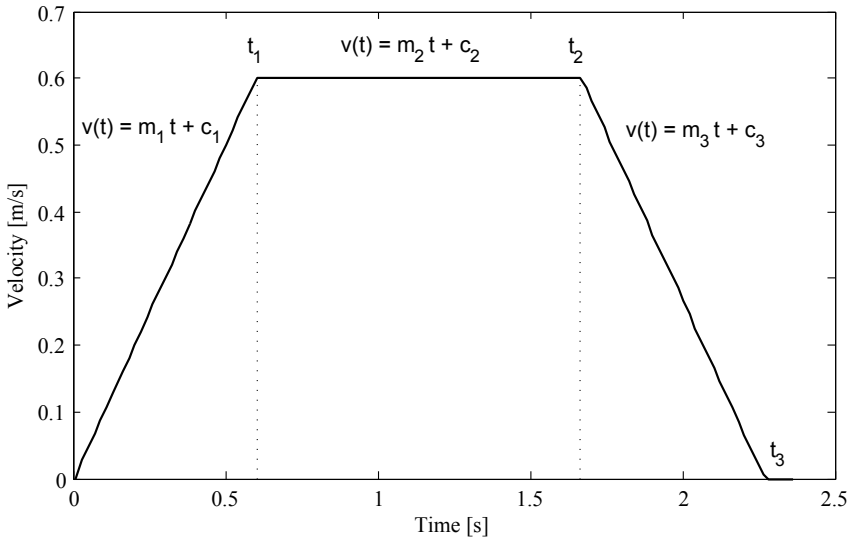


Figure 5.5: Velocity Profile

The algorithm performed satisfactory when tested using MATLAB®. It was then implemented on the soccer robots using C++. Once the algorithm was implemented on the soccer robots the robots could be used for further development to accomplish the objectives of this project.

5.2.3. Trajectory following

Trajectory tracking is the procedure of ensuring that the robot follows a specified trajectory. Various techniques of trajectory tracking were discussed in section 5.2.1. The path planning method of **?** decouples the translation and rotational motions from each other. Therefore the trajectory following will be performed using separate controllers for each degree of freedom.

The LLC will be similar to that of **?**. A separate PI controller will be used for each motor on the robot. The encoder's output is used as feedback for each controller. **?** use torque control instead of velocity control for the wheel controllers. Although the torque controllers yield better results they are more difficult to implement. **?** used separate controllers for each wheel based on velocity control. They had a HLC consisting of three additional controllers which used the feedback of the encoders to control the three DOF. However, slippage resulted in inaccurate feedback resulting in poor performance. The LLC will implemented using only one controller per wheel using velocity based PID control. The controller can be seen on figure 5.6.

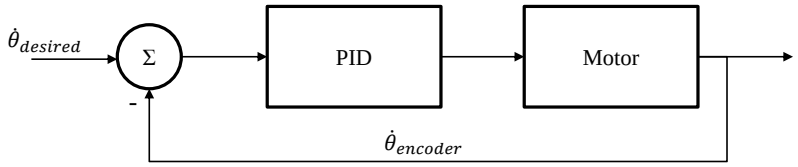


Figure 5.6: Motor PI controller

The HLC of **?** was implemented using a PI-fuzzy controller. Part of the HLC is situated on the robot while the rest is situated in the OFC. Figure 5.7 shows the control architecture used by **?**. The path planner with PI controller and the velocity filer is on the OFC. The fuzzy controller is situated on the robot.

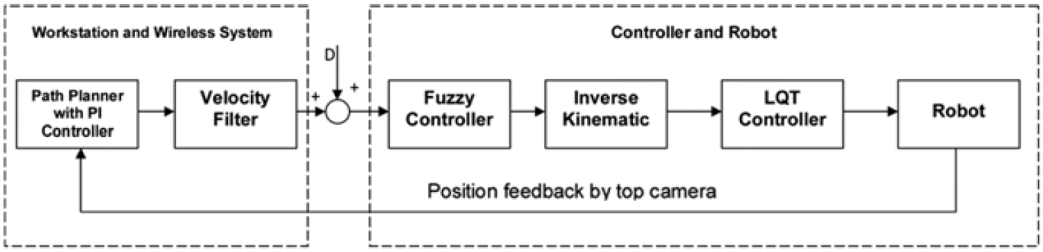


Figure 5.7: Control architecture of **?**

For the proposed system the HLC will be completely situated on the robot. This project was not interested with developing a new type of controller and therefore only a normal PID controller was used. With this new system the occurrence of slippage or random disturbances will immediately be sensed and corrected for. As a result of the HLC being situated on the robot the control loop will be much faster resulting in better control.

5.3. Vision System

The SSL-Vision system is the global vision system used by all the teams in the RoboCup SSL. This shared vision system has been used since 2010. This is done to ensure a more competitive environment and this also greatly reduces the setup and testing time at competitions. The system is developed by volunteers from participating SSL teams (**?**).

The SSL-vision application encompasses all the functionality required of the vision system. This is achieved by using a multi-threading approach. The SSL uses two cameras, one for each half of the field. The vision system supports concurrent image processing for both cameras in the same application by

having a dedicated thread for each. The system supports IEEE 1394/DCAM cameras, which includes 1394b cameras. All the standard DCAM parameters can be adjusted and seen in the main Graphical User Interface (GUI) application. DCAM parameters are for example the shutter speed, white-balance, exposure and gain of the camera.

The robots in a SSL team all have colour markers to identify each robot. The robots have a center marker which is either blue or yellow to indicate which team they belong to. They then have colour markers around this marker to uniquely identify each robot and its orientation. The SSL-Vision system does the image capturing, colour thresholding, region extraction and all the necessary processes to determine the location and orientation of each robot and the position of the ball.

The colour segmentation of the SSL-Vision system is based on software of Carnegie Mellon University. The colour thresholding is done using a Lookup Table (LUT) that maps the input image's 3D YUV colour space to unique colour labels (**?**). These unique colour labels represent the ball, robot colour markers and other identifiable objects's colours. The SSL-Vision system allows the LUT to be generated in the main application GUI by using the incoming video stream.

The main GUI is able to fulfil most of the vision system tasks. One of these tasks is camera calibration which is also done in the main GUI. This is done to establish a relationship between the field geometry and the image plane. The calibration procedure is straight forward and no additional calibration objects are required. The image processing results are transmitted to the participating teams via UDP Multicast (**?**). The client programs can receive the packets on port 10002 and multicast address 224.5.23.2. Data packets that include the timestamp, frame number and positions, orientations and confidences of all the detected objects are encoded using Google Protocol Buffers (**?**) (**?**). No sensor merging is performed by the vision software. Thus the data received from the two cameras have to be merged by the individual teams. However, this project will not be doing the sensor merging. Only one half of the field will be used because testing will only be done on one robot and not an entire team of robots. The camera detection packet is defined by `messages_robocup_ssl_detection.proto` which contains the complete detection results. This is used by the client program which will be discussed in section 5.5.

5.4. Driver Program

The driver programs provide the central controller with access to the robots. Player drivers were discussed in section 3.3.2. The section explains how the driver provides access to the physical hardware of the robot. This is done through messages that are sent and received between the driver and the client.

The driver uses interfaces to communicate with the client's proxies. The driver has two major tasks, the first is to provide access to the soccer robot's hardware. This is sufficient for a centralised control system where all the computational work is done on a central controller. The second objective of the project is to develop a distributed control system which implements sensor fusion on the robot. This is the second purpose of the driver, to perform tasks required of the robot in a distributed control architecture.

Interfaces were discussed in section 3.3.2, this section elaborates on interfaces and discusses the implementation thereof. An instance of the `position2d` interface is used to control the motors of the robot. The pose and the physical geometry of the robot is the only two request messages used by the `position2d` interface. This enables new clients connecting to the robot to establish were the robot is situated and its physical size. In the SSL all the robots have to be within a certain size, thus the geometry information is unnecessary however the driver is written so that other useful information regarding the robot can be sent using the geometry request messages.

The `position2d` interface has four main command messages, they are the position, velocity, carlike and vel/head commands. The position and velocity commands are used for their intended purposes. When the velocity command is received the robot's velocity is set to the desired speed. The position command receives a new desired position for the robot. The robot then moves to this desired position. The carlike command was intended to control the speed and turning angle of the robot and the vel/head command was intended to be used to control the speed and angular position of the robot. The carlike and vel/head commands are not used in this manner and are used for other purposes. The vel/head command could for example be used to send information regarding the position of the ball to the robot.

The command messages that will be used in this project and their purposes can be seen in table 5.1. The $\texttt{position2d}_{\texttt{m}}$ interface is mostly used for controlling the motor's position. Whereas the $\texttt{position2d}_{\texttt{k}}$ interface is intended to be used to control the kicker, however some of its functionality will be used for other purposes. The coordinates of the robot's position, determined by the vision system, will be transferred to the robot via the velocity command of the $\texttt{position2d}_{\texttt{k}}$ interface. Data messages could have been used instead of a command messages to send the data. However, data messages require an ACK message to be sent back, this makes the process slow. Therefore a command message was used because of its speed and because some loss in transmission is acceptable.

The driver needs to implement all the functionality required by the distributed control architecture. This is done by custom classes written in C++. The classes and the names of the instances of each class that is used in the driver is listed in table 5.2. The most important classes will be discussed briefly. The `RoboDriver` class uses SPI communication to communicate with the motor controller circuit. Individual motor velocities or global robot veloci-

Table 5.1: Interface command messages

Interface	Command	Variables	Purpose
`position2d`$_m$	velocity	$\dot{x}\ \dot{y}\ \dot{\theta}$	Send a translational and rotational velocity command to the robot
`position2d`$_m$	position	$x\ y\ \theta$	Send a desired destination command to the robot
`position2d`$_k$	velocity	$x\ y\ \theta$	Send the camera data to the robot

ties can be sent using this class. It is also able to access the encoder data from the motor controllers.

Table 5.2: Custom classes used in the Player drivers

Class	Instances	Purpose
`RoboDriver`	mtrCntrl	Communicates with the motor controller circuit. Sends velocity commands to the wheels
`IMU`	imu	Initializes the IMU sensor and reads all the IMU data
`Controller`	xcntrl	Implements the state estimator in the x direction
	ycntrl	Implements the state estimator in the y direction
`AngleKalman`	acntrl	Implements the angular state estimator
`PID`	xPID	Implements the trajectory tracking control in the x direction
	yPID	Implements the trajectory tracking control in the y direction
	anglePID	Implements angular trajectory tracking control
`DTime`	driveTime	Determines the sampling time for the estimator algorithms
`MotionPlan`	mp	Generates the minimum time velocity profile

The `IMU` class is used to initialize the I^2C communication with the IMU sensor which is discussed in section 4.2.3. This class returns the acceleration and rotational data measured by the sensor. The `Controller` and `AngleKalman` classes implement the estimator algorithms discussed in Chapter 6. The `Controller` class implements the Kalman filter algorithm of equations 6.3 to 6.5. Every time new IMU or camera data is available the state estimate is updated accordingly. The `AngleKalman` class is very similar and implements the estimator discussed in section 6.3.

The `MotionPlan` class implements the algorithm discussed in detail in section 5.2.2. It generates the new velocity profile and returns the desired speed for a specified time. The `PID` class implements the Trajectory Tracking control which was discussed in section 5.2.3. The `DTime` class is used as a timer. This class was used to track how far the robot was into a manoeuvre, as well as the sampling times for certain algorithms.

The main function of the driver in the distributed control architecture is the `ProcessPos2dPosCmd()` function. This function receives a desired position command from the central controller. Figure 5.8 depicts a flow chart of this function. The total duration of one cycle of this function is between 13 to 22 ms depending on the decisions made. This function generates a velocity profile that the robot needs to follow to get to the desired position. This profile is then followed using trajectory tracking as explained in section 5.2.3. The subroutine labelled *Process new incoming messages* checks for new messages from the client. These include new commands or information. One example of this is a message containing camera data. When this message is received the `ProcessCameraData()` function is called. This function updates the state estimate based on the camera data received.

This section briefly discussed the classes used in the Player driver written for the SSL soccer robots at Stellenbosch University. This driver could either be used in a centralised or distributed control system. It depends on the messages it receives from the central controller. It is able to receive and process commands and can communicate with hardware on the robot, such as the motor controller and the IMU sensor.

5.5. Client Program

The client program is the software that is used to connect the central controller to all the robots on the field. The basic function of `PlayerClients` were discussed in section 3.3.3. The client program connects the player proxies to the interfaces on the robots so that messages can be sent between the drivers and the central controller.

The client program has four main classes. They are `Control`, `Vision`, `MotionPlan` and `Robot`. The `MotionPlan` class is used to calculate velocity profiles for moving a robot in a centralised control architecture. The algorithm implemented by this class is discussed in section 5.2.2. The `Vision` class receives the vision packages that are sent by the SSL-Vision system. Instances of this class can be used to access the vision information.

The `Robot` class is used to communicate with a robot. For each robot an instance of the `Robot` class is required. This class communicates directly with the driver on the robot. The connection to the robot is established using the `PlayerClient` object in this class, then the proxies are used to communicate with the interfaces. The proxies that are used are briefly discussed in section

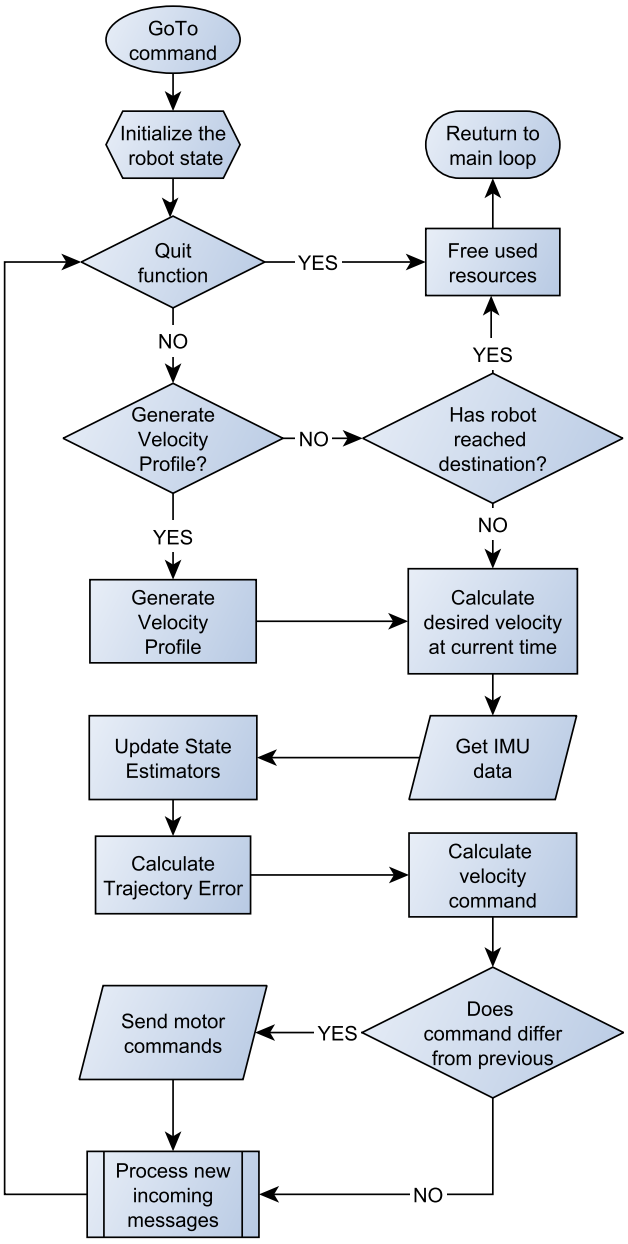


Figure 5.8: Flowchart of `ProcessPos2dPosCmd()` function

3.3.3. The commands that the robots can be sent are defined in this class. The `Robot` class provides all the Player functionality required for the soccer robots, but hides the technical details. Therefore no knowledge of Player is required to control the soccer robots when using this class.

The `Control` class is used to control a team of soccer robots. This combines the `Vision` and the `Robot` class to create a class that can be used to control the whole team of robots. Although this project only does testing on one robot,

the software was designed in such a way that it can be used for further research with multiple robots. During the development of this project various versions of the `Control` class were developed. The first versions were Command-Line Interface (CLI) and commands were entered using the keyboard. No GUI was used. This was used to develop the client software using Stage at first. Thereafter further CLI versions were used to test and develop the drivers on the robots.

The SSL-Vision software contains a GUI vision client which is used to display the robots and the field on the computer screen. This software was distributed under the GNU General Public License (GPL) v3. The software was modified and an instance of the `Control` class was added to use this GUI to control the robots. This is still a very basic program and is used to perform the testing of the robot functionality in a distributed and centralised architecture. The programs are used to move the robot around the field. The robot is moved by clicking on a certain point on the field and the robot will drive towards that point.

The control program mentioned above is used to test the soccer robots. Figure 5.9 illustrates a typical SSL software architecture. This client program will be used in the Control Module. After the navigation has been done the commands will be sent to the Communication Server using the client program.

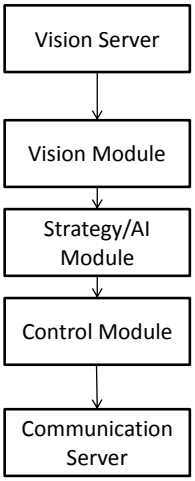


Figure 5.9: Software architecture

5.6. Sensors

This section discusses the software implementation of the on-board sensors. Each subsystem has its own coordinate frame. This section discusses the trans-

formation of the data obtained from these subsystems to a reference coordinate frame. The process of calibrating IMU will be explained.

5.6.1. Coordinate System Transformations

The various subsystems each have their own coordinate frame. In order to control the robot these coordinate frames need to be transformed to a single reference frame. The coordinate frames of the motor controller unit, IMU and camera is illustrated in figure 5.10. The coordinate frame of the camera is stationary as the camera is fixed with respect to the field. The robot's local coordinate frame constantly moves with respect to the global coordinate frame.

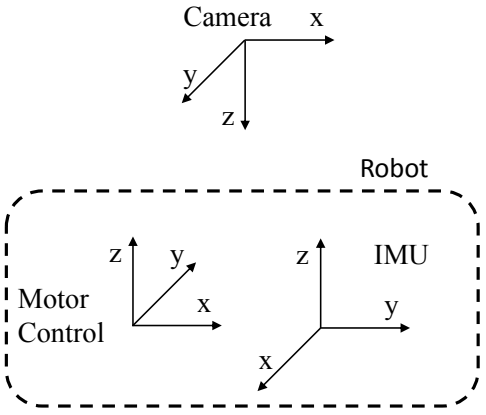


Figure 5.10: Reference frames of the robot, camera and IMU

The system's reference frame can be selected either as a global coordinate frame or a local coordinate frame. When the a local reference frame is selected, each robot has its own reference frame. Thus the local coordinate frames on the robot will stay stationary and the camera's coordinate frame will need to be transformed to that of the local reference frame. The second option is to transform all the local coordinate frames to a global reference frame.

The first approach was to use the local coordinate frame as a reference frame. This was done because the camera is the only global subsystem and the data rate of the vision system is lower than that of the on-board sensors. This means fewer transformations are required. However, this approach did not work. Take the example of a robot moving a distance d in the global x direction. During the translation the robot rotates counter-clockwise by 90 degrees for half of the manoeuvre and then rotates clockwise by 90 degrees so that when it reaches its destination it is in its original orientation.

Now assume the IMU was able to measure the acceleration at all five of the orientations that are shown in figure 5.11. Also assume the camera was able to observe only the first and last orientation. If the camera data is transformed to the robot's coordinate frame it signifies a movement of distance d in the

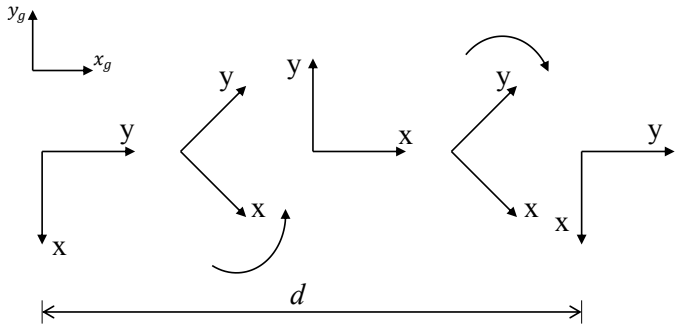


Figure 5.11: Rotation and translation of a robot

robot's y-direction. However, the IMU will indicate a movement in the x and y direction. Thus although all the data was converted to the same reference frames; the data indicate different movements of the robot. This eliminates the possibility of using this approach and as a result the second approach has to be followed.

The subsystems on the robot are rotated to the robot's coordinate frame. Then they are transformed to a global reference frame. The global reference frame is selected as that of the camera. A rotation matrix R is used to describe the orientation of the robot's coordinate frame to that of the fixed reference frame. The rotation matrix with respect to roll, pitch and yaw around the reference frame is described by,

$$R = R_{z,\phi} R_{y,\beta} R_{x,\psi} \ . \tag{5.6}$$

The rotation matrix relating the IMU's coordinate frame to that of the global reference frame is derived in appendix E. The rotation matrix transforms the acceleration in the x and y directions to the global coordinate frame and is shown in the following equation,

$$R_{x,y} = \begin{bmatrix} cos(\theta + \pi/2) & sin(\theta + \pi/2) \\ sin(\theta + \pi/2) & -cos(\theta + \pi/2) \end{bmatrix} \tag{5.7}$$

The rotation matrix needs to be determined every time new IMU data is received and every time a new command is sent to the motors. The only variable in the rotation matrix is the robot's orientation (θ). Consequently obtaining an accurate estimate of the robot's orientation is important. An estimator must be used to obtain the orientation of the robot. The robot's orientation is required at the rate at which the IMU is operating and the data rate of the vision system is to low. The acceleration information used on the next chapter has been transformed to that of the global reference frame using the orientation (θ) obtained from the orientation estimator.

5.6.2. IMU Calibration

The alignment and orientation of the IMU is important for accurate measurements. However, as a result of the physical layout of the robot it is difficult to align the sensor accurately. Constructing an adjustable mechanism to align it is expensive and this will not be done. The IMU will have to be calibrated and the error compensation needs to be down in the software.

When the IMU is not aligned properly a percentage of the gravitational acceleration will be measured in the x and y directions. This results in a biased error even if the robot is stationary. This also means that some of the acceleration in the x- and y-direction will be measured in the z-direction.

Various techniques exists for calibrating the IMU. It was decided to simply zero the IMU by deducting the bias. This crude technique is used as a result of the field not being completely flat and the IMU fixture not being completely rigid. More complicated techniques that not yield significantly better results and this method was deemed satisfactory.

6. State Estimator Design

In order to fulfil the objectives of this project an estimator performing sensor fusion must be designed. This chapter describes the design and testing of a state estimator for the soccer robots.

6.1. Introduction

6.1.1. Background

A state estimator uses information obtained from multiple sensors and based on the system's dynamics predicts what is happening in the system. The robots may obtain data from various sensors, such as encoders, accelerometers, gyroscopes and cameras. Based on this data the robots should estimate their current state (velocity, orientation and position).

Various techniques exist for estimating the state based on multiple sensor inputs. Kalman filters are used in a wide variety of applications (**?**). They have especially been applied in the field of autonomous navigation (**?**). To the author's knowledge, KFs have never been used in combination with inertial sensors on a SSL robot. Although **?** used a KF to combine data of the wheel encoders with vision data. KF have been used extensively to filter the vision data of SSL robots on the OFC.

? used a continuous time KF, called a Kalman-Bucy filter. Camera data was used as the input and the filter ran on the central controller. The position and orientation of the ball and soccer robots on the field were predicted using this filter. **?** used an Extended Kalman Filter (EKF) in a high level control loop on the OFC. The SSL robot's velocity was estimated to determine the error between the velocity command and the actual robot velocity.

In the MSL KFs are run on the robots themselves. **?** evaluated the performance of a KF being used to perform self-localization on a MSL robot. It was found that when using a KF the robot's trajectory was very smooth compared to when a KF was not used. The most widely used form of the KF is the EKF which is a nonlinear form of the KF (**?**). **?** states that in navigation systems aided by vision, the KF effectively combines the visual feature observations with other measurements from GPS, inertial, or force sensors. A great deal of research has been done on fusion of vision and inertial measurements using

different types of KFs. ? did work on calibrating vision and inertial sensors on a robot using unscented KF (?). ? compares the fusion of inertial and vision data by using either an Unscented KF or an EKF. ? designed a visual augmented state estimator which used inertial measurements in combination with visual and GPS data.

Therefore it is clear that KFs can be used to effectively estimate the state of a robot when measurement noise is present. Currently in the most SSL teams the KFs are situated on OFC and only use the vision data obtained from the vision system. KFs, which use inertial and vision data, have been researched extensively and yield good results. However this has only been used in the MSL for navigation and localisation purposes. This project proposes to develop the control architecture in such a way that the KFs are run on the SSL robots themselves and that they will use both inertial and vision data. The Kalman filter equations will be discussed further in the rest of this chapter.

6.1.2. Kalman Filter Equations

KFs can be used when measurement noise is present, even if the system's true model is unknown. The KF requires a system to be described in state-space form as

$$\begin{aligned} \dot{x} &= F_c x + G_c u + w \\ y &= H_c x + v \ . \end{aligned} \tag{6.1}$$

These equations are then discretized to obtain the following equations:

$$\begin{aligned} x^{k+1} &= Fx^k + Gu^k + Nw^k \\ y^k &= Hx^k + v^k \ , \end{aligned} \tag{6.2}$$

where F is used to propagate the state forward and G describes the influence of u^k on the state vector. From equation 6.2 it is seen that the next state of the system x^{k+1} is determined from the state vector x^k , the system input vector u^k and the process noise vector w^k. The system input vector is sometimes referred to as the control vector. This is because it represents the way in which the system is controlled or driven. The measured system output y^k is driven by the state vector x^k and the matrix H. The matrix H indicates how x^k is related to the system output. The error present in the measurement of y^k, is compensated for by adding the measurement noise vector v^k.

The KF is a recursive algorithm that consists of two main parts: state estimation and state covariance estimation. These two parts are done in two distinct steps, the time update and measurement update steps. The time update step predicts the next estimate of the state and the covariance based on the system dynamics and the system input vector. This is shown in these equations,

$$\begin{aligned} {x_p}^k &= Fx^{k-1} + Gu^{k-1} \\ M^k &= FP^{k-1}F^T + Q^k \end{aligned} \tag{6.3}$$

where ${x_p}^k$ is next predicted state, P^{k-1} is the covariance matrix representing the errors in the state estimate and M^k is the predicted covariance matrix before a measurement update has been done. The measurement update step is where the state and covariance matrix is corrected by the measured output vector of the system.

$$\begin{aligned} x^k &= {x_p}^k + K^k(y^k - H{x_p}^k) \\ P^k &= (I - K^k H)M^k \end{aligned} \tag{6.4}$$

where K^k is the Kalman gain and is computed here,

$$\begin{aligned} S^k &= HM^k H^T + R \\ K^k &= M^k H^T {S^k}^{-1} \end{aligned} \tag{6.5}$$

R is the covariance matrix of the measurement noise v^k. In the case of a polynomial KF R is scalar and it is computed as,

$$R = E(v^k {v^k}^T) = {\sigma_z}^2 \tag{6.6}$$

with σ_z the standard deviation of the noise (**?**).

The sampling time of the system input u^k is usually much smaller than that of the measured system output y^k. Thus the time update steps, which are seen in equation 6.3, are done more frequently. The update steps are performed for every new u^k received. The measurements which are usually done by either a camera or a GPS are much slower and only when a new y^k is available the measurement update steps are performed. These equations will now be used to derive a filter to estimate the state of the robot based on the fusion of available sensor data.

6.2. Derivation of Estimator for Position and Velocity

6.2.1. Estimator Design

This section describes the dynamics and Kalman equations for the velocity and position estimator. The first thing is to decide what the inputs and measured outputs for the system will be. There are a variety of possible inputs for the system. **?** modelled three and four wheeled soccer robots using the voltages applied to the motors as the inputs to the system and the wheel velocities where selected as the outputs. This results in a complicated system model which is dependent of the physical system dynamics and motor dynamics. The major drawback of using this system is that it is difficult to derive an accurate model, because this requires that some of the physical robot parameters need to be

calculated experimentally. A model for the soccer robots were derived based on the method of **?** and can be seen in the appendix B.

? chose a 3-axis gyroscope and a 3-axis accelerometer as input to the system. This results in a much simpler state-space model, with most of the parameters readily available. This approach is much easier to implement and therefore a similar approach will be followed for the soccer robots.

The input to the system is acceleration and the system's output is measured using cameras. The KF derived in this section is only applied to the velocity and position of the robot in the x and y directions. The movement in the x and y directions can be decoupled. Therefore to simplify the control the estimator will be made one-dimensional and thus the position and velocity in the x and y directions will be determined separately. The system's continuous state space model for movement in one direction is seen here,

$$\begin{aligned} \dot{x} &= F_c x + G_c u + w \\ \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \end{aligned} \tag{6.7}$$

$$\begin{aligned} y &= Hx + v \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + v \ . \end{aligned} \tag{6.8}$$

The discrete state space model is required for the KF equations. The fundamental matrix (F) of a time-invariant system can de derived according to

$$F = \mathcal{L}^{-1}[(sI - F_c)^{-1}] \tag{6.9}$$

where I is the identity matrix (**?**). Using equation 6.7 and 6.9 the fundamental matrix is calculated in the following manner,

$$\begin{aligned} F &= \mathcal{L}^{-1}[(sI - F_c)^{-1}] \\ F &= \mathcal{L}^{-1} \begin{bmatrix} s & -1 \\ 0 & s \end{bmatrix}^{-1} \\ F &= \mathcal{L}^{-1} \begin{bmatrix} 1/s & 1/s^2 \\ 0 & 1/s \end{bmatrix} \\ F &= \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{6.10}$$

where T_s is the sampling time. The next step is to calculate G this is done according to

$$G = \int_0^{T_s} F(\tau) G_c \, \mathrm{d}\tau \ . \tag{6.11}$$

Using equation 6.7, 6.10 and 6.11 G is calculated as

$$G = \begin{bmatrix} T_s^2 \\ T_s \end{bmatrix} . \tag{6.12}$$

These matrices are used with the Kalman equations (6.3, 6.4, 6.5) to estimate the position and velocity of the robot. The time update step (eq. 6.3) will be performed when new acceleration data is available. The high frequency acceleration data is used to propagate the system forward. The low frequency position data obtained from the camera will be used to correct the state by using the measurement update equation 6.4. The low frequency position data prevents deterioration as a result of the low accuracy accelerometer that is used to propagate the system forward. The end result is high frequency position and velocity data describing the state of the system.

The complete KF equations for the translational filter can be seen in appendix D. From the equations it is evident that the system is linear and therefore a normal KF is used. The EKF is used for non-linear processes will not be discussed.

The Kalman gain matrix (K^k), equation 6.5, needs to be computed every time a measurement update is done. Evidently this requires that the inverse of the matrix S^k be calculated regularly. Computing the inverse is computationally expensive and since the processing power of the robot is limited, can result in a substantial reduction in the speed of the system. The Kalman gain matrix converges to a steady state value after a finite amount of iterations. The value that it converges to is dependent on the system dynamics, process noise covariance matrix and measurement noise covariance matrix. This value can be computed experimentally and then used in the system, which results in the KF reducing to a simple estimator with a fixed gain.

It is important to note that the speed of computing an inverse of the matrix is related to the size of the matrix by $O(n^3)$. Therefore reducing the size of the inverted matrix S^k will decrease processing time. This can be done by choosing the correct matrix H and other corresponding matrices. When H is selected as in the following equation, S^k reduces to

$$\begin{aligned} S^k &= HM^kH^T + R \\ S^k &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + R \\ S^k &= m_{11} + R \ . \end{aligned} \tag{6.13}$$

Thus S^k is a scalar value and the inverse simply becomes division by a scalar value. This means that the steady state value of the Kalman gain matrix does not need to be calculated, as the new value can easily be calculated by scalar division. This equation also confirms that measurement noise matrix R is a scalar value. The measurement noise is the noise associated with measurement

of the system output y^k and can be computed as indicated in equation 6.6. The standard deviation of the measurement noise of the SSL-Vision system is not known, and as a result the measurement noise has to be computed experimentally.

Process noise is the noise associated with uncertainty in the system's behaviour. The discrete process noise depends on the continuous process noise spectral density Φ_s and the sampling time T_s. **?** gives the discrete process noise matrix for a first order polynomial KF as

$$Q^k = \Phi_s \begin{bmatrix} \frac{T_s^3}{3} & \frac{T_s^2}{2} \\ \frac{T_s^2}{2} & T_s \end{bmatrix} . \tag{6.14}$$

When the actual model noise uncertainty is unknown Φ_s is a factor that can be used to optimise the KF (**?**).

Thus to optimise the KF both R and Φ_s need to be determined experimentally. When Φ_s is selected as 0 it can be assumed that the KF model of the system is accurate and it is not necessary to compensate for process noise. However with an increase in uncertainty the system's behaviour will be more accurate, but this will also result in a steady state error for the system. The addition of process noise prevents divergence, but results in a steady state error. The larger the process noise the more the system will depend on the actual measurements.

A popular form of numerical simulation optimization is the Monte Carlo method. Random samples are run and the results are used to optimize the system. In order to optimize the KF a desired true measurement is required to compute the filter's error. This was done by fitting a large order polynomial to the actual camera data. The error function was defined as the difference between the polynomial fitted to the camera data and the KF estimate.

The simulation was run for random values of process and measurement noise using data obtained from the robots. The simulation yielded a very large value for $\Phi_s = 5000$ and a very small value $R = 2$. This is as expected, because the error function which was used in the Monte Carlo simulation was formulated assuming that the position information from the camera was 100% correct. That is why R is so small. However these values can't be used as the camera information is not 100% correct. Experimental testing was done to determine optimal values. The optimal values were selected as $\Phi_s = 1000$ and $R = 25$.

6.2.2. Results

Figures 6.1 and 6.2 illustrate the results obtained with the optimal values of R and Φ_s. Figure 6.1 depicts the position estimation of a moving robot. The position data was obtained from the overhead camera, the IMU and the KF.

The positional data from the IMU is obtained by integrating the acceleration data twice. The KF estimate performs as it should and follows the camera data closely. The position estimate obtained from the IMU deviates as a result of bias errors and sensor noise.

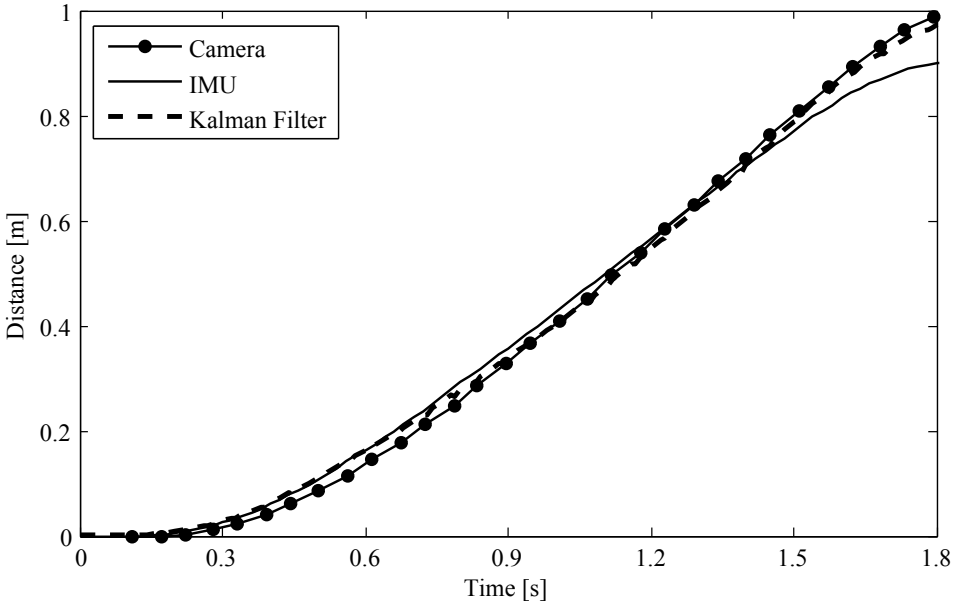


Figure 6.1: Position estimate with $\Phi_s = 1000$ and $R = 25$

Figure 6.2 shows the velocity estimate of the moving robot. When the camera data is differentiated to obtain velocity information the noise in the data is increased. This results in oscillation in the velocity derived from the camera data. From figure 6.2 it is seen that the KF estimate results in a good velocity estimate. It is important to note that the velocity estimate is more important than the positions estimate; this is because the velocity will be used as the feedback for the HLC.

The two mayor advantages of using the KF will be illustrated in the rest of this section. Firstly the KF enables the robot to estimate its position and velocity even though there is a loss in communication between the OFC and the robot. Tests were performed during which no camera data was sent to the robot for a period of 500 ms, the results are shown in figure 6.3. Case 1 shows the filter output with the 500 ms breach of communication, whereas Case 2 shows the filter output for no loss in communication. Although the 500 ms communication break signifies a loss of 27% of the camera data, there is only a 2.5% increase in the positional error. When there is no camera data available the filter relies solely on the acceleration data. Therefore the accuracy of the acceleration data has a large influence on the error induced by the communication delay. Notwithstanding, the robot is still able to estimate its position and velocity within reasonable accuracy.

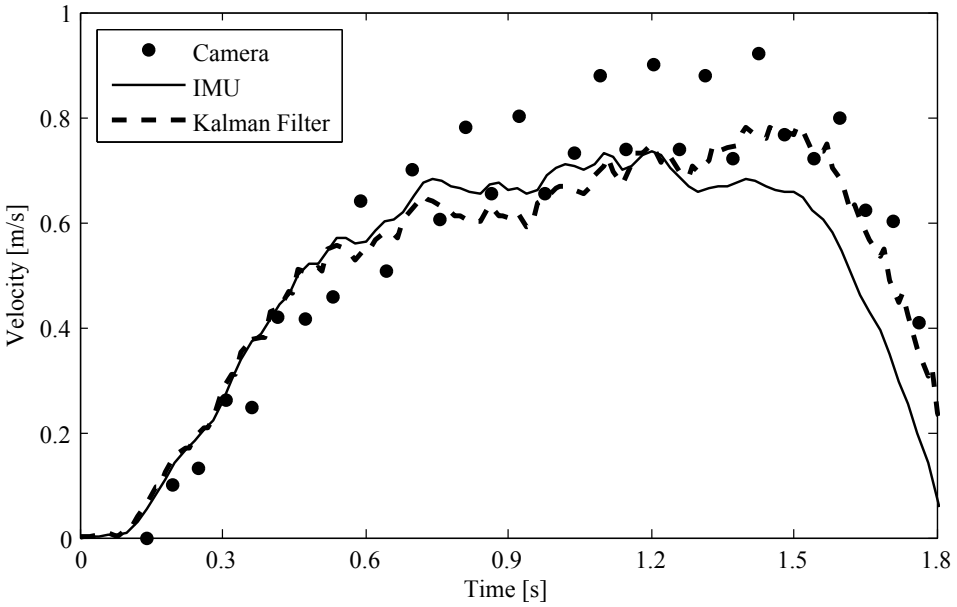


Figure 6.2: Velocity estimate with $\Phi_s = 1000$ and $R = 25$

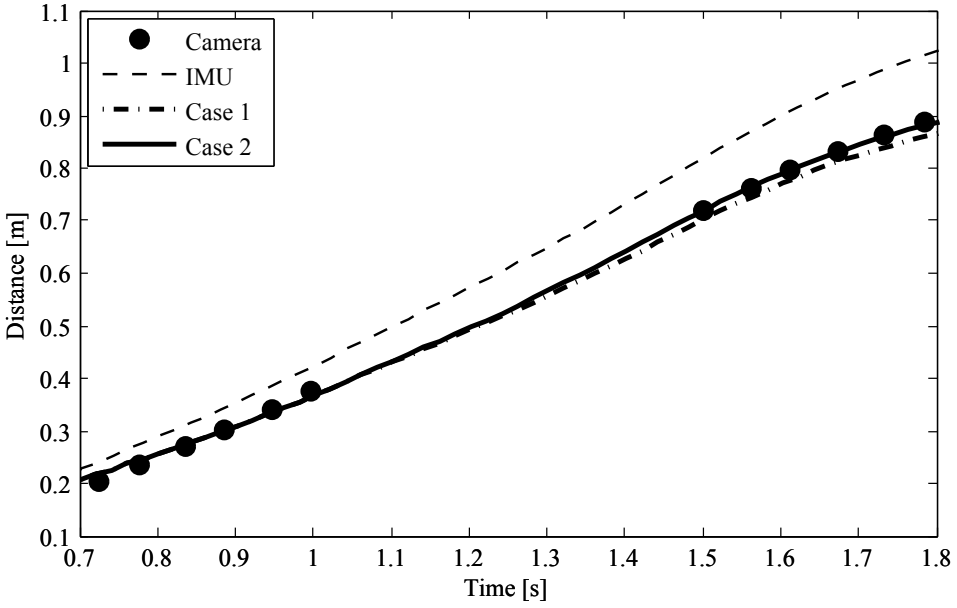


Figure 6.3: Position estimate during a 500 ms loss in communication

The second major advantage is that the KF provides data at a higher rate than the camera, as seen in figure 6.4. The KF has a data rate very close to that of the IMU. This is because a time update step is performed when new IMU data is available. However although the accelerometer's maximum available data rate is 3200 Hz, the accelerometer is set up to operate at a data output rate of 100 Hz. This is because the acceleration data is only sampled

at a rate of 50 Hz, which is the frequency at which the main loop is executed. This means that the data rate of the KF can be increased by optimizing the main control loop.

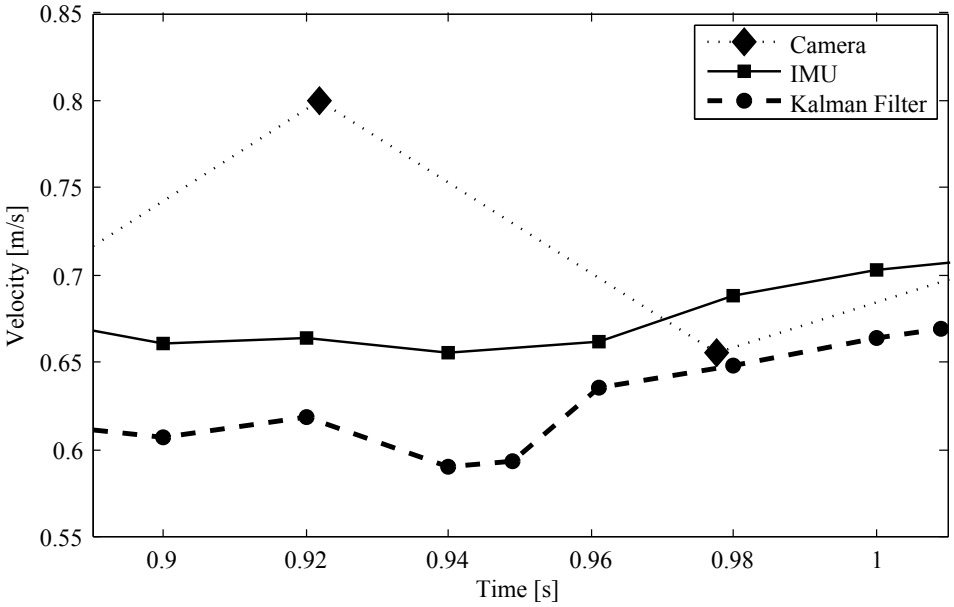


Figure 6.4: Close up of velocity estimate indicating the data rate

The velocity of a stationary robot was estimated to verify the accuracy of the KF. This was done because the position and velocity of a stationary robot is known. The results obtained are illustrated in figure 6.5. The vision system has a positional error of 0.001 m on a stationary robot, however this increases when the robot as moving. This produces an error in the velocity estimate of 0.02 m/s. At low velocities the bias errors and measurement noise of the accelerometer becomes significant. The velocity derived from the acceleration data produced a maximum error of 0.047 m/s. The maximum error of the estimator is 0.045 m/s, which is lower than the error produced by he IMU data. Although the error is larger than that produced by the camera it is less oscillatory and thus more suitable for use in the HLC.

6.3. Derivation of Estimator for Orientation and Angular Velocity

The gyroscope and camera can be used to determine the orientation and the angular velocity of the robot. With these sensors the orientation and angular velocity is directly observable, thus no estimator is actually required. However, the data obtained from the camera is at a low frequency and the data from both sensors contain noise. Therefore in order to improve the quality of the

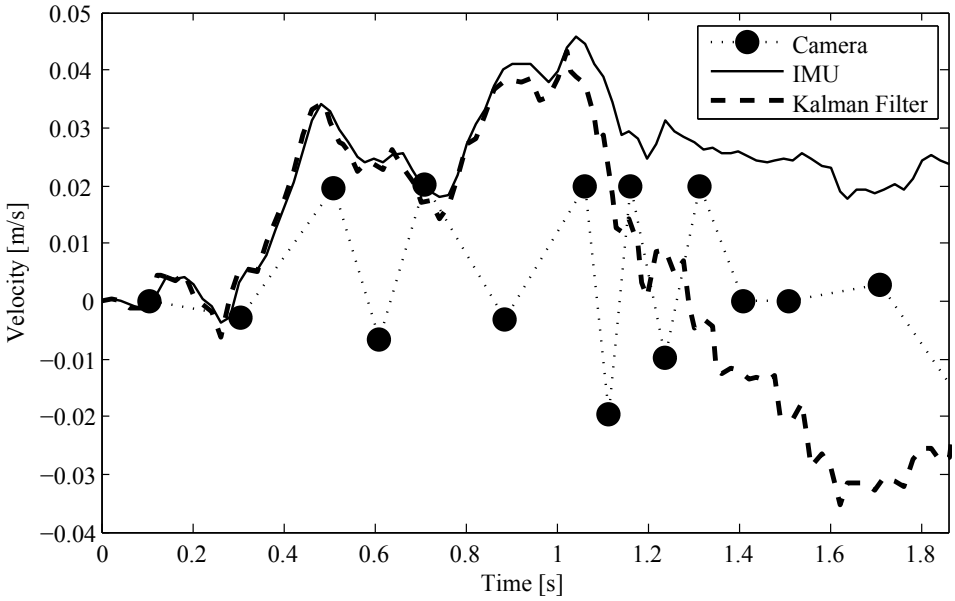


Figure 6.5: Velocity estimate of stationary robot

information an estimator or filter is required. The same approach as in the previous section can be followed to derive a KF. There is an alternative, discrete filters can also be used. This section starts by looking at discrete filters and then designs a KF. Both solutions are then compared to each other.

6.3.1. Complementary Filters

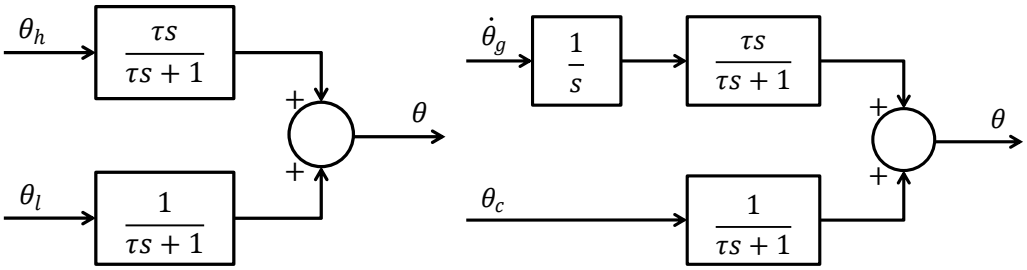
Simple discrete filters could be used; if the low frequency camera data was filtered with a low-pass filter it would remove the noise. However it would still be low frequency data and the output will lag due to the averaging effect of the low-pass filter. The angular rate obtained from the gyroscope could be integrated and filtered with a high-pass filter. This data would be high frequency, although it would contain a bias error.

There is an alternative to using a KF, which combines low-pass and high-pass filters. The Complementary filter is basically a steady-state KF which does not take any statistical description of the noise into account (**?**). The filter adds together two different signals and produces one filtered signal. The one input signal has reliable high frequency information whereas the other one has reliable low frequency information. The filter combines these signals to estimate the actual state. Figure 6.6a illustrates a basic Complimentary Filter (CF). The low-pass and high-pass filters must be chosen with a combined gain of one. This is demonstrated when the gain of both filters are added,

$$\frac{\tau s}{\tau s+1}+\frac{1}{\tau s+1}=\frac{\tau s+1}{\tau s+1}=1\ , \tag{6.15}$$

were τ is the time constant of the filter.

The CF that will be used to determine the orientation is depicted in figure 6.6b. The orientation (θ_c) data obtained from the camera has a lower data rate and contains high frequency noise. Therefore it is filtered with the low-pass filter to remove the high frequency component. The angular rate obtained from the gyroscope ($\dot{\theta}_g$) is integrated to determine the orientation. This data contains a bias error due to noise, sensor drift and the integration process. Therefore only the high frequency component is used. The result is an estimate of the robots orientation consisting of the reliable components of both sensors signals.



(a) Basic complementary filter (b) Complementary filter for orientation estimation

Figure 6.6: Complementary filters

The filtered orientation is obtained by adding the filtered high-pass and low-pass orientations,

$$\theta = \theta_{hp} + \theta_{lp} \ , \tag{6.16}$$

where θ_{hp} is obtained by integrating and filtering the gyroscope data and θ_{lp} is the filtered camera data. The high-pass part of the CF is discretised, yielding the following equations,

$$\begin{aligned} {\theta_g}^k &= T_s \cdot \dot{\theta}_g + {\theta_g}^{k-1} \\ \theta_{hp}^k &= \frac{\theta_g^k - \theta_g^{k-1} + \theta_{hp}^{k-1}}{1 + \frac{T_s}{\tau}} \end{aligned} \tag{6.17}$$

where T_s is the sampling time and τ is the time constant of the filter. The low-pass part of the filter was discretised to obtain the following equation,

$$\theta_{lp}^k = \frac{\frac{T_s}{\tau} + \theta_{lp}^{k-1}}{1 + \frac{T_s}{\tau}} \ . \tag{6.18}$$

Both these equations show that the filters are dependent on the two parameters, the sampling time (T_s) and the time constant (τ).

The sampling time is measured every time these equations are executed and does not need to be determined experimentally. The filter's time constant τ was selected with a value close to that of the sampling time, $\tau = 0.019\ s$. This value was validated using simulations which used the actual gyroscope and camera data obtained from the robot. Figure 6.7 shows the results obtained from the CF when using different values of τ. The values of was selected as $\tau = 0.05\ s$ for Case 1, $\tau = 0.019\ s$ for Case 2 and $\tau = 0.01\ s$ for Case 3. From this figure it can be seen that Case 2 follows the camera data with reasonable accuracy and without excess oscillations.

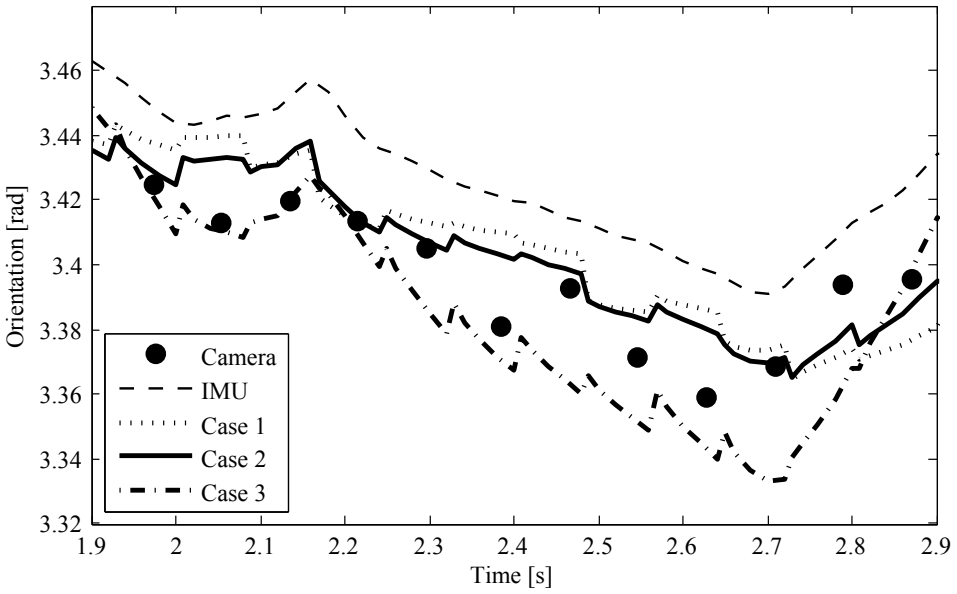


Figure 6.7: Complementary filter results for various time constants

6.3.2. Kalman Filter for Orientation

The Kalman filter can be used to estimate the orientation of the robot. Therefore a KF will be designed and tested. Where after the result will be compared to that of the CF. The filter was derived following the same process as in section 6.2.1. The input to the system is the angular rate obtained from the gyroscope and the observed output is the orientation obtained from the vision system. The system's continuous state space model for the orientation of the robot reduces to,

$$\begin{aligned} \dot{\theta} &= F_c\theta + G_c u + w \\ \dot{\theta} &= 0 \cdot \theta + 1 \cdot u + w \end{aligned} \tag{6.19}$$

$$\begin{aligned} y &= H\theta + v \\ y &= 1 \cdot \theta + v\ . \end{aligned} \tag{6.20}$$

The fundamental matrix of the discrete state space model is determined using equation 6.9, as $F = 1$, and G is calculated using equation 6.11 as $G = T_s$, where T_s is the sampling time. Therefore the KF equations for estimating the orientation of the robot can be written using equation 6.2 as

$$\begin{aligned} \dot{\theta}^{k+1} &= \theta^k + T_s \cdot u^k + w \\ y^k &= \theta^k + v \ . \end{aligned} \tag{6.21}$$

The rest of the variables are all calculated as in section 6.2.1. One important variable that needs to be checked is S^k, the inverse of S^k is required in equation 6.5.

$$\begin{aligned} S^k &= HM^kH^T + R \\ S^k &= 1 \cdot m_1 \cdot 1 + R \\ S^k &= m_1 + R \end{aligned} \tag{6.22}$$

Thus S^k also becomes a scalar value and the inverse does not need to be computed. The process noise (Q^k) is scalar and is defined as $Q^k = \Phi_s$ for the orientation estimator. Therefore this KF also has two scalar parameters, R and Φ_s, that need to be determined experimentally. Figure 6.8 illustrates the KF's estimate of the robot's orientation for various values of Φ_s and R. Using the experimental results Case 3 was chosen, with $\Phi_s = 20$ and $R = 400$.

This is in sharp contrast to the parameters of the KF that is used to estimate the position and velocity of the robot. In this case Φ_s is much smaller than R, were for the position and velocity KF it was the other way around. There are two reasons for this, firstly the gyroscope has less noise than the accelerometer and therefore the uncertainty is much lower. Secondly there is a higher uncertainty in the orientation data than in the position data obtained from the vision system.

6.3.3. Estimation of the Angular Velocity

Unlike the linear velocity of the robot, the angular rotation rate can be measured directly and therefore does not need to be estimated. **?** used the unfiltered output of a gyroscope to obtain the angular velocity of a SSL robot. The gyroscope which is used in this project has a built in low-pass filter with an adjustable bandwidth (**?**). Thus no filtering of the output is required. Nonetheless tests were done to see what the effect was of filtering the output. The filtered output did not produce significant improvements while introducing lag. The angular velocity obtained from a stationary robot is shown in figure 6.9. The output oscillates around zero with a maximum error of 0.04 rad/s. This is deemed adequate and therefore the output of the gyroscope will be used as it is.

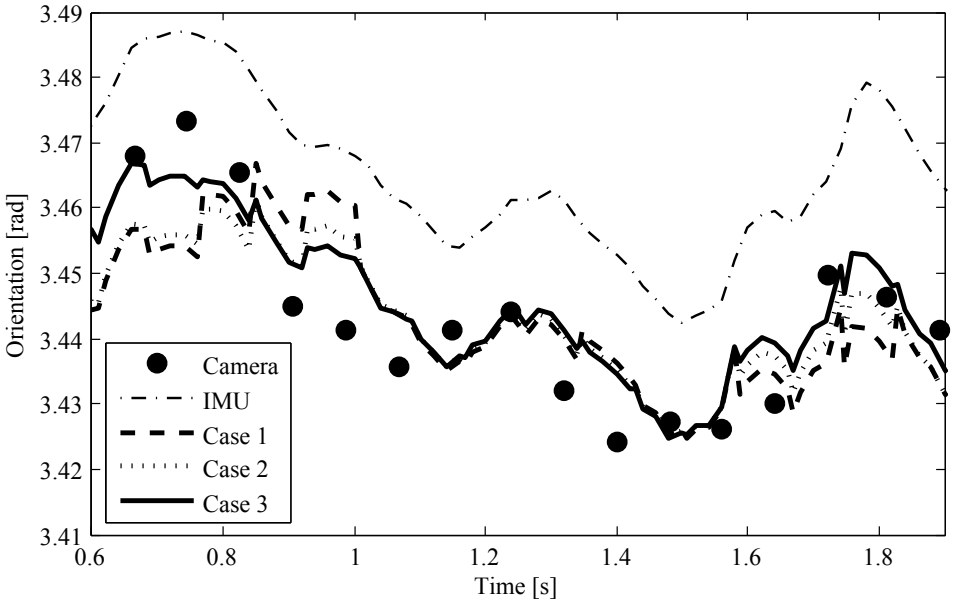


Figure 6.8: Orientation estimate obtained from Kalman filter for: Case 1 $\Phi_s = 100$ and $R = 100$, Case 2 $\Phi_s = 25$ and $R = 100$ and Case 3 $\Phi_s = 20$ and $R = 400$

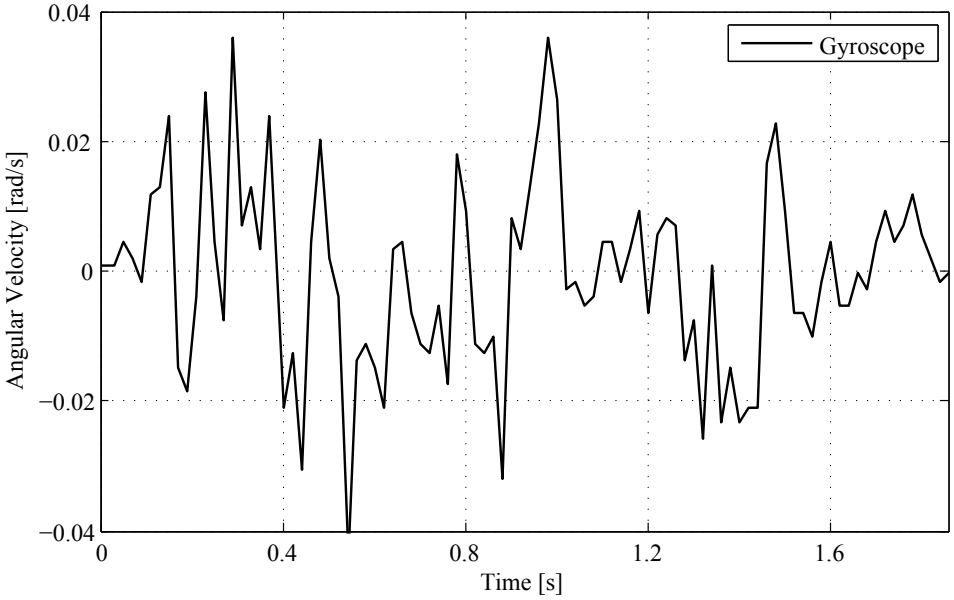


Figure 6.9: Gyroscope output of stationary robot

6.3.4. Outliers in Orientation

Adequate light is required for the vision system to perform optimally when the robots are moving at high speeds. Poor light conditions result in inaccurate orientation data from the vision system. The measured orientation data

contains outliers as a result of incorrect measurements. These outliers have an adverse effect on both the KF and the CF, this can be seen in figure 6.10.

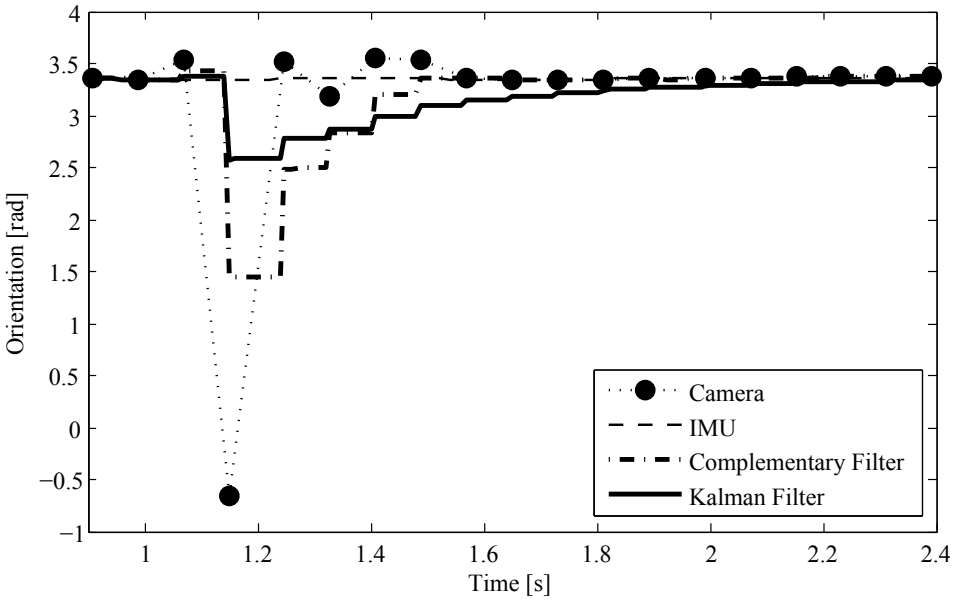


Figure 6.10: Orientation data showing effect of false camera reading

These outliers create large disturbances in the orientation control of the robot. To ignore these outliers the system was modified in the following way. The angular rate required to obtain the change in orientation, indicated by the last two camera measurements, is determined. If this value is above the maximum angular rotation of the robot, it is neglected and not used. The result is seen in figure 6.11, the simulation was repeated with the same data as in figure 6.10. The modified algorithm resulted in a 91.56% reduction in error of the CF and a 81.54% reduction in the error of the Kalman filter. There is still an error in the estimate of the orientation although it is considerably less than when the algorithm is not modified.

The outliers are not the only rapid change experienced in the orientation data. The vision system's angular range is between $-\pi$ and π. Therefore when a robot rotates past π the measurement changes by an angle of 2π to $-\pi$. The system was modified to handle this non-linearity in the orientation data. Thus when this occurs the filters are adjusted accordingly. This allows the filters to provide an accurate estimate of the robots' orientation.

6.3.5. Results

The Kalman filter and the Complementary filter yield very similar results. Figure 6.12 compares the output of both the filters. Although the KF seems to yield a less oscillatory output, there is no clear difference between the filters.

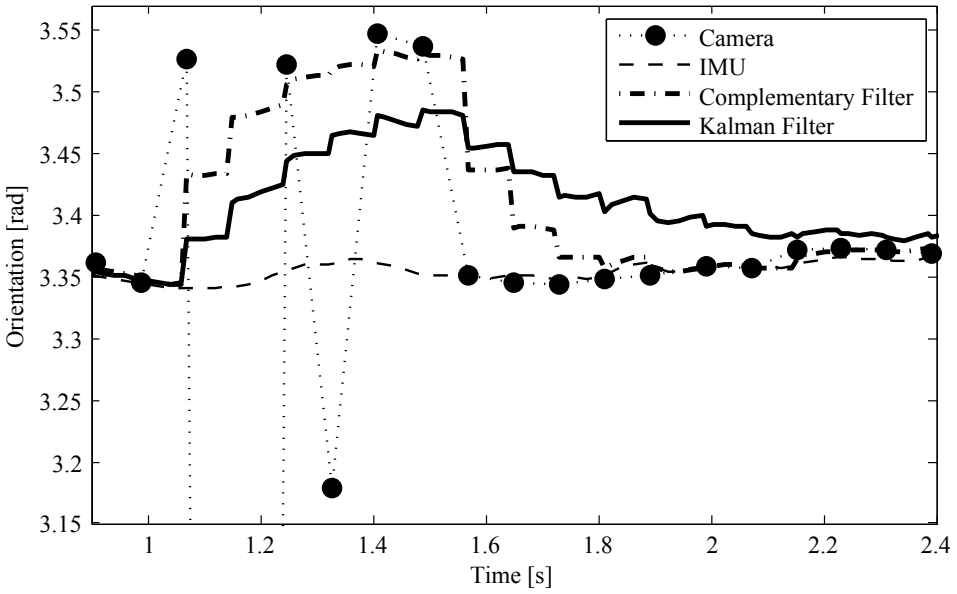


Figure 6.11: Orientation data showing effect of false camera reading on modified system

Figure 6.13 shows the output of both filters for a stationary robot. Although both perform satisfactory the KF is less affected by the noise in the camera data than the CF. Therefore the KF will used to to estimate the orientation throughout the rest of the project.

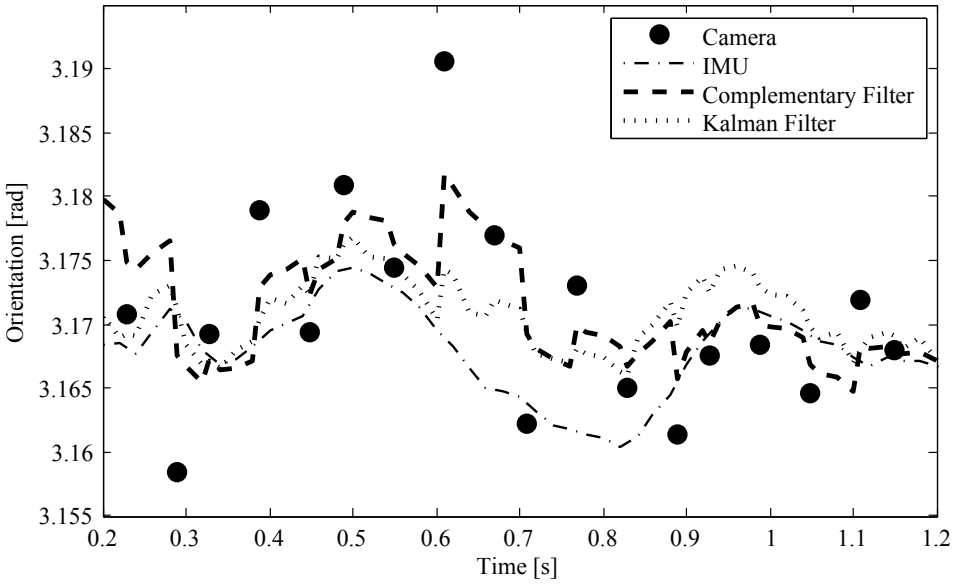


Figure 6.12: Simulation results of Kalman filter and the Complementary filter

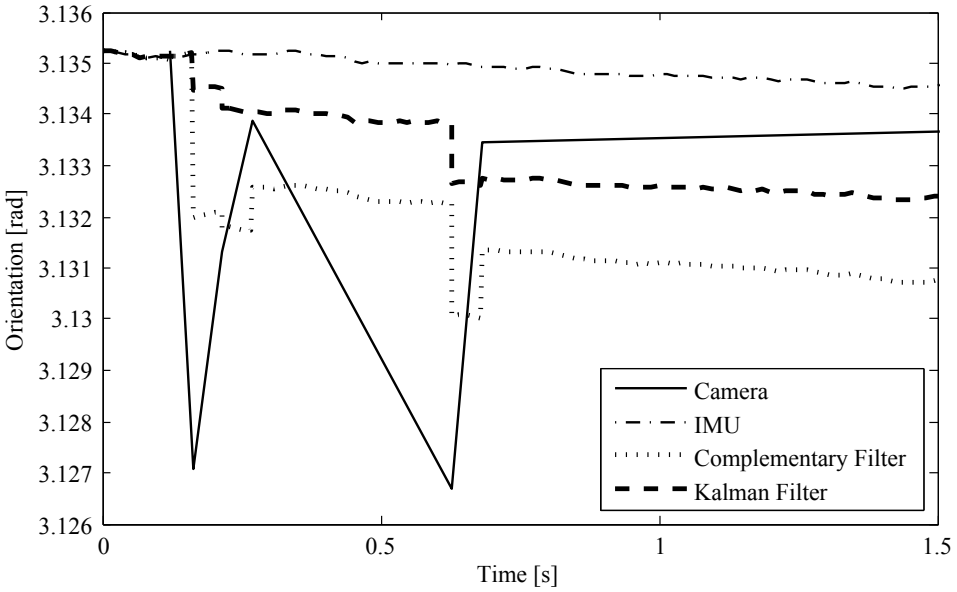


Figure 6.13: Simulation results of Kalman filter and the Complementary filter for a stationary robot

7. Implementation and Testing

The translational and rotational velocities obtained from estimators are used as the feedback for the HLC. This chapter illustrates the testing and implementation of the HLC on the soccer robots.

7.1. Experimental Setup

The test performed in this chapter were done at the robot soccer field at Stellenbosch University. The OFC used the Player system (chapter 3) to communicate with the robots. The SSL-Vision system was used to record the position of the robot for the various tests. No artificial light was used during the tests and therefore the test relied on ambient light. All the hardware that is used in the experiments are discussed in chapter 4.

7.2. High Level Controller PI Calibration

The HLC was designed to consist of three separate PI controllers. Two controllers were designed to control the translational movement in the x and y directions, while a third controller controls the robots rotation. This section starts by discussing the optimization translational controller and thereafter discusses the rotational controller optimization.

The controller parameters have to be calibrated manually by performing experimental tests. Tests were performed during which one parameter was kept constant and the effect of varying the other parameter was observed. Figure 7.1 illustrates the effect of varying the integral parameter of the translational controller. SP is the velocity set point, while the measured velocities are labelled according to the integral parameters used, I = 3 and I = 5. The velocity estimate of the KF is assumed to be 100% correct and throughout this chapter it will be used to indicate the robots measured velocity. From the figure it is evident the steady state error decreases with an increase of the integral parameter. However higher integral parameters result in an increase in oscillation of the controller output. An intermediate integral parameter of I = 6 was chosen for the translational PI controller.

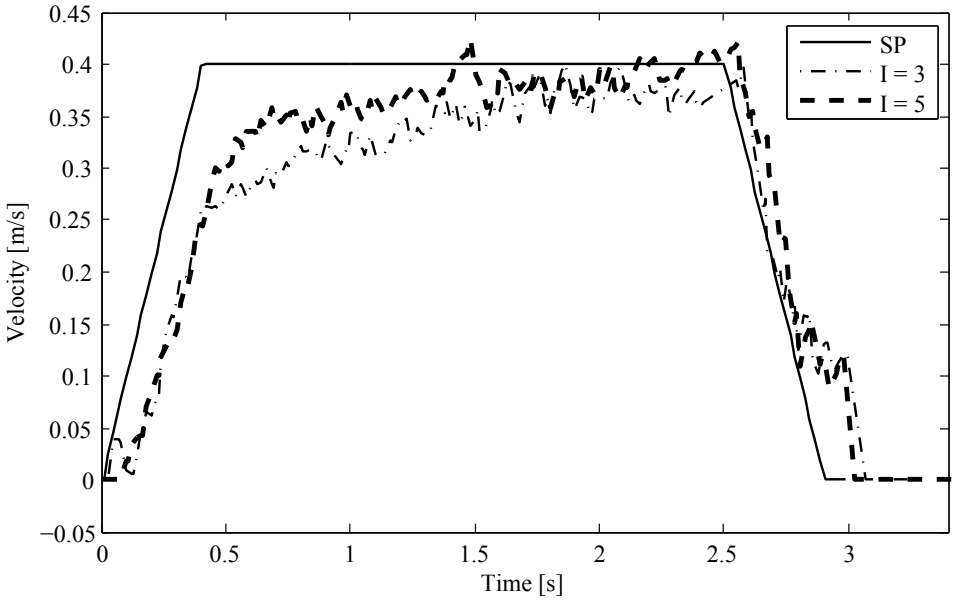


Figure 7.1: Velocity output as a result of varying integral parameter

The proportional value was optimised in a similar manner. The proportional parameter was varied and the output recorded. Figure 7.2 illustrates the measured robot velocity with two different proportional values, P = 2 and P = 4. The proportional parameter does not yield a large improvement in the output of the system and large proportional values result in an oscillatory controller output. Using the optimal integral value of 6 the proportional parameter was selected as P = 3.

The motor controller's design limits the value of the velocity commands that can be sent to the wheels. The result is that the translational and rotational velocity commands are restricted to values below 1 m/s and 2 rad/s respectively. Thus the outputs of the PI controllers are limited to these maximum values. The robot's maximum velocity was restricted to 0.6 m/s, the reason for this can be seen in appendix B.2.

Figure 7.3 demonstrates the behaviour of the translational PI controller. The velocity Set Point (SP), the Controller Output (CO) and the measured speed (MS) is indicated on the graph. The optimal translational PI parameters, P = 3 and I = 6, were used for this controller. The CO is very large at the beginning and almost reaches the maximum allowed value of 1 m/s. Therefore even though the PI parameters could have been increased this would not have resulted in a significant improvement in the performance of the controller.

The robot only reaches its desired SP velocity after about 1.75 s. This is as a result of the mechanical inertia of the robot's motors and gears. This means that the ability of the robot to follow its velocity SP can be improved by improving the manufacturing and design of the robot's drive system.

The rotational PI controller was experimentally calibrated in the same way

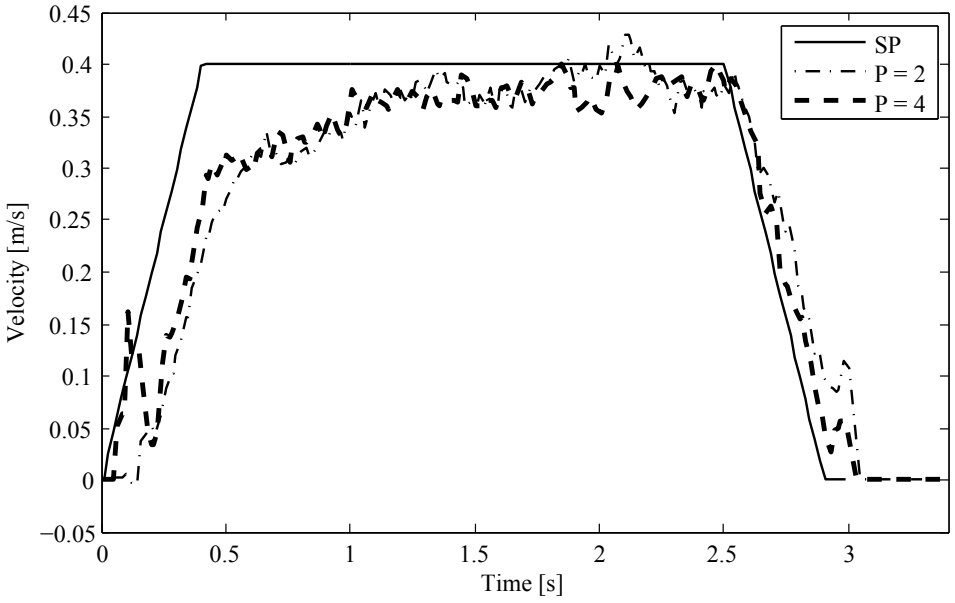


Figure 7.2: Velocity output as a result of varying proportional parameter

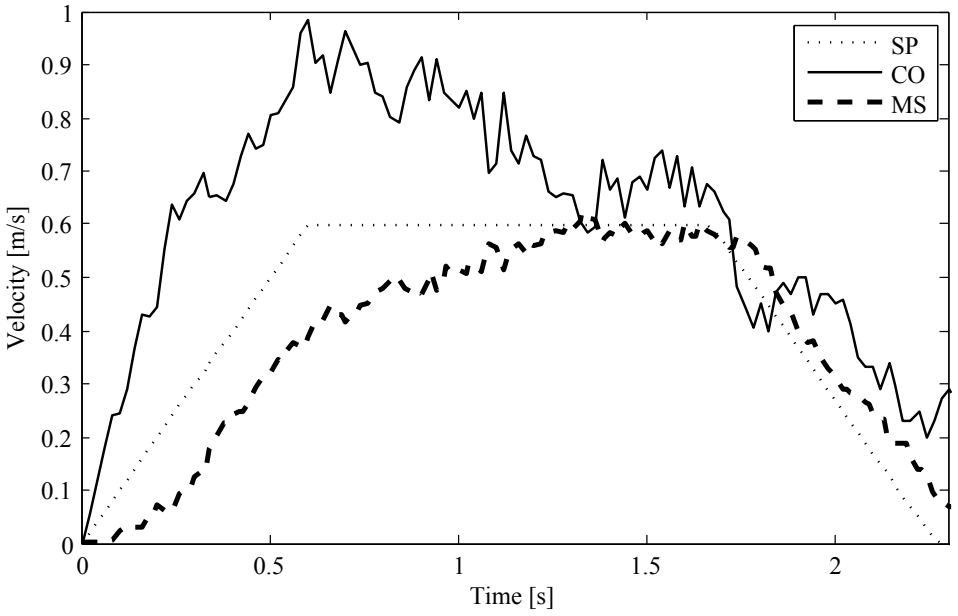


Figure 7.3: Behaviour of translational PI controller

as the translational controller. Figure 7.4 illustrates the performance of the rotational controller. The angular velocity SP will always be zero for this project. The optimum PI parameters were selected as P = 1.5 and I = 2. It is evident from this figure that there is two regions were the robot significantly deviates from its desired angular velocity. This is a result of the acceleration and deceleration of the robot at the beginning and end of a manoeuvre.

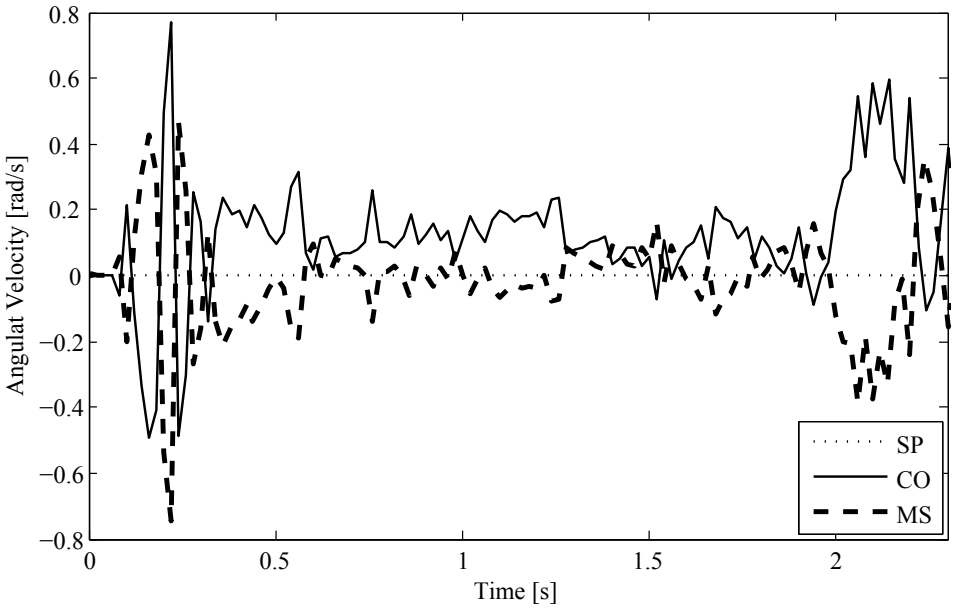


Figure 7.4: Behaviour of PI controller

The deviation from the desired angular velocity is as a result of the motor controller's performance. The motor controller uses a daisy chain to communicate. This results in the motors not receiving their velocity commands at the same time. The first motor in the daisy chain is always the first to receive a new command. When it start to rotate the others motors are still rotating at the previous required velocities. The result is that when the robot accelerates undesired rotation occurs.

The maximum allowed acceleration has a large effect on the robot's ability to perform accurate trajectory tracking. Figure 7.5 illustrates the orientation of the robot during deceleration for various maximum acceleration values. The values are a = 1 m/s^2, b = 1.5 m/s^2 and c = 2 m/s^2. The sharpest change in orientation is that of c, which has the largest acceleration. Because the robot needs to stop in a shorter period, the variance between the consecutive velocity commands are greater, resulting a greater rotation of the robot.

7.3. Trajectory Tracking

The goal of the HLC is to enable the robot to follow a desired trajectory. This section will evaluate the performance of HLC, which is situated on the robot instead of the OFC. The ability of the robot to follow a square 1 m by 1 m trajectory will be tested.

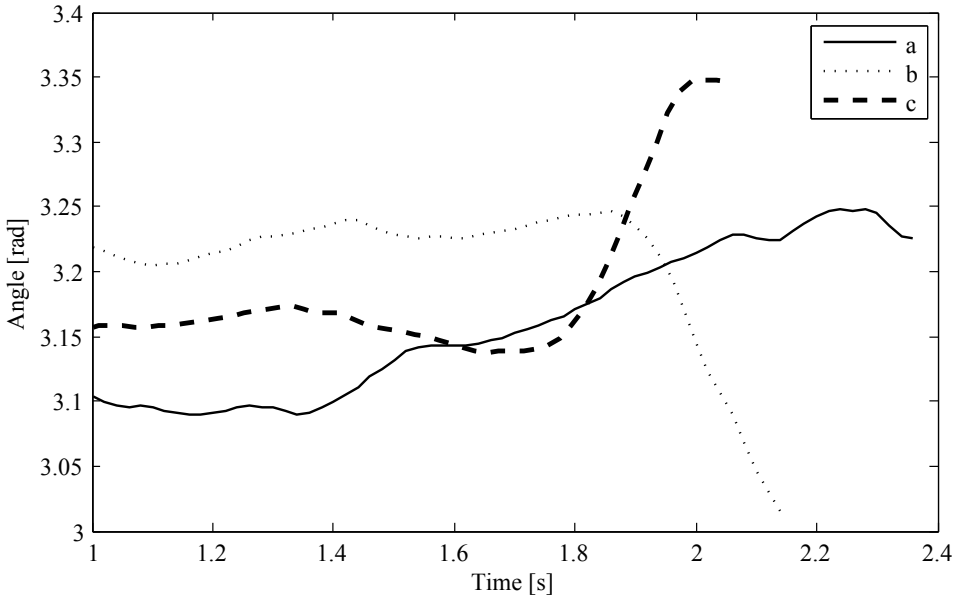


Figure 7.5: Orientation of robot during different maximum accelerations

7.3.1. Trajectory Tracking Using the IMU as HLC Feedback

The previous chapter discussed the estimators which are used to obtain accurate feedback for the HLC. This section will compare the performance of a robot using the KF developed in section 6.2 to that of a robot using only the feedback of the on-board IMU.

The Kalman estimate is corrected by using the measured output of the system, obtained from the vision system. Thus if the robot is not sent camera data the Kalman estimate is not corrected and drifts away from the actual state of the robot. However the burden on the communication system is greatly reduced if the robot does not require camera data. This is a result of only commands having to be sent, because no data is required. This results in a much faster communication system.

Figure 7.6 shows the ability to follow a desired trajectory. As expected the velocity estimate obtained from the IMU gets progressively worse as a result of the bias error of the IMU. The difference between the velocity estimate of the KF and the IMU is plotted in figure 7.7b. The induced positional error is displayed in figure 7.7a. The positional error at the end of this section is 0.1 m which is relative small compared to the distance travelled. However at low speeds this becomes increasingly prominent as is evident in figure 7.8. The figure shows that while the robot is actually moving at a velocity of 0.02 m/s, the IMU indicates that the robot is stationary.

The ability to follow the square trajectory is illustrated in figure 7.9. The movement in the negative x direction is larger than that in the positive x

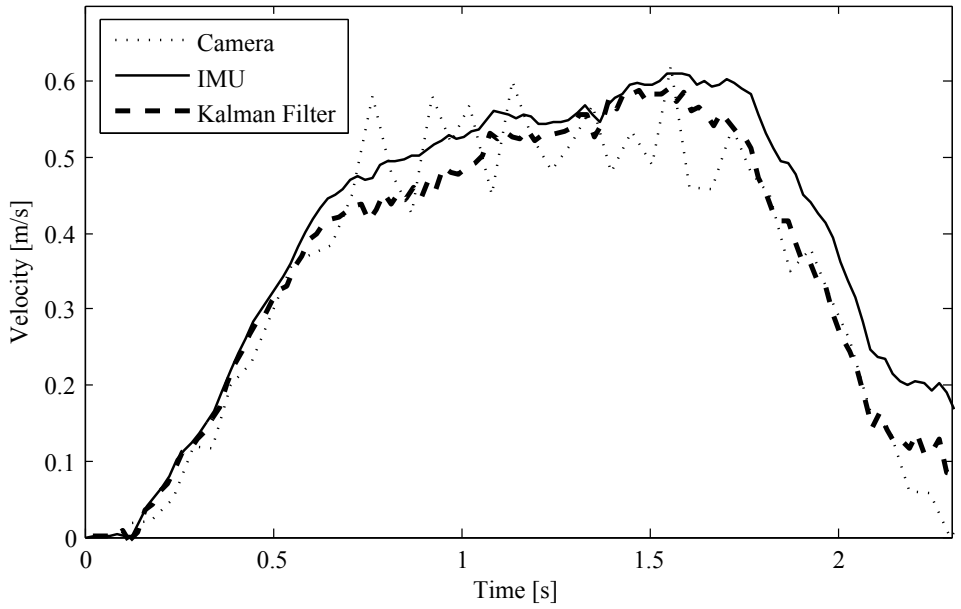
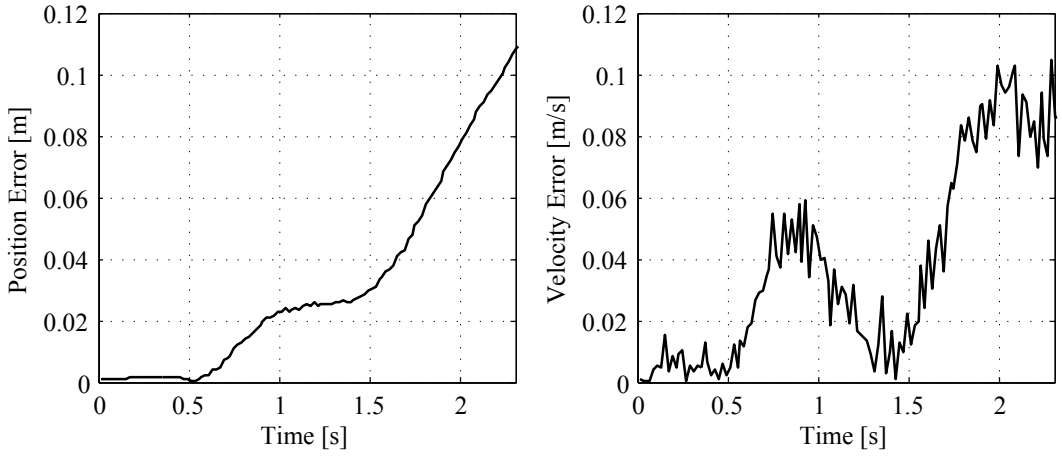


Figure 7.6: Velocity output of KF compared to that of IMU



(a) Position error without camera data

(b) Velocity error without camera data

Figure 7.7: Error in estimate of system not using camera data

direction. This is as a result of the bias error of the accelerometer measuring the acceleration in the x direction. This causes the deviation in the negative x direction when the robot is moving in the y direction. This bias error is as a result of the orientation and placement of the IMU. Gravity has an effect if the IMU is not calibrated and positioned correctly. The robot's final destination is 0.542 m from the original position from where the trajectory was commenced. This is a considerable error and will be compared to the error obtained when the KF is used as feedback.

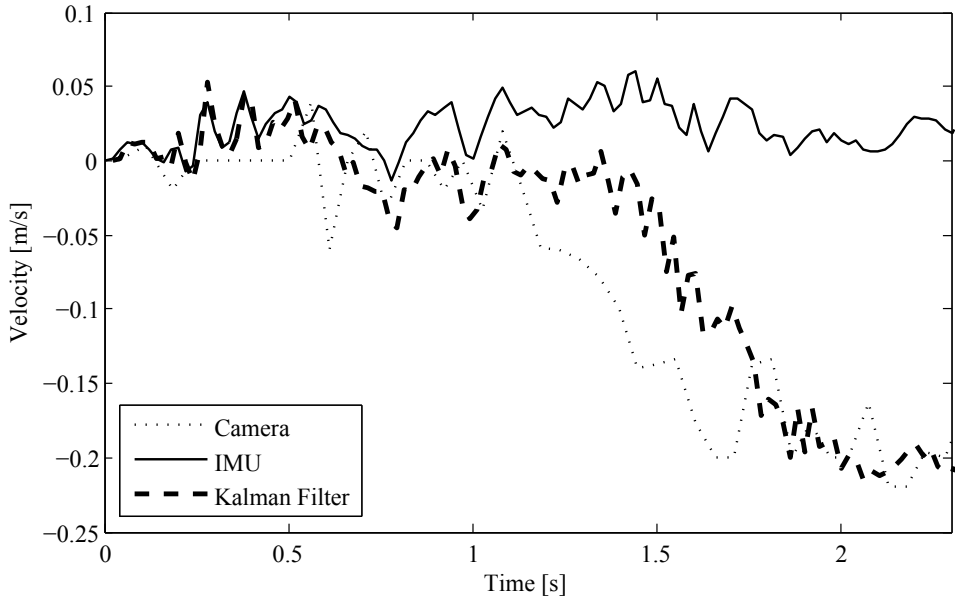


Figure 7.8: Output of the KF compared to that of the IMU for a stationary robot

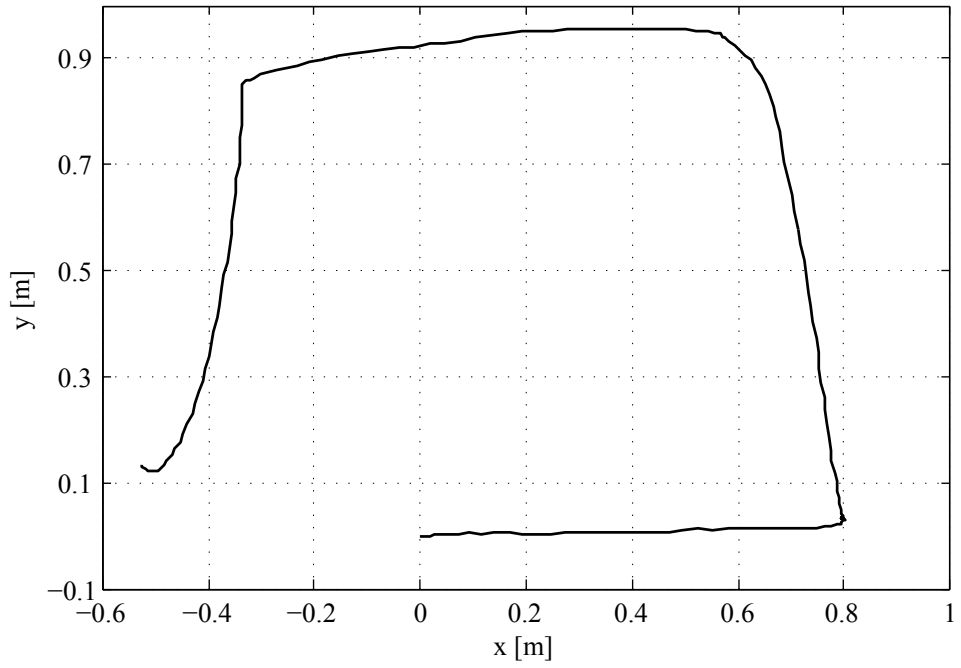


Figure 7.9: Robot following square trajectory without aid of vision system

7.3.2. Trajectory Tracking Using the Kalman Filter as HLC Feedback

Similar square trajectory tracking tests have been performed by other authors. Figure 7.10 compares different results obtained by **?** using various control

strategies. The *PID with Dynamic Model* is very similar to the controller being used in this project. The only major difference is that in this project the HLC is situated on the robot and not on the OFC.

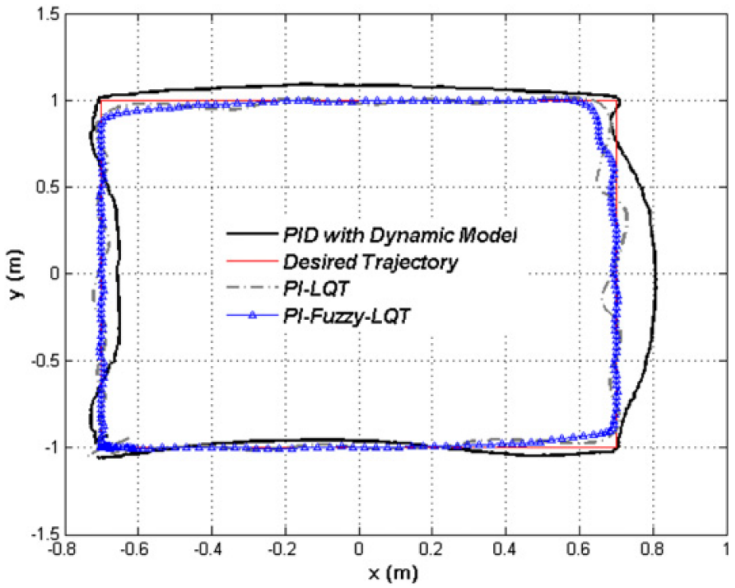


Figure 7.10: Square command tracking performance of (**?**)

The *PID with Dynamic Model* is able to follow the trajectory closely, stopping very close to the point from where the manoeuvre was commenced. The robot's movement oscillates around the desired trajectory and is seldom on the specified path, although the maximum deviation is less than 0.1 m.

Figure 7.11 shows a square trajectory tracking test performed with the system developed by this project. The robot was told to move 1 meter in four directions. Thus it was not given destinations, and rather told to move 1 m in each directions one after the other. Thus the error accumulates throughout the test. The ideal would be for the robot to accurately follow the desired path. This test illustrates the consistency of the robot's movement. The robot is able to stop within 0.088 m from the start position. This is an 83.3% reduction in the error compared to when only the on-board IMU is used. The large reduction in error justifies the implementation of the KF on the robots.

From figure 7.11 it is evident that the robot is able to follow the trajectory although the distance travelled is always short of the required distance. The maximum deviation from the desired ideal trajectory is always within 0.1 m, which is similar to that of the PID in figure 7.10. However the robot the trajectory in figure 7.10 was generated using a higher maximum velocity indicating a possible cause of the large oscillation

Figure 7.12 shows an image of the movement of the robot while executing a square trajectory. This image was constructed out of the images used by the

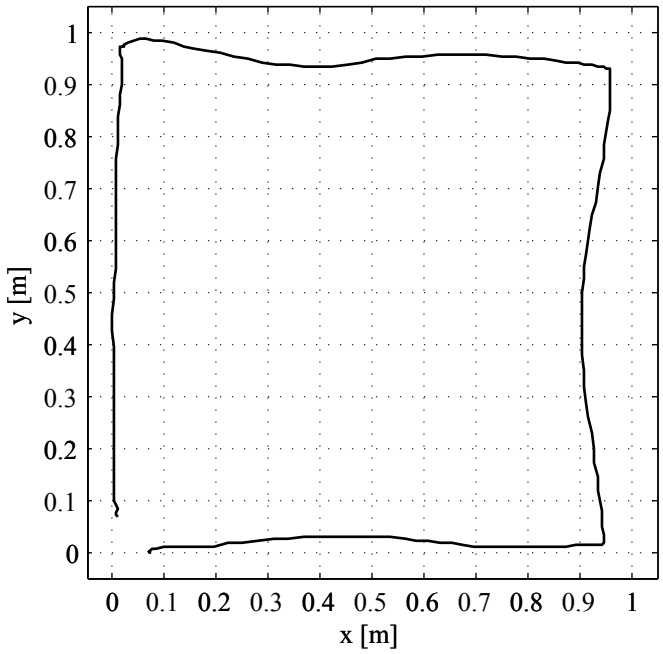


Figure 7.11: Square command tracking performance of SSL robot

vision system. The result is consistent with the one obtained in figure 7.11, thus validating the accuracy of the KF as a position estimator.

Figure 7.12: Overhead image illustrating square command tracking performance

7.4. Robot Direction and Radial Deviation

The previous section demonstrated the robot's ability to follow a specified trajectory. This section will quantify the performance of the HLC and compare it to that of a robot using only a LLC.

? developed a LLC for the first iteration SSL soccer robots at Stellenbosch University. Figure 7.13 depicts the angular deviation from the desired direction. The green line only indicates the final destination of the robot, it does not indicate the radial distance or the exact path followed. Figure 7.13 demonstrates that the robot experiences random deviation from the desired robot direction, this is because **?** only uses a LLC.

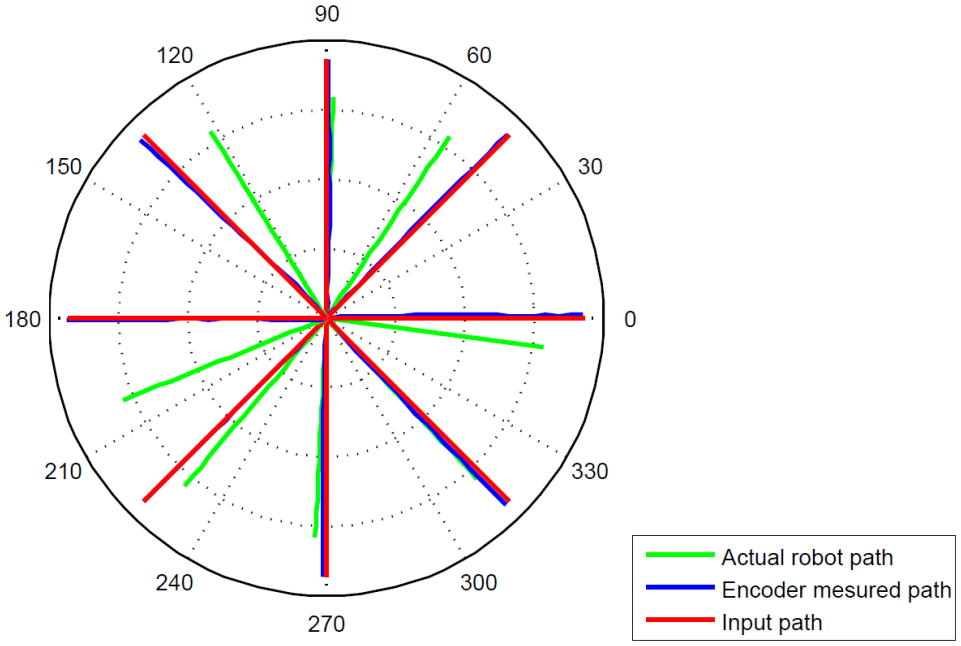


Figure 7.13: Directional deviation with LLC (**?**)

? performed the directional test six times and the mean angular deviation is listed in the second column of table 7.1. It was found that there is larger angular deviation in certain directions, this is as a result of the geometry of the robot. The robot is not symmetric with respect to its x-axis as seen in figure 4.3.

Directional deviation and radial deviation tests were performed using the developed LLC with the current soccer robots. The robot's trajectory was recorded using the overhead camera instead of physically measuring the final destination. Multiple tests were performed and the results are shown in the polar plot in figure 7.14. The radial distance as well as the direction of the robot is indicated. The desired objectives were 1 m trajectories 45° apart.

Table 7.1: Mean angular deviation from desired trajectory using LLC

Desired direction	Mean error (deg) (?)	Mean error (deg)	Standard deviation (deg)
0	-7.77	4.74	3.41
45	10.85	6.94	5.04
90	-1.71	-1.40	6.63
135	-13.22	5.78	9.75
180	21.88	-8.43	1.40
225	4.65	-10.24	2.18
270	-2.97	-7.08	7.78
315	-1.74	-15.43	1.24
Average	1.25	-3.14	4.68

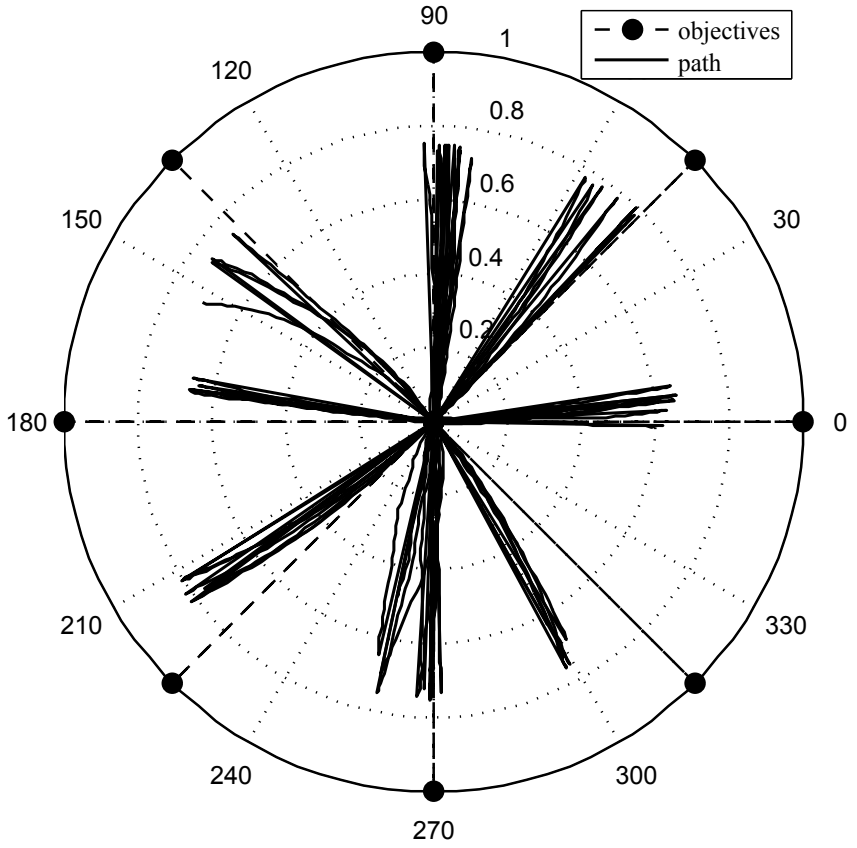


Figure 7.14: Direction deviation with LLC

It is clear from the graph that there is a lot of random behaviour during the execution of the manoeuvre. This is as a result of random disturbances in the mechanical system. The gears and wheel are not all the same and due

to mechanical imperfections disturbances are introduced. Another influence on the behaviour is the performance of the motor controller. It cause a slight rotation of the robot when the manoeuvre is commenced, this explains why the trajectory is not straight and has a slight curvature.

The third and fourth column of Table 7.1 lists the mean directional error and the standard deviation obtained during the testing. From this table it can be seen that there are clear directional errors when only the LLC is used. Although the encoders are used as feedback, slippage of the wheels and the behaviour of the motor controller cause significant errors in following the desired trajectory.

There is no correlation between the error obtained by **?** and the current study. The average mean error obtained by **?** is smaller than that obtained with the current system. However the maximum average error experienced by **?** is 21.88° which is considerably larger than the maximum average error of 15.43° obtained by this study. The error obtained by **?** was expected to be considerably larger because the robots were given a step input and no form of acceleration limiting was implemented. In the current study the robot were controlled to ensure that they never exceed their maximum acceleration.

The radial distances obtained in the various directions are displayed in figure 7.14. The radial distance vary depending on the direction in which the robot is driving, the results are listed in table 7.2. The best results are achieved in the 45° and the 225° directions. The largest mean error is that of the 0° and the 180° directions, the error is about 35% and is as a result of the geometry of the robot. In these directions the robot requires a higher torque to move resulting in the controller taking a longer time to eliminate the steady state error, which results in a lower average speed. The average standard deviation is 0.023 m which is a small value indicating that the results are consistent and depend on the direction travelled.

Table 7.2: Radial distance achieved by LLC

Desired direction	Mean radial error (m)	Standard deviation (m)	Mean error (%)
0	0.645	0.015	35.48
45	0.781	0.009	21.86
90	0.739	0.016	26.15
135	0.729	0.018	27.07
180	0.658	0.011	34.17
225	0.801	0.018	19.93
270	0.725	0.035	27.55
315	0.708	0.061	29.19
Average	0.723	0.023	27.67

The HLC was used to test the robot in the same way. The robot was sent positional commands for which the output was recorded. The feedback to the HLC is that of the estimator algorithms and therefore the robot was able to detect slippage on the wheels. The optimal PI parameters selected earlier in this chapter were used.

The results obtained are illustrated in figure 7.15. There is a clear improvement compared to the result obtained when only the LLC was used. The results obtained for the directional deviation is listed in table 7.3. The directional error is normally distributed and therefore it can be analysed using a t-distribution. The higher and lower bounds were determined with a 95% certainty. There is a 95% certainty that the maximum directional deviation will never be more than 2.6°.

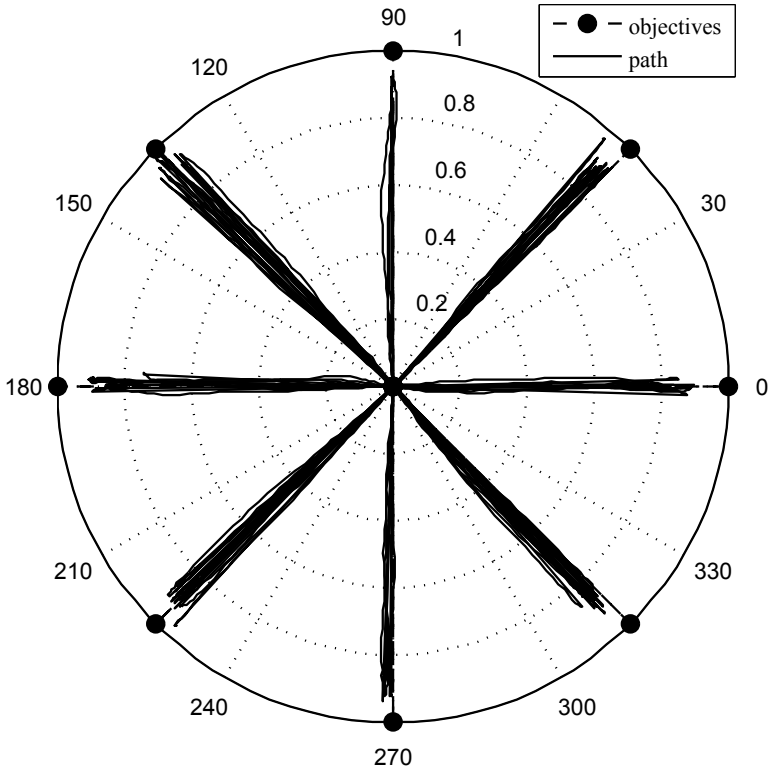


Figure 7.15: Direction deviation with HLC

The maximum standard deviation reduced from 9.75° to 1.70°, amounting to a reduction of 82.56%. The mean angular error reduced by 88.53%, from -3.14° to −0.36°. The HLC also reduced the maximum mean error by 87.49%, thus the HLC is successfully in greatly reducing the directional deviation of the robot.

The radial distances travelled is shown in table 7.4, the desired distance is 1 m. From the table it is clear that that robot is always short of this distance.

Table 7.3: Directional deviation of HLC

Desired direction	Upper bounds (deg)	Mean error (deg)	Lower bounds (deg)	Standard deviation (deg)
0	0.45	-0.05	-0.55	1.08
45	1.59	0.98	0.37	1.51
90	0.23	-0.32	-0.86	1.16
135	0.88	0.04	-0.81	1.70
180	0.76	0.08	-0.60	1.36
225	-0.22	-0.72	-1.23	1.20
270	-0.53	-1.00	-1.47	0.97
315	-1.26	-1.93	-2.60	1.31
Average	0.24	-0.36	-0.97	1.29

The corresponding direction yield similar results for the HLC and the LLC. The 135° direction achieved the smallest error of 7.86%. The mean error was reduced from 27.67% with the LLC to 10.49% when the HLC is used.

Table 7.4: Radial distance achieved by HLC

Desired direction	Mean error (m)	Standard deviation (m)	Mean error (%)
0	0.868	0.020	13.25
45	0.920	0.041	8.01
90	0.897	0.043	10.30
135	0.921	0.027	7.86
180	0.876	0.017	12.44
225	0.919	0.055	8.06
270	0.889	0.049	11.09
315	0.871	0.025	12.91
Average	0.895	0.035	10.49

This chapter illustrated that the movement of the robot can be accurately controlled when both the HLC and the LLC is situated on the robot. The KF has a clear advantage over a system which only uses an IMU. The PI controllers were calibrated to yield satisfactory results, although the motor controller design limited the performance. The HLC was not sufficiently compared to that of one running on an OFC, although the performance was good enough to warrant further research.

8. Conclusions

This Chapter will discuss the work that was done for this thesis and state to what extend the purpose of this project was accomplished. The findings of each chapter will be stated together with the problems encountered. The conclusions will end by discussing the implications of this project on the current research platform. This will be followed by recommendations for future work.

8.1. Conclusions

The purpose of this project is to develop a distributed control system for the SSL robots. This project builds on work done towards developing a robotic research platform at Stellenbosch University (**?**). The RoboCup SSL was selected as a research initiative which could be used to fulfil this ideal.

Previous work has been done to design and build a robotic platform, consisting of a omnidirectional robot with a SBC and a motor controller (**?**, **?**). **?** started by developing the research platform by building SSL robots with motor controllers which were able to execute basic movements using a LLC (encoder feedback). The robot designed by **?** was not within the specified size limit of the SSL and a second iteration robot was designed by a technician. **?** continued on this work by redesigning the motor controllers and the LLC.

Chapter 2 proposed a distributed control system with the HLC situated on the robot. This is in contrast to the centralised control system usually used by SSL teams. The distributed control system has the potential of greatly reducing the burden of the wireless communication system by reducing the amount commands that have to be sent. It also increases the performance of the HLC as the robot is able to react faster on disturbances such as slippage. The distributed control architecture necessitates the development of semi-autonomous robots to enable them to perform tasks on their own.

The first objective was to develop and test a centralised control system using overhead cameras. In order to fulfil this objective a wireless communication system is needed. **?** suggested using Player stage as it is adaptable and can be programmed using high level programming languages. Player consists of drivers and clients and the Player software architecture was discussed in chapter 3.

The hardware used for this project was discussed in chapter 4. A review was given of the existing hardware that was used. This included the on-board sensors which were used to give the robot some form of autonomy. It was decided that a 6 DOF IMU would be used to provide inertial measurements of the robot's movement. Inertial measurements have the advantage of detecting slippage which is a common problem in the SSL robots.

Testing of the centralised control system requires a fully functional LLC. Unfortunately **?** did not have sufficient time to test and implement the software on the motor controller. The motor controller software was therefore updated and corrected. Once this was done the LLC was calibrated to ensure optimal robot movement. The Player drivers and clients were updated to enable the OFC to communicate with the motor controllers.

Trajectory tracking consist of two parts, path planning and trajectory following. The path planning needs to be done in such a way to ensure that a robot operates within its physical limits while avoiding slippage. The algorithm developed by **?** was implemented in C++ and testing using a soccer robot. The only drawback of the algorithm is that it not able to compute rotational profiles and only generates velocity profiles for the translational DOF. The trajectory following entails the control of the robot in such a way to accurately follow the desired path. This required the implementation of both the HLC and the LLC.

In a centralised control architecture the only feedback to the HLC is that of the vision system. The shared SSL-vision system was set up at the robot soccer field at Stellenbosch University. After this the Player clients were modified to access the vision information and control the robots based on this information. Thus fulfilling the first objective of this project by producing a centralised control system utilizing the SSL-vision system.

The second objective of this project was to develop a distributed control system which uses information obtained from the vision system as well as sensors on the robot. In order to give the robots some form of autonomy the IMU was implemented. This required the modification of the Player drivers of the robot. The drivers were required to perform sensor fusion of the IMU and vision data.

The sensor fusion was implemented using estimator algorithms. The Kalman filter was selected for this purpose. The basic operation of the KF is explained after which KFs are designed to estimate the position, velocity and orientation of the robot. It was decided to obtain the angular velocity directly from the IMU. The sensor fusion was effectively implemented using the KF and the estimator results are shown in chapter 6.

The estimator output was used as the feedback for the HLC of the distributed control system. The HLC was implemented using a PID controller. The input to the HLC was a desired position and this command could be executed using the distributed control system, fulfilling the second objective.

The last chapter in this report compares the performance of using differ-

ent sensor feedbacks for the HLC. The ability of a robot to follow a square trajectory when using the IMU as feedback for the HLC is compared to that of a KF using the vision data as well as the IMU. The experimental results show that the proposed method is able to accurately measure the state of the robot and significantly improve the performance of the robot control.

The performance of the distributed control system is quantified and measured. The ability of the robot to execute desired position commands is experimentally tested. The comparison between a robot using the LLC and a robot using the both the LLC and the HLC is made. The results indicate that the HLC yields a large improvement in the robots ability to follow a specified trajectory.

The progress made towards developing a robotic research platform at Stellenbosch University. Player drivers were developed for the robots, that provide all the functionality required by the SSL. These drives were written in C++ and can be easily modified to provide additional functionality. Using the IMU, vision system and encoders the robots are now able to drive toward a desired destination. Therefore the system can now be used to perform research in other robotic disciplines such as AI or behaviour strategies.

Summarizing this report found that a distributed control architecture can be used effectively to implement velocity and position control on a SSL robot. Kalman filters can be used to provide accurate positional and rotational data improving the performance of the control system and reducing the burden on the wireless network.

8.2. Recommendations for Future Work

This section discusses recommendations for the continuation of this project towards the goal of developing a team of SSL robots that will be able to compete against other teams. Additionally this recommendations would assist in expanding and further improving the research platform at Stellenbosch University.

There were two factors limiting the effectiveness of this research. Firstly the design of the current SSL robot made it susceptible to manufacturing imperfections and irregularities which led to a system where the performance is limited by the mechanical abilities of the robot. The second limiting factor was the delay between when the motors received their velocity commands. This was as a result of the design of the motor controller. The controller was designed to operate in a daisy chain, however as a result of the software design the speed of the controller was effectively quartered. This severely limited the performance of the robot. Therefore the author would recommend redesigning the mechanical drive train as well as the redesigning the software implementation of the motor controllers.

Furthermore the system would benefit if the following task were performed. Currently the path planning is only done for translational motion of the robots. The agility of the robots would be improved if path planning was done for the rotational motion as well. The path planning also does not include any obstacle avoidance, which is a necessity in multiple robot environments. This is also a topic for future work.

The SSL-vision system was set up at the field and consists of all the required hardware. However, the system was only set up to use one half of the field as the merging of the vision data was not done. This will be required if the whole field is to be utilised. The raw vision data was used in the Kalman filter. This data is usually filtered using EKF before it is used for control purposes. For this project the KF on the robot was used and this was not required. This is however needed to accurately estimate the position and velocity of the ball and will be required in the future.

A. Software Diagrams

A.1. Driver

The UML class diagram of the Driver is displayed in figure A.1. The main class is SRDriver which process all the messages comming over the network. The other classes used by SRDriver is descussed in section 5.4.

A.2. Client

The UML class diagram of the Client is displayed in figure A.1. This does not show the GUI that was used to control the robots. An instance of the Control class is used in the GUI to control the robots. The control class is modular and can be used in a GUI or with a CLI

Figure A.1: UML class diagram of Driver software

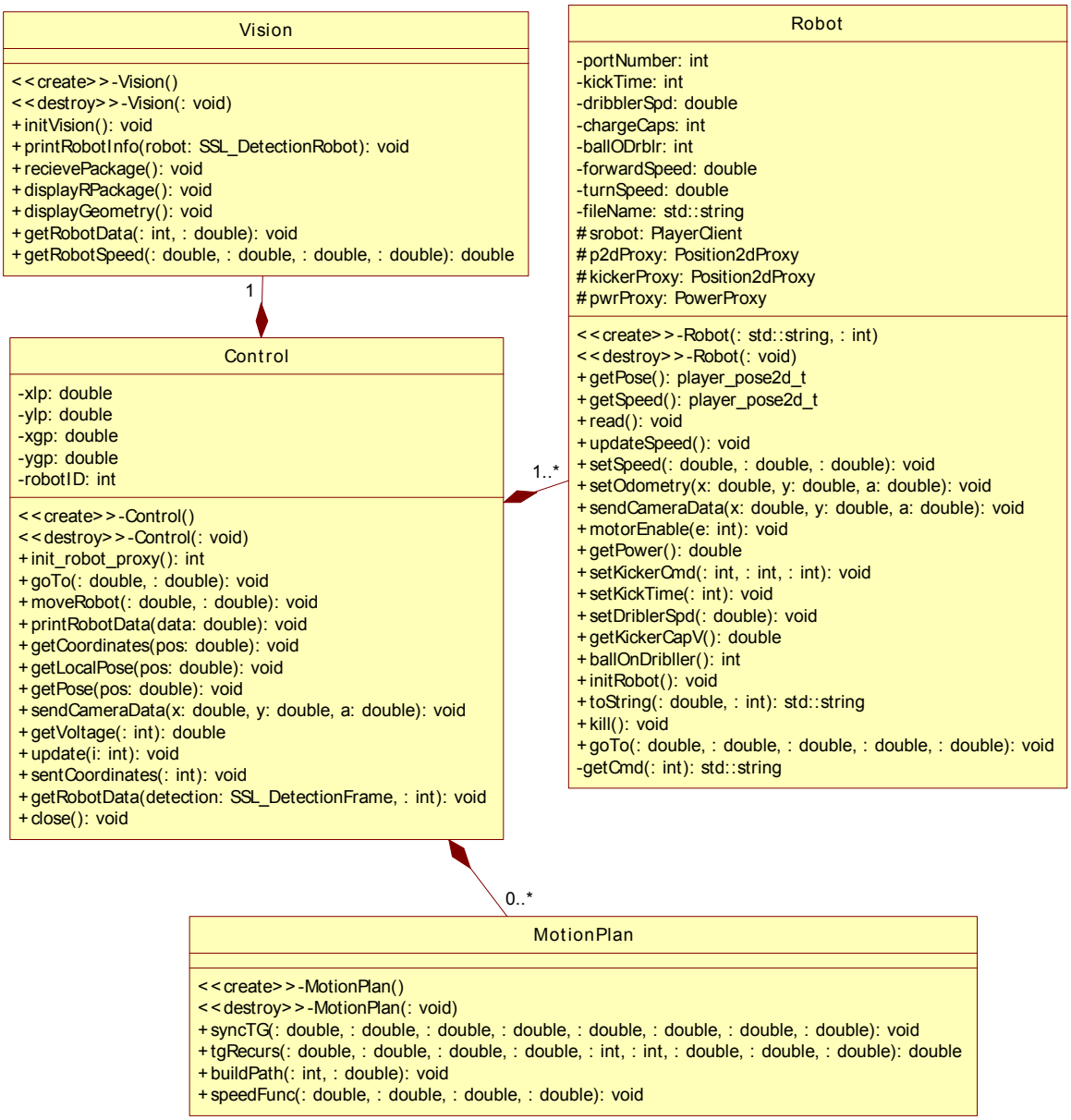


Figure A.2: UML class diagram of Client software

B. SSL Robot Dynamics

This chapter derives a dynamic model for a soocer robot. Then the theoretical maximum velocity of the soccer robots at Stellenbosch University is calculated.

B.1. Dynamic Model of SSL Robot

The dynamic model of the SSL robot can be used to simulate and test the control of the robots. The model derived in this section is based on the derivation of **?**. **?** derived the dynamic model with the assumption that all the wheels are 90° apart. This model does not make this assumption and can be used to obtain the dynamic model of a omnidirectional robot with n wheels situated at angles θ_1, θ_n, ... ,θ_n. The platform of the omnidirectional robot is depicted in figure B.1.

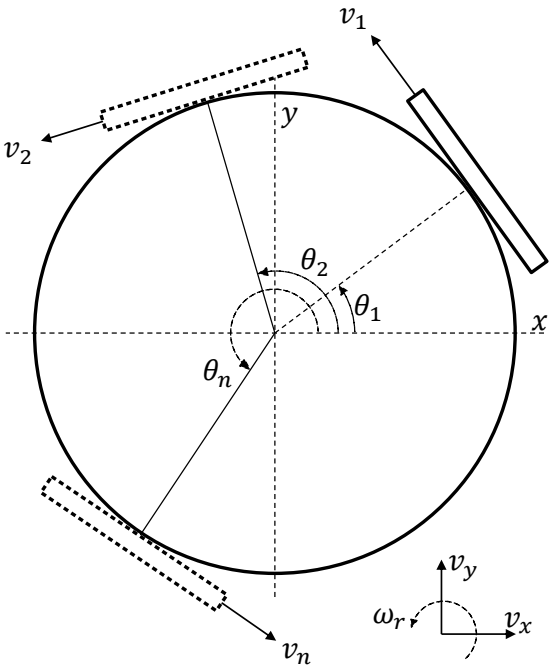


Figure B.1: Omnidirectional platform

The individual wheel velocities are related to the robot velocities using the geometry of the robot.

$$\begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} -sin\theta_1 & cos\theta_1 & R_r \\ \vdots & \vdots & \vdots \\ -sin\theta_n & cos\theta_n & R_r \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_r \end{bmatrix} \tag{B.1}$$

where v_1 to v_n are the wheel velocities and v_x, v_y and ω_r are the robot velocities as illustrated in figure B.1.

Using Newton's second law the dynamics of the robot is related to the nett forces acting thereon.

$$\begin{aligned} m\frac{dv_x}{dt} &= \Sigma F_x - F_{Bx} - F_{Cx} \\ m\frac{dv_y}{dt} &= \Sigma F_y - F_{By} - F_{Cy} \\ J\frac{d\omega}{dt} &= \Sigma T\omega - T_{B\omega} - T_{C\omega} \end{aligned} \tag{B.2}$$

where

$$\begin{aligned} F_{Bx} &= B_x \cdot v_x, \quad & F_{Cx} &= C_x \cdot sign(v_x) \\ F_{By} &= B_y \cdot v_y, \quad & F_{Cy} &= C_y \cdot sign(v_y) \\ T_{B\omega} &= B_\omega \cdot \omega_r, \quad & T_{C\omega} &= C_\omega \cdot sign(\omega_r) \end{aligned} \tag{B.3}$$

In steady state the model of a DC motor becomes (**?**),

$$\begin{aligned} E_n &= R_m \cdot i_n + K_v \cdot (w_m)_n \\ (T_m)_n &= K_t \cdot i_n \end{aligned} \tag{B.4}$$

However, the wheel torque can be related to the motor current by using equation B.4 as

$$\begin{aligned} T_n &= F_n \cdot r_w \\ &= l \cdot K_t \cdot i_n \end{aligned} \tag{B.5}$$

this is rewritten yielding

$$i_n = \frac{F_n \cdot r_\omega}{lK_t} \tag{B.6}$$

Combining equation B.4 and B.6 the voltage of the motor is related to the force induced by the wheel and the speed at which the motor is rotating.

$$E_n = \frac{R_m r_w}{lK_t} \cdot F_n + K_v \cdot (w_m)_n \tag{B.7}$$

, but the motor velocity is related to the wheel velocity as

$$(\omega_m)_n = l \cdot \omega_n \tag{B.8}$$

Equation B.7 and B.8 is then combined and the result is rewritten to make the wheel force the subject

$$\begin{aligned} F_n &= B \cdot E_n - A \cdot w_n \\ A &= \frac{l^2 K_t K_v}{R_m {r_w}^2} \\ B &= \frac{l K_t}{R_m r_w} \end{aligned} \tag{B.9}$$

The angular velocity of the wheel is related to the wheel velocity by the radius of the wheel and by combing this with equation B.1 the following results are obtained

$$\begin{aligned} \omega_n &= v_n \cdot \frac{1}{r_w} \\ \omega_n &= (-sin\theta_n \cdot v_x + cos\theta_n \cdot v_y + R_r \cdot \omega_r) \cdot \frac{1}{r_w} \end{aligned} \tag{B.10}$$

The nett forces as a result of the wheels can be calculated as shown in the following equations,

$$\begin{aligned} \Sigma F_x &= -F_1 sin\theta_1 - F_2 sin\theta_2 - \cdots - F_n sin\theta_n \\ \Sigma F_y &= F_1 cos\theta_1 + F_2 cos\theta_2 + \cdots + F_n cos\theta_n \\ \Sigma T &= (F_1 + F_2 + \cdots + F_n) \cdot R_r \end{aligned} \tag{B.11}$$

These equations can be combined with equation B.10 and B.13. This is done for ΣF_x, but can be done in a similar way for ΣF_y and ΣT.

$$\begin{aligned} \Sigma F_x &= -F_1 sin\theta_1 - F_2 sin\theta_2 - \cdots - F_n sin\theta_n \\ \Sigma F_x &= -(B \cdot E_1 - A \cdot w_1) sin\theta_1 - \cdots - (B \cdot E_n - A \cdot w_n) sin\theta_n \\ \Sigma F_x &= -[B \cdot E_1 - A \cdot (-sin\theta_1 \cdot v_x + cos\theta_1 \cdot v_y + R_r \cdot \omega_r) \cdot \frac{1}{r_w}] sin\theta_1 - \\ &\quad \cdots - [B \cdot E_n - A \cdot (-sin\theta_n \cdot v_x + cos\theta_n \cdot v_y + R_r \cdot \omega_r) \cdot \frac{1}{r_w}] sin\theta_n \\ \Sigma F_x &= -D(sin\theta_1^2 + \cdots + sin\theta_n^2) \cdot v_x \\ &\quad + D(sin\theta_1 cos\theta_1 + \cdots + sin\theta_n cos\theta_n) \cdot v_y \\ &\quad + D R_r (sin\theta_1 + \cdots + sin\theta_n) \cdot \omega_r \\ &\quad - B sin\theta_1 E_1 - \cdots - B sin\theta_n E_n \end{aligned} \tag{B.12}$$

with

$$D = \frac{l^2 K_t K_v}{R_m {r_w}^2} \tag{B.13}$$

Once similar equations to B.12 has been derived for ΣF_y and ΣT. They can be combined with equations B.2 and B.3, yielding the dynamic model of the robot.

$$\begin{aligned}\begin{bmatrix} a_x \\ a_y \\ \dot{\omega} \end{bmatrix} &= \begin{bmatrix} -D_2 \cdot C_1 - B_x & D_2 \cdot C_2 & D_2 R_r C_3 \\ D_2 \cdot C_2 & -D_2 \cdot C_4 - B_y & -D_2 R_r C_5 \\ \frac{DR_r}{J} C_3 & -\frac{DR_r}{J} C_5 & -\frac{4DR_r^2 - B_\omega}{J} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_r \end{bmatrix} \\ &+ \begin{bmatrix} -\frac{B}{m} sin\theta_1 & -\frac{B}{m} sin\theta_2 & \cdots & -\frac{B}{m} sin\theta_n \\ \frac{B}{m} cos\theta_1 & \frac{B}{m} cos\theta_2 & \cdots & \frac{B}{m} cos\theta_n \\ \frac{BR_r}{J} & \frac{BR_r}{J} & \cdots & \frac{BR_r}{J} \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \\ &+ \begin{bmatrix} C_x \cdot sign(v_x) \\ C_y \cdot sign(v_y) \\ C_\omega \cdot sign(\omega_r) \end{bmatrix}\end{aligned} \tag{B.14}$$

where

$$\begin{aligned} C_1 &= sin\theta_1^2 + \cdots + sin\theta_n^2 \\ C_2 &= sin\theta_1 cos\theta_1 + \cdots + sin\theta_n cos\theta_n \\ C_3 &= sin\theta_1 + \cdots + sin\theta_n \\ C_4 &= cos\theta_1^2 + \cdots + cos\theta_n^2 \\ C_5 &= cos\theta_1 + \cdots + cos\theta_n \\ D &= \frac{l^2 K_t K_v}{R_m {r_w}^2} \\ D_2 &= \frac{l^2 K_t K_v}{R_m {r_w}^2 m} \\ B &= \frac{l K_t}{R_m r_w} \end{aligned} \tag{B.15}$$

B.2. Maximum Robot Velocity

The force required to move the robot is calculated using the following equation

$$\begin{aligned} F_R &= m \cdot g \cdot \mu_f \\ &= 3 \cdot 9.81 \cdot 1.37 \\ &= 40.32\ N \end{aligned} \tag{B.16}$$

This value was calculated assuming the weight of the robot is 3 kg and using the static friction obtained by **?**. The least amount of wheels will be used when the motor is driving at equal speeds in the x and y direction. Thus the

maximum velocity will be calculated if only two wheels are used. The torque required from a motor during this conditions is

$$\begin{aligned} T_m &= \frac{1}{l} \cdot \frac{F_R \cdot r_w}{2} \\ &= \frac{1}{9} \cdot \frac{40.32 \cdot 0.035}{2} \\ &= 78.4\ mNm \end{aligned} \tag{B.17}$$

This is very close to the stall torque of the motor of 80 mNm (**?**). However, the motor will drive the robot as long as the weight of the robot stays below 3 kg.

The recommended maximum torque for a motor is 16 mNm, thus this is far below the the torque required to get the robot moving. Through various tests it was seen that the velocity of the robot is not limited by the power of the motors, but more by inaccuracies in the mechanical design and the electronic design. Therefore the maximum velocity is not calculated here and the velocity is restricted to a value below 0.6 ms to ensure adequate robot performance.

C. Optical Sensor

This chapter explains the design of a optical sensor which measures the velocity of the robot.

C.1. Introduction

The position, orientation and angular velocity of the robot is directly observable. However, the translational velocity of the robot is obtained using acceleration and positional data. This information is dependent on the performance of the estimator used, and therefore requires more effort and time to determine. It would be desirable if there was a way of measuring the velocity directly.

An optical mouse sensor measures the linear translational velocities of an object. **?** stated that the SSL soccer robots operate up to 3.5 m/s. The ADNS-9500 LaserStream™ Gaming Sensor which has a Vertical-Cavity Surface-Emitting Laser (VCSEL) and sensor on one single package (**?**). The sensor is able to measure speeds of 3.81 m/s. Thus this sensor would be able to measure the robot velocity and the design will be desrcibed in this rest of this chapter.

C.2. Mechanical Design

There are high positional tolerances regarding the sensor positions. The sensor could be used in one of either two setups. Firstly the sensor could be placed exactly in the middle of the robot to prevent inaccurate measurements as a result of orientation. The second possibility is to use two sensors spaced equally far apart from the center these sensors could be used to determine the rotational and translational velocity of the robot.

It was decided to implement one sensor and based on the results an additional sensor would be purchased. Because this project will not be using the kicker system, it was decided to design the sensor assembly to fit within the designated space of the kicker system. Various concepts were designed and their advantages and disadvantages were discussed. There were two important

design requirements. Firstly the sensor unit needed to be as small as possible and secondly the sensor has very strict height requirements. The height of the sensor above the measuring surface needs to be between 2.18 mm and 2.62 mm, that is a tolerance of ±0.022 mm (**?**).

Figure C.1 shows a CAD representation of the sensor mounted on the robot. The designed sensor assembly is seen in figure C.3. The lower platform glides on the field, while it is free to move in a vertical direction and has springs holding it down. The unit was manufactured and assembled at Stellenbosch University. The ADNS-6190-002 LaserStream™Gaming Round Lens was purchased and fitted in the bottom platform according to the prescribed specifications.

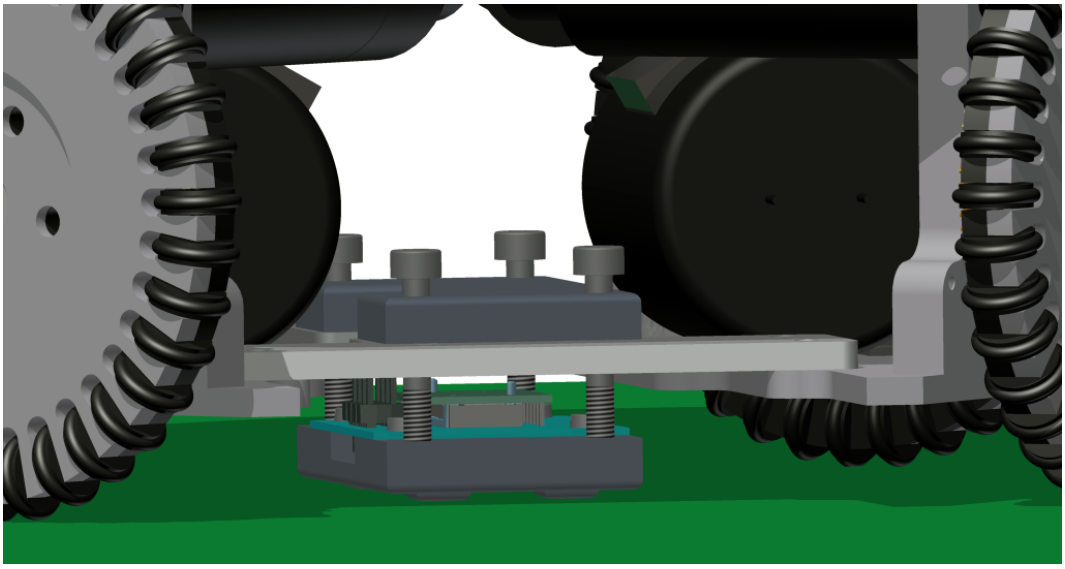


Figure C.1: Optical sensor placement

C.3. Electronic Design

The electronic design requires a circuit to operate the gaming sensor. The ADNS-9500 comes with application circuits which can be used to control the sensor, theses circuits are all focussed on the usage of the sensor in a mouse (**?**). A PCB design of a circuit which uses the ADNS-9500 was found on the internet (**?**). This PCB design was modified to adapt for the mechanical design of the sensor unit. The modified PCB is seen in figure C.2. The sensor communicates with the main controller using the SPI interface.

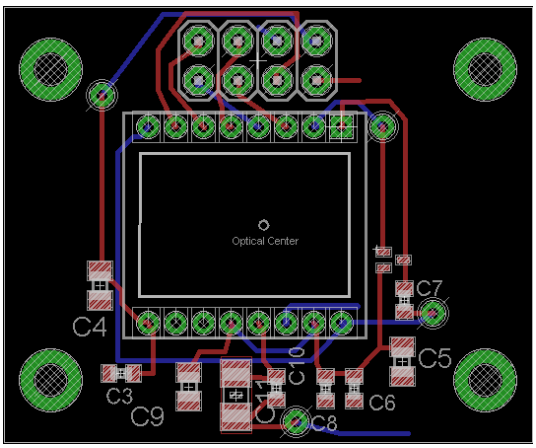


Figure C.2: Optical sensor PCB design

C.4. Remarks

The optical sensor can be used to measure the translational velocities of a robot. However, this sensor was never tested. This was a result of a various factors. Firstly the final design was quite large and the robot did not have sufficient space for the sensor and the kicker module. Thus there would have been no future usage for the sensor once the kicker module was implemented.

The second factor is that **?** did implement an optical sensor with SSL robots. However, they found that the robot's rotation had an influence on the measured velocity, which resulted in inaccurate measurements. Additionally due to time constraints it was decided to not implement the sensor although the electronic and mechanical design had been assembled. The interface between the main controller and sensor had not yet been implemented.

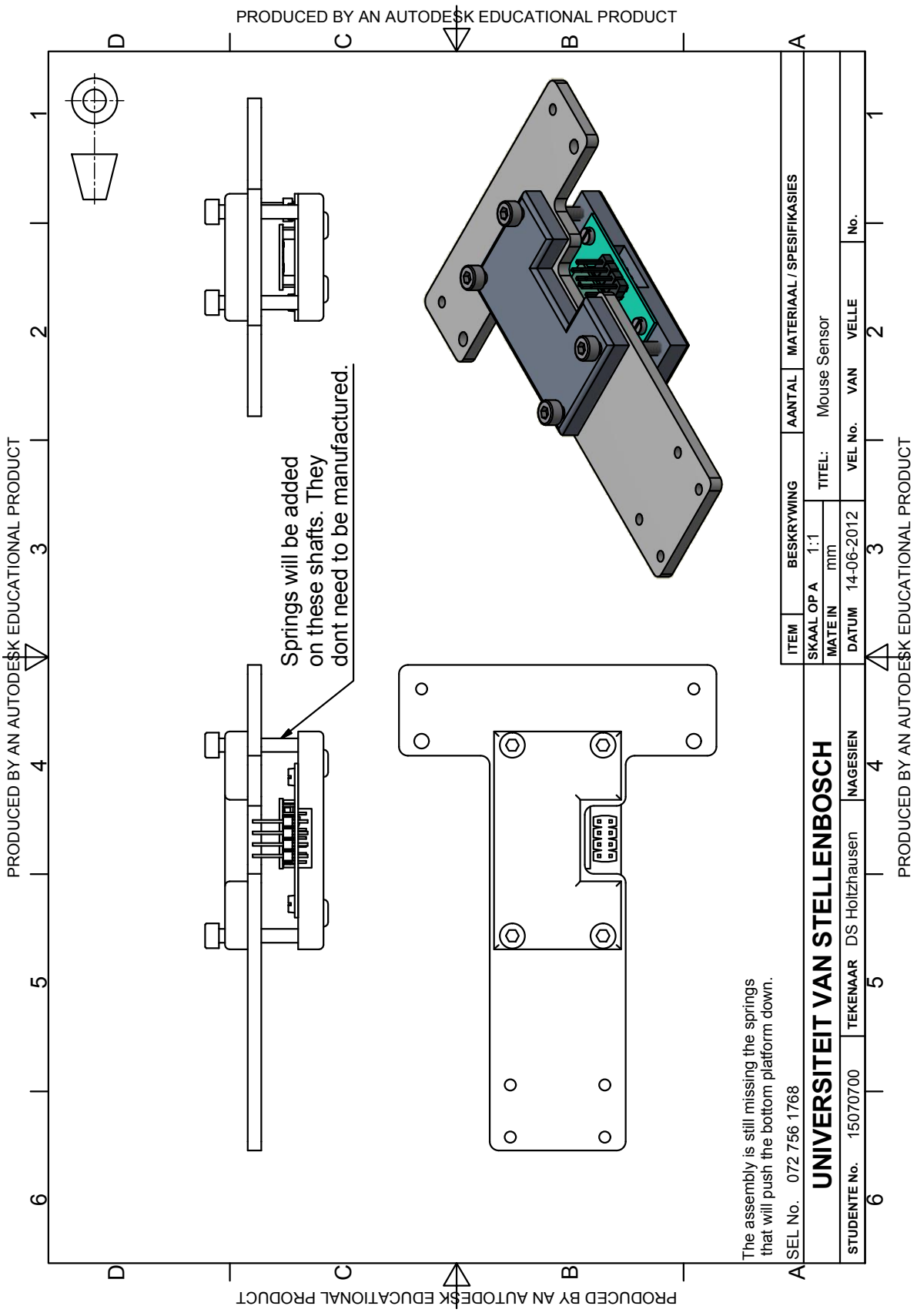


Figure C.3: Omnidirectional platform

D. Kalman Filter Simulations

D.1. Translational Kalman Filter

The developed Kalman filters were tested and calibrated using MATLAB simulations. The simulations made it easy to observe the effect of varying certain parameters. This chapter will focus on the translational KF, although similar simulations were run for the angular KF and CF.

The time update and measurement update equations of the KF were derived in section 6.1.2 and for the implemented translational KF become

$$\begin{aligned} {x_p}^k &= Fx^{k-1} + Gu^{k-1} \\ &= \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} x^{k-1} + \begin{bmatrix} T_s^2 \\ T_s \end{bmatrix} u^{k-1} \end{aligned} \tag{D.1}$$

$$\begin{aligned} x^k &= {x_p}^k + K^k(y^k - H{x_p}^k) \\ &= {x_p}^k + K^k(y^k - \begin{bmatrix} 1 & 0 \end{bmatrix} {x_p}^k) \end{aligned} \tag{D.2}$$

the system input u^{k-1} is the accelerometer data and the measured system output y^k is obtained from the vision system. This data was not generated theoretically, rather the actual data was recorded while the robot was performing a manoeuvre. Therefore the data contains the actual disturbances and noise present in the system. This ensured that the estimates obtained from the simulation will be the same that obtained from KF on the robot.

The system input is which is the acceleration data can be seen in figure D.1. From the figure it is clear that there is distinct acceleration zones. The first is negative acceleration, followed by zero acceleration and ending with acceleration in the positive direction. The measured system output is obtained from the actual vision data. The position data is shown in figure D.2.

The simulation is run to obtain the position and velocity estimate. The position estimate is shown in figure D.3 and the velocity estimate in figure D.4. The position estimate results in the same final position as that obtained from the camera data. Although the velocity estimate ends at a value close to zero it does not completely reach zero indicating that a small error in the velocity estimate.

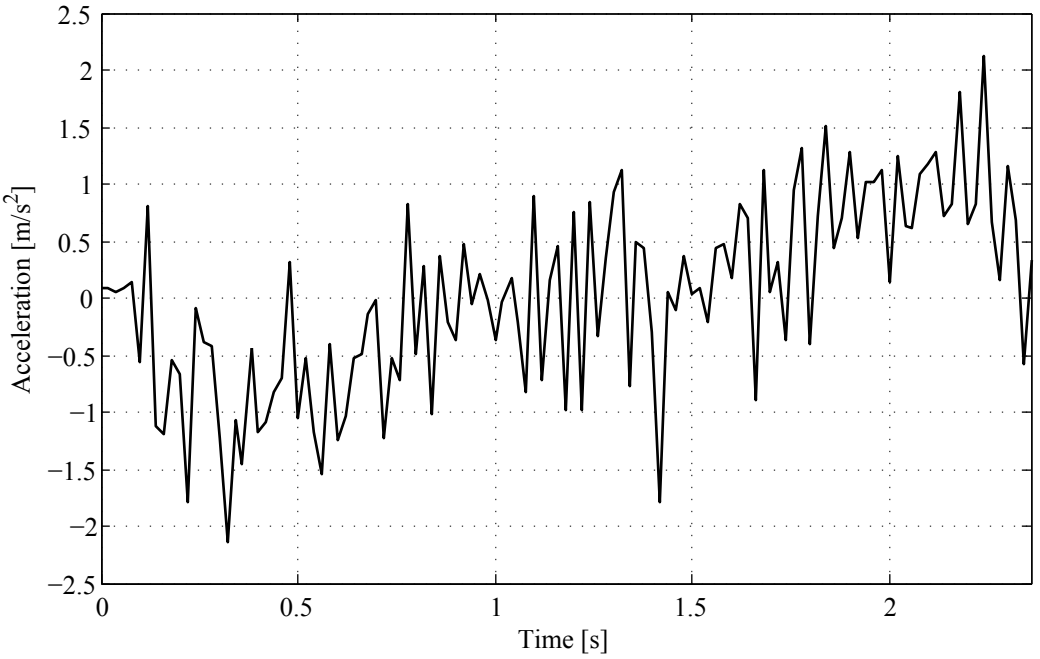


Figure D.1: The input to the translational KF is the acceleration obtained from the IMU.

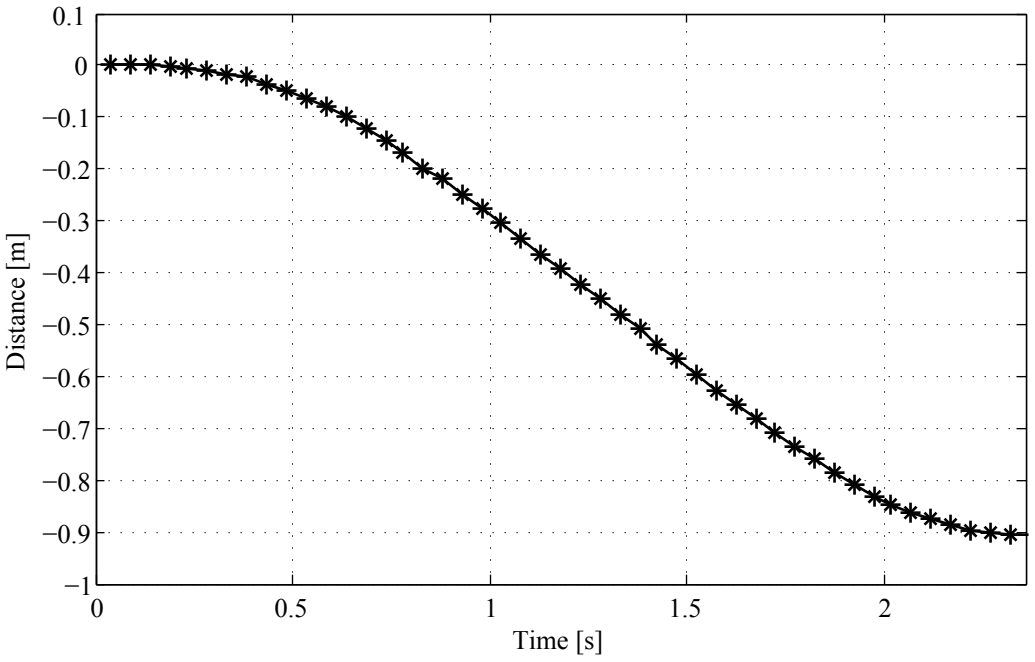


Figure D.2: The measured system output obtained from the vision system.

D.2. MATLAB Code

The simulations were run using MATLAB. The code is seen below. The function is called `kalmanDilterRData` and has four inputs. `Data` is the position

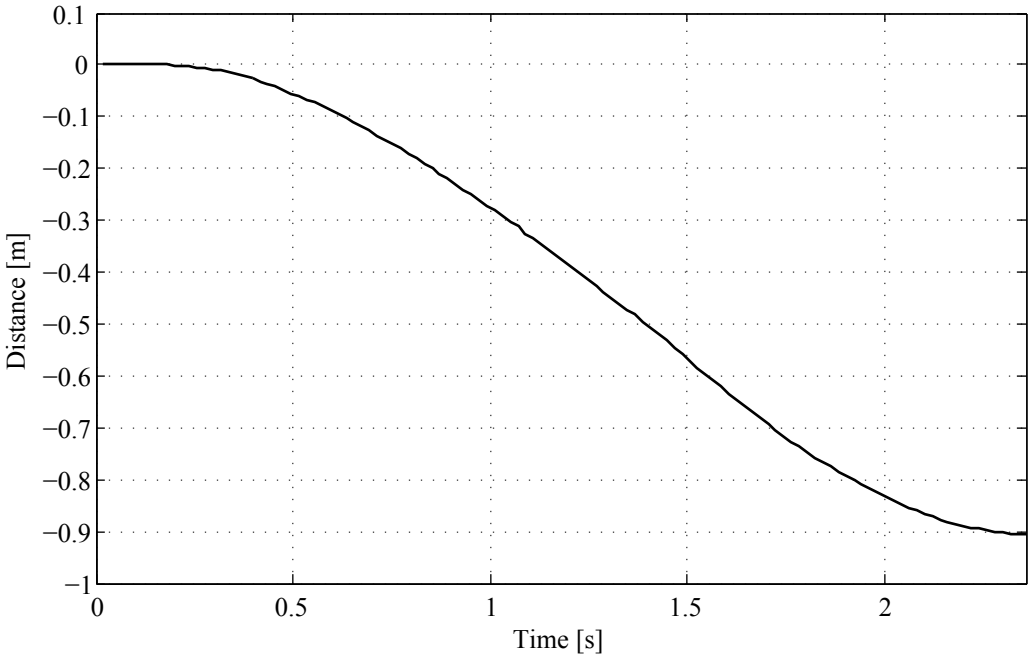


Figure D.3: The position estimate obtained from the KF.

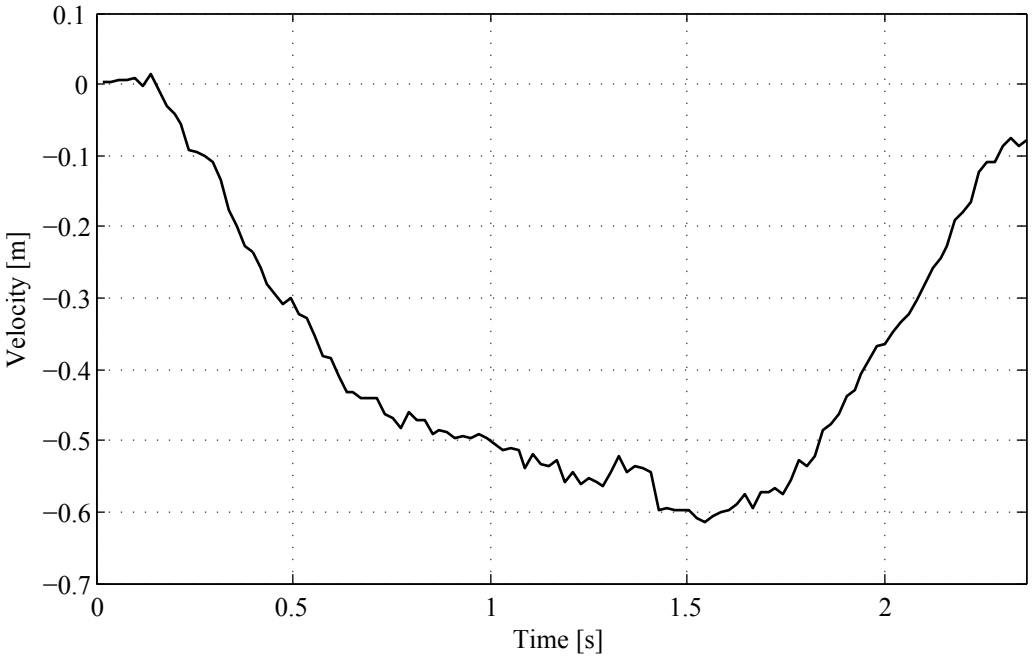


Figure D.4: The velocity estimate obtained from the KF

and acceleration data recorded on the robot. The variable `dT` is the sampling time of the acceleration data. The variables `ms` and `qs` are the measurement noise and process noise spectral density that will be used for the filter. The

outputs are three arrays, one with the time and the other two with the position and velocity estimate respectively.

```
function [tx Xx Xv] = kalmanFilterRData(Data,dT,ms,qs)
%kalmanFilterRDataN(Data,dT,ms,qs) takes the
%information obtained from the robot and
%simulates the estimator output. The inputs
%are the robot %data (Data), the sampling
%time (dT), the measurement noise (ms) and
%the process noise spectral density (qs)

%Indicates wheter it is vision or IMU data
pU = Data(:,1);
%Time data
tx = Data(:,2)';
%Data obtained from robot
inputData = Data(:,5);

%initial guess for state space vector
x = [0 0]';

%initial guess for Covariance matrix
P = [1 0;0 1];
%2 by 2 identity matrix
I = [1 0; 0 1];
%encodes dynamics of the system
F = [ 1 dT; 0 1];
%describes how inputs drive dynamics
G = [dT^2; dT];
%describes how state vectors are mapped into
H = [1 0];
%Store the position estimate
Xx = [];
%Store the velocity estimate
Xv = [];
%covariance matrix for process noise
Q = qs*[(dT^3)/3 (dT^2)/2; (dT^2)/2 dT];
%covariance matrix for measurement noise w(k)
R = ms;

%Returns the size of the data array
[m n] = size(Data);

for k = 1:1:m

    if(pU(k))% //// Propogation ////
        %x^(k+1|k)
        x = F*x + G*inputData(k);
        %covariance matrix P(k+1|k)
        M = F*P*F' + Q;
    else      % //// Update ////
        %Calculate Kalman gain
        K = M*H'/(H*M*H' + R);
```

```
        %Calculate corected state estimate
        x = x + K*(inputData(k)-H*x);
        %Calculate covariance matrix
        P = (I - K*H)*M;
        %Enforce symetry
        P = P/2+P'/2;
    end
    %Store the position estimate
    Xx = [Xx x(1)];
    %Store the velocity estimate
    Xv = [Xv x(2)];
end

end
```

kalmanFilterRData.m

D.3. C++ Code

The code that is used to implement the translational KF is shown in this section. This is not used for any simulation purposes and purely shows the implementation of the KF in C++. The functions are called from the Player driver whenever new acceleration or vision data is available. The Eigen C++ library is used to preform the matrix and vector algebra (**?**).

```
#ifndef KALMAN_H_INCLUDED
#define KALMAN_H_INCLUDED
#include <time.h>
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <Eigen/Dense>
#include <iostream>
#include <fstream>

using namespace Eigen;
using namespace std;

class Controler
{
    double invTerm;    //Used to calculate the inverse
    Eigen::Matrix2d P;  //Covariance matrix
    Eigen::Matrix2d F;  //Matrix that encodes system dynamics dt =
        0.02
    Eigen::Matrix2d Ft; //Transpose of F
    Eigen::Matrix2d I;  //2 by 2 Identity matrix
    Eigen::Vector2d G;  //Describes how input drives dynamics
    Eigen::Vector2d G2; //Describes how input drives dynamics
    Eigen::RowVector2d H;//Describes how state vectors are mapped
       into outputs
```

```
24      Eigen::Vector2d Ht; //Transpose of H
25      double qs;       //Process noise estimate
26      Eigen::Matrix2d Q;  //Covariance matrix for process noise
27      double ms;       //Measurment noise estimate
28      double R;      //Covariance for measurement noise
29      Eigen::Matrix2d M;  //Covariance matrix
30      Eigen::Vector2d K;  //Stores the inverse of a term
31      Eigen::Vector2d x;  //State vector
32      double y;      //Output vector
33
34  public:
35      EIGEN_MAKE_ALIGNED_OPERATOR_NEW
36      Controler();
37      ~Controler();
38      void initializeKalman(double MS, double QS, double xi, double
            xdoti);
39      void propagateKalman(double xr[],double,double);
40      void updateKalman(double xr[],double);
41      void closeKalman(void);
42
43  };//end class
44
45  #endif
```

Controler.h

```
1   #include "Controler.h"
2
3   using namespace Eigen;
4
5   void Controler::initializeKalman(double MS, double QS, double xi,
        double xdoti)
6   {
7       ms = MS;
8       qs = QS;
9       x << xi, xdoti;
10      invTerm = 0;
11      y = 0;
12
13      double dT = 0.02;
14
15      //Initial quess fot covariance matrix
16      P << 1, 0, 0 , 1;
17      I << 1, 0, 0 , 1;
18      //System dynamics
19      F << 1, dT, 0 ,1;
20      Ft = F.transpose();
21      //Input dynamics
22      G << dT^2, dT;
23      //Outputs
24      H << 1,0;
25      Ht << 1,0;
26      //Covariance matrices
```

```
27      Q << qs*pow(dT,3)/3, qs*pow(dT,2)/2,
28                  qs*pow(dT,2)/2,    qs*dT;
29      R = ms;
30      M << 0, 0, 0 , 0;
31      K << 0, 0;
32  }
33
34  void Controler::closeKalman()
35  {
36  }
37
38  // u is the input vector, the measured acceleration
39  void Controler::propagateKalman(double xr[],double u,double dt)
40  {
41      double xp = 0;
42      double vp = 0;
43      //Propagation
44    x = F*x + G*u;
45    M = F*P*Ft + Q;
46
47      xr[0] = x(0);
48      xr[1] = x(1);
49      // std::cout <<"x = "<< x <<"\n\n";
50      M = F*P*Ft + Q;
51      // std::cout <<"M = "<< M <<"\n\n";
52  }
53
54  void Controler::updateKalman(double xr[],double cameray)
55  {
56      //Update
57      y  = cameray; //System output
58      //K term with inverse
59      invTerm = H*M*Ht+R;
60      //std::cout <<"Ni = "<< Ni <<"\n\n";
61      K = M*Ht/invTerm;
62      //Or used previously calculated values
63      x = x + K*(y-H*x);
64      xr[0] = x(0);
65      xr[1] = x(1);
66      //std::cout <<"x = "<< x <<"\n\n";
67      P = (I - K*H)*M;
68      P = P/2 + P.transpose()/2;
69      //std::cout <<"P = "<< P <<"\n\n";
70  }
71
72  Controler::Controler()
73  {
74      return;
75  }//end constructor.
76
77  Controler::~Controler()
78  {
```

```
79 }
```

Controler.cc

E. Coordinate Transformation

The robots coordinate frame needs to be transformed to that of the global reference frame. This is done using the rotation matrix described using the roll pitch and yaw around the reference frame. The yaw (ψ) is the rotation around the x-axis, pitch (β) is the rotation around the y-axis and roll (ϕ) is the rotation around the z-axis.

The rotation matrix will be used to transform the IMU data to the coordinate frame of the camera. The respective coordinate frames are shown in figure E.1. The IMU coordinate frame must be rotated around the z axis by an angle of $\pi/2$ to get it to the same coordinate frame as that of the motor controller. Then it must be rotated by an angle of π around the x-axis to correctly orientate it with the global coordinate frame of the camera. There is not rotation around the y-axis. Therefore the pitch is $\beta = 0$ and the yaw $\psi = \pi$. The robot will be driving on the field and only rotate around the z-axis. Therefore the roll is defined as $\phi = \theta + \pi/2$. This is the initial rotation of $\pi/2$ as well as the measured orientation of the robot (θ).

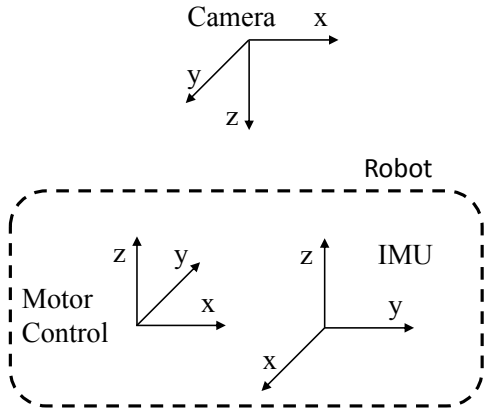


Figure E.1: Reference frames of the robot, camera and IMU

The roll, pitch and yaw values are used to derive the rotation matrix. The

derivation is shown in the following equation,

$$
\begin{aligned}
R &= R_{z,\phi} \cdot R_{y,\beta} \cdot R_{x,\psi} \\
R &= \begin{bmatrix} cos\phi & -sin\phi & 0 \\ sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\psi & -sin\psi \\ 0 & sin\psi & cos\psi \end{bmatrix} \\
R &= \begin{bmatrix} cos(\theta+\pi/2) & -sin(\theta+\pi/2) & 0 \\ sin(\theta+\pi/2) & cos(\theta+\pi/2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos0 & 0 & sin0 \\ 0 & 1 & 0 \\ -sin0 & 0 & cos0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\pi & -sin\pi \\ 0 & sin\pi & cos\pi \end{bmatrix} \\
R &= \begin{bmatrix} cos(\theta+\pi/2) & -sin(\theta+\pi/2) & 0 \\ sin(\theta+\pi/2) & cos(\theta+\pi/2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \\
R &= \begin{bmatrix} cos(\theta+\pi/2) & sin(\theta+\pi/2) & 0 \\ sin(\theta+\pi/2) & -cos(\theta+\pi/2) & 0 \\ 0 & 0 & -1 \end{bmatrix}
\end{aligned}
\tag{E.1}
$$

The rotation matrix transforms the acceleration measurements in the x, y and z directions to the reference coordinate frame. However, only the acceleration in the x and y direction is used. The rotation matrix therefore reduces to a 2 by 2 matrix. The transformation from the IMU's coordinate frame to the global reference frame is demonstrated in the following equation,

$$
\begin{bmatrix} a_{x,g} \\ a_{y,g} \end{bmatrix} = \begin{bmatrix} cos(\theta+\pi/2) & sin(\theta+\pi/2) \\ sin(\theta+\pi/2) & -cos(\theta+\pi/2) \end{bmatrix} \begin{bmatrix} a_{x,IMU} \\ a_{y,IMU} \end{bmatrix} . \tag{E.2}
$$

F. Motion Planning Scenarios

The motion planning algorithm discussed in section 5.2.2 is based on the fact that in order for the robot to reach its destination in the minimum amount of time it will be moving at either its maximum acceleration or maximum velocity. All the possible optimal control scenarios will be illustrated in this appendix.

The first scenario is when the robot is moving in the opposite direction of the desired destination. This is called Case 1 and is shown in figure F.1. The robot must accelerate in the opposite direction at its maximum acceleration till it reaches a standstill. From a standstill it will either stay stationary if it has reached its final destination or it will go to Case 2 or Case 4.

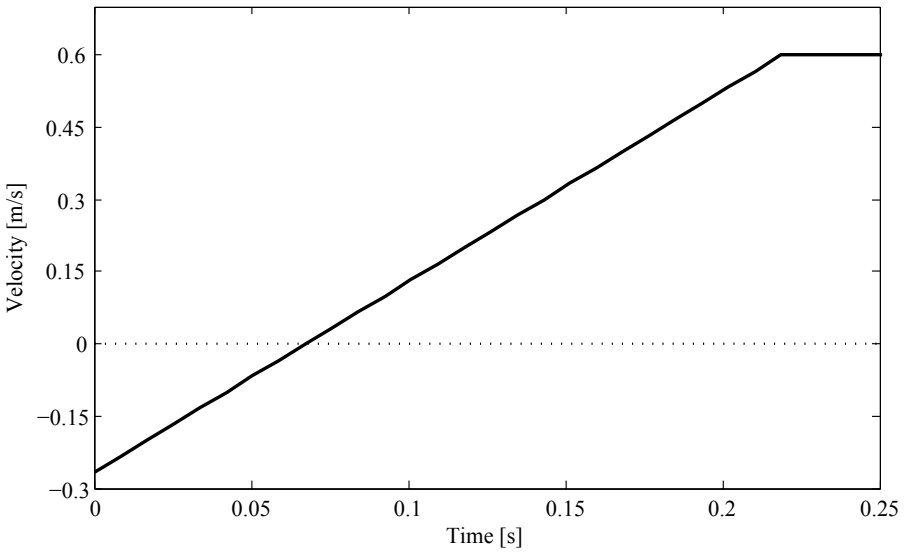


Figure F.1: Velocity profile Case 1

The Case 2 is the second scenario and is illustrated in figure F.2. This is when the robot needs keep on moving in the same direction, but it has not yet reached its maximum velocity. The robot must move at its maximum acceleration till it reaches its maximum velocity. Once it has reached its maximum velocity it moves to Case 3.

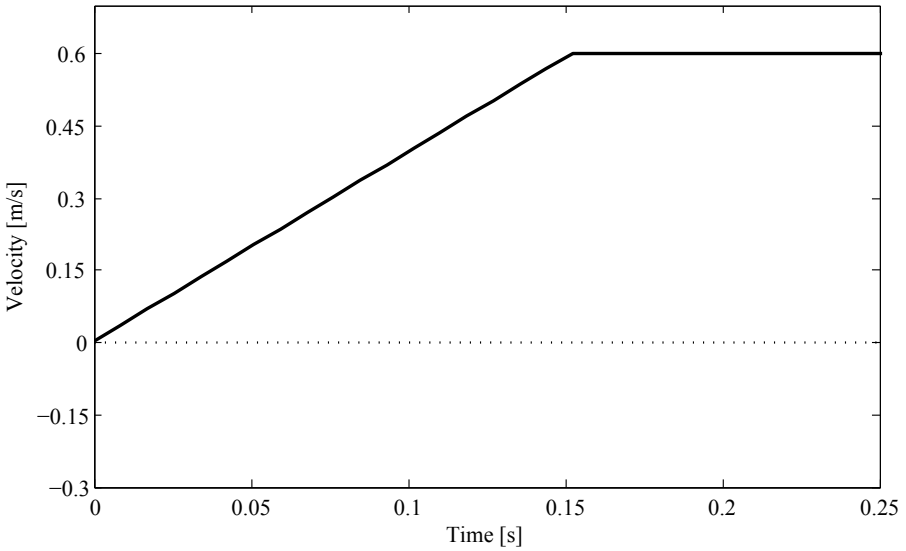


Figure F.2: Velocity profile Case 2

Case 3 is when the robot is travelling at its maximum velocity. The robot should maintain this velocity until it is required to decelerate because it needs to stop. Figure F.3 demonstrates the robot travelling at its maximum velocity.

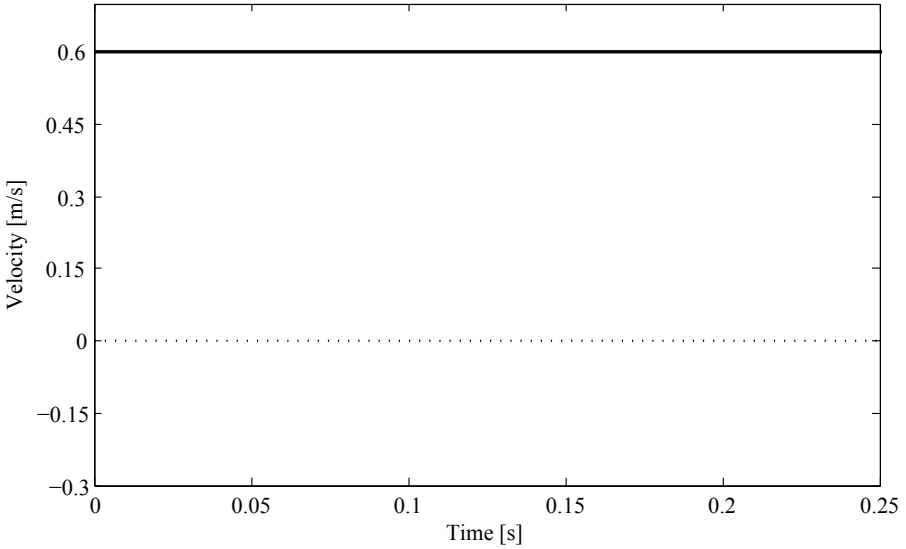


Figure F.3: Velocity profile Case 3

The fourth scenario occurs when the distance that the robot is required to travel is so short that it has to start decelerating before it has reached its velocity. Thus it will not reach its maximum velocity and only move at its maximum acceleration and deceleration. Case 4 is shown in figure F.4.

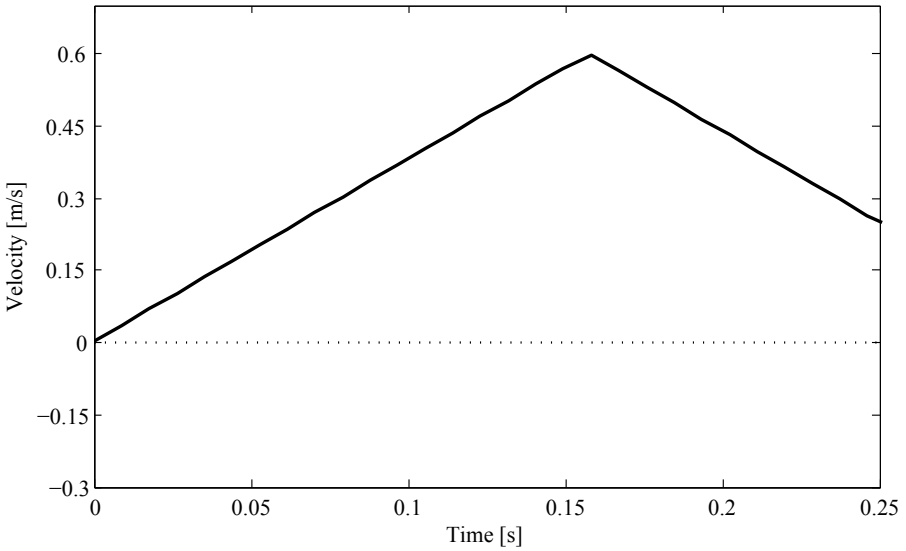


Figure F.4: Velocity profile Case 4

Case 5 is when the robot needs to stop. In order to stop in the minimum amount of time the robot needs to move at maximum deceleration till it reaches standstill.

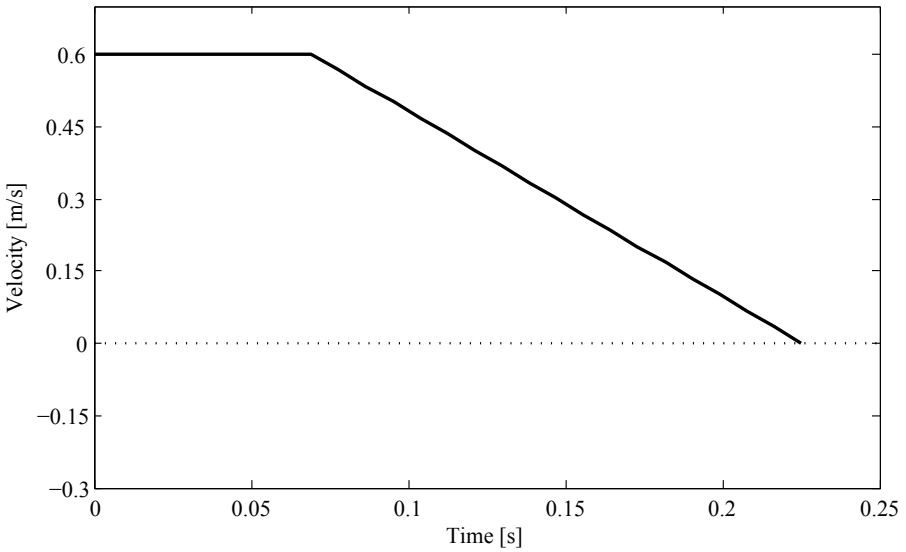


Figure F.5: Velocity profile Case 5

-

List of References