Development of a robot for RoboCup Small Size League, utilizing a distributed control architecture for a multi-robot system development platform

by

Albert Smit

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

Department of Mechanical and Mechatronic Engineering
Stellenbosch University
Private Bag X1,Matieland 7602, South Africa

Supervisors:

Prof. C. Scheffer
Dr. Y. Kim

December 2011

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: . . . .

Abstract

Development of a robot for RoboCup Small Size League, utilizing a distributed control architecture for a multi-robot system development platform

A Smit

Department of Mechanical and Mechatronic Engineering
Stellenbosch University
Private Bag X1,Matieland 7602, South Africa

Thesis: MScEng (Mechatronic)

December 2011

RoboCup promotes research in robotics and multi-robot systems (MRS). The RoboCup Small Size League (SSL), in particular, offers an entry level opportunity to take part in this field of study. This thesis presents a starting phase for research in robotics and MRS at Stellenbosch University. It includes the full documentation of the mechanical, electronic and software design of an omni-directional soccer robot for RoboCup SSL. The robot is also meant to operate as a hardware and software development platform for research in MRS. The platform was therefore designed with high-level programming language compatibility, a wide range of connectivity, and modularity in mind. The robot uses a single board computer (SBC) running a Linux operating system to accomplish these objectives. Moreover, a driver class library was written in C++ as a software application interface (API) for future development on the robot platform. The robot was also developed with a particular focus on a distributed control architecture. "Player" was implemented as the middleware, which can be used for communication between multiple robots in a distributed environment. Additionally, three tests were performed to demonstrate the functionality of the prototype: a PI speed control test, a direction accuracy test and a static communication test using the middleware. Recommendations for possible future work are also given.

Uittreksel

Ontwikkeling van ’n robot vir ’n RoboCup Klein Liga, deur gebruik te maak van ’n gedesentraliseerde beheerstelsel vir ’n *multi-robot-stelsel* ontwikkelingsplatform

A Smit

Departement Meganiese en Megatroniese Ingenieurswese
Universiteit van Stellenbosch
Privaatsak X1,Matieland 7602, Suid Afrika

Tesis: MScIng (Megatronies)

Desember 2011

RoboCup bevorder navorsing in robotika en *multi-robot-stelsels* (MRS). Die *RoboCup* Klein Liga (KL) bied in die besonder die geleentheid om op intreevlak navorsing te doen in hierdie veld. Hierdie tesis verteenwoordig die eerste fase van navorsing in robotika en MRS by Stellenbosch Universiteit. Dit sluit die volledige dokumentasie van die meganiese, elektroniese en sagteware-ontwerp van ’n omnidireksionele sokker-robot vir die KL in. Die robot is ook veronderstel om te dien as ’n hardeware- en sagteware-ontwikkelingsplatform vir navorsing in MRS. Die platform is dus ontwerp met ’n verskeidenheid van uitbreingsmoontlikhede en modulariteit in gedagte asook die moontlikheid om gebruik te maak van ’n hoë-vlak programmeertaal. Om hierdie doelwitte te bereik, maak die robot gebruik van ’n *enkel-bord-rekenaar* met ’n Linux bedryfstelsel. Verder was ’n sagteware drywer in C++ geskryf om te dien as ’n sagteware-koppelvlak vir toekomstige ontwikkeling op die robot platform. Die robot is ook ontwikkel met die besondere fokus op ’n gedesentraliseerde beheerstels. *Player* was geïmplementeer as die middelware, wat gebruik kan word vir kommunikasie tussen verskeie robotte in ’n gedesentralliseerde beheerstelsel. Daar is drie toetse uitgevoer om die funksionaliteit van die prototipe te demonstreer, ’n PI spoed beheer toets, ’n rigting akkuraatheidstoets en ’n statiese kommunikasie toets deur van die middelware gebruik te maak. Aanbevelings vir moontlike toekomstige werk word ook verskaf.

Acknowledgements

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

- The Department of Science and Technology (DST) and Dr. Chris R. Burger from the Meraka Institute at CSIR for funding the research and giving the opportunity.
- Mr. Ferdie Zietsman and the workshop personnel for their patience, endurance and advice in the manufacturing of the mechanical parts.
- Mr. Arno Barnard from the Electrical and Electronic department for helping to solve the problems encountered during the electronic design.
- Mrs. Susan Van der Spuy for her patience and help in making the endless amount of purchases.
- My supervisors, Dr. Yoonsoo Kim and Prof. Cornie Scheffer, for their continuous guidance and mentorship throughout the project, while also leaving space for self development.
- My friend and brother, John Cockcroft, for his continuous encouragement and support, reminding me to keep perspective and also for making himself available to bounce of ideas and finding solutions.

Dedications

To the Lord Jesus Christ
...we rejoice in our sufferings, knowing that suffering produces endurance, and endurance produces character, and character produces hope... and in all this, we glorify God.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

ADC	Analogue to Digital Converter
API	Application Interface
AI	Artificial Intelligence
CAD	Computer Aided Design
CAN	Controller Area Network
CCA	Central Control Architecture
CC	Central Controller
CPU	Central Processing Unit
CSIR	Council for Scientific and Industrial Research
DCA	Distributed Control Architecture
DC	Direct Current
DST	Department of Science and Technology
EFU	Electronic Functional Unit
ER	Electronic Requirements
FPGA	Feld Programmable Gate Arrays
GPIO	General Purpose Input Output
GPS	Global Positioning System
I^2C	Inter Integrated Circuit
IC	Integrated Circuit
IF	Interface
KZN	University of Kwazulu Natal
LiPo	Lithium Polymer
MARIE	Mobile and Autonomous Robotics Integration Environment
MC	Motor Controller
MIL	Mediator Interoperability Layer
MOSFET	Metal Oxide Field Effect Transistor
MR	Mechanical Requirements
MRS	Multi-robot Systems

MSL	Medium Size League
ODM	Omni-directional Drive Mechanism
ODW	Omni-directional Wheel
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PI	Proportional Integral
PID	Proportional Integral Derivative
PVC	Poly Vinyl Chloride
PWM	Pulse Width Modulation
RDE	Robot Development Environments
RF	Radio Frequency
ROS	Robot Operating System
RSDP	Robot Software Development Platform
RTAI	Real Time Application Interface
SA	South Africa
SBC	Single Board Computer
SPI	Serial Peripheral Interface
SR	Software Requirements
SSL	Small Size League
SSH	Secure Shell
SUN	University of Stellenbosch
TCP	Transition Control Protocol
UCT	University of Cape Town
UML	Unified Modelling Language
UP	University of Pretoria
USB	Universal Serial Bus
VGA	Video Graphics Array
WDM	Wheel Drive Mechanism
WLAN	Wireless Local Area Network

Chapter 1

Introduction

With the fast developing electronic age that we live in, the field of robotics is becoming more popular and accessible. At the same time, the application for robots is growing and stretches from toys to lawn mowers to rescue robots. The research presented in this thesis should be regarded as a starting phase for research in robotics and multi-robot systems (MRS) at the Department of Mechanical and Mechatronic Engineering at Stellenbosch University. This chapter will start with a background section and the motivation for the work. Thereafter, the objectives for this study are formulated. Finally, the scope of this research is presented which discusses the boundaries and the outline of this document.

1.1 Background on RoboCup

RoboCup, which is short for Robot Soccer World Cup, is a national and international research and education initiative that attempts to promote research in artificial intelligence (AI) and robotics. It provides a standard research platform, robot soccer, which allows for the development and integration of various technologies, while the competition element takes care of motivation for teams to compete as well as provide entertainment for the public. Robot soccer is a research field found within the context of MRS and robotics. In robot soccer, multiple, highly dynamic robots cooperate in a team in order to play a game of soccer. To accomplish this task, a number of research areas in AI and robotics enjoy attention, including autonomous agents, multi agent collaboration, behaviour learning, and sensor-fusion are typical research topics (Asada *et al.*, 1999).

RoboCup is considered as a benchmark for various research areas in robotics and provides a grand challenge for this domain which classifies it as a landmark project (a project that will have a socially significant impact). The RoboCup grand challenge is:

> ***"To develop a team of fully autonomous humanoid robots that can play and win against the human world champion soccer team, by the year 2050"*** (Burkhard *et al.*, 2002).

Building a robot that plays soccer might not seem significant, but the achievements made along the way to accomplish such a goal will definitely have a huge impact on technology. This is exactly the purpose of RoboCup's grand challenge, to apply the technological advances made in the pursuit of this challenge to other significant problems areas, for instance robot rescue missions (Tadokoro *et al.*, 2002; Kitano and Tadokoro, 2001).

RoboCup mainly consists of four events, namely RoboCupSoccer, RoboCupRescue, RoboCupHome and RoboCupJunior. RoboCupSoccer is the main event, initiating the landmark goal described above. RoboCupRescue aims to develop highly manoeuvrable robots to participate in search and rescue missions in large-scale disaster scenarios. Furthermore, RoboCupJunior is also a robot soccer event but gives the youngsters an opportunity by providing educational institutions with a much cheaper and simpler challenge. The last event is RoboCupHome which aims at developing service and assistive robots for the domestic area.

More attention will be given to RoboCupSoccer since it is the focus of this research. The RoboCupSoccer event has a number of soccer leagues of which the Soccer Small Size League (SSL) is of main interest. RoboCup SSL is named this way because of the size of the robots which is smaller relative to the other leagues. SSL robots consist of three or four wheels and a kicker and dribbler device. The robots are capable of holonomic movement. This is the ability to move in any direction from the starting point without the need to turn first, as in the case of a normal differential drive. This is also called an omni-directional drive mechanism (ODM). The robots have onboard control electronics, typically a microcontroller, to control the motors and onboard sensors. In a typical SSL soccer mach, the higher level control, for instance path planning, takes place on a central off-field computer. This computer monitors the game through an overhead camera and communicates to the robots over a wireless link. A central control architecture (CCA) like this is characteristic of the SSL. See Figure 1.1 for an illustration of the centralised architecture of the RoboCup SSL. Only a brief discussion on RobotCup is given here as an introduction to the motivation and objectives of this research. A more thorough description will be given in the next chapter.

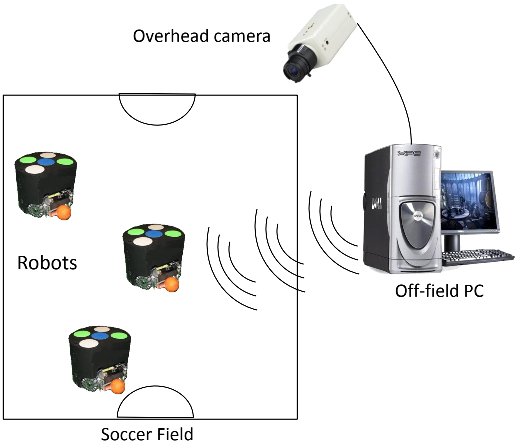


Figure 1.1: Central control architecture (CCA) of the RoboCup SSL (Shopland, 2010; Cincinnati, 2010; Zickler *et al.*, 2010)

1.2 Research Motivation

Universities all over the world participated in the RoboCup challenge since 1997 (Asada *et al.*, 1999). South Africa (SA) has only recently joined the race to reach the RoboCup goal. Some attempts to build a team of SSL robots have been made by various universities, but there was no participation on an international level until 2009. The University of Cape Town (UCT) participated in the world championships in 2009 and 2010. Other local institutes such as the Robotics Association of South Africa, the Mechatronics and Robotics Research Group at the University of Kwazulu-Natal (KZN), and some departments within the Council for Scientific and Industrial Research (CSIR) also fosters research in robotics, but not necessarily related to RoboCup. However, the Department of Science and Technology (DST) of SA has made funds available for research in robotics in 2008. This gave the opportunity to four universities in SA including the University of Cape Town, University of Kwazulu-Natal (KZN), University of Pretoria (UP) and Stellenbosch University (SUN) to take part in the RoboCup challenge. Hence, the first motivation for the research that started in 2009 at the SUN's Department of Mechanical and Mechatronic Engineering is due to the funding received from the DST.

Furthermore, there has not been many other research activities at SUN in the field of robotics before 2009. Consequently, everything necessary for robotic research needed to be built from scratch. Evidently, a set of robots or some form of development platform was first on the agenda. However, to ensure the continuation of the research and a broader application field, the development platform should not be limited to the RoboCup SSL domain, but rather to also open the door to development in the wider field of MRS. For instance, the centralised architecture of the SSL does give a lot of space for research in multi-robot cooperation and coordination, but is not appropriate to investigate fully autonomous and distributed robotic behaviour.

Therefore, it is of particular interest to go beyond the conventional centralised control system, towards a hybrid approach, keeping the functionality of the centralised system but adding the advantages of a distributed architecture. In this architecture, all robots are equipped with processors capable of high level autonomous control. The robots are set up to communicate with each other via a local network instead of receiving commands from the off-field computer. The distributed control architecture (DCA) is displayed in Figure 1.2. Although the other RoboCup leagues, such as the Medium Size League (MSL), are more suited for research in distributed multi-robot systems due to their already distributed nature, the SSL robots are much quicker and less expensive to develop for the initial phase of research in soccer robots. Also, by using this hybrid approach, it is possible to conduct research for the RoboCup SSL, and then at a later stage it will be very easy to shift to the higher leagues or any other multi-robot research that goes beyond RoboCup.

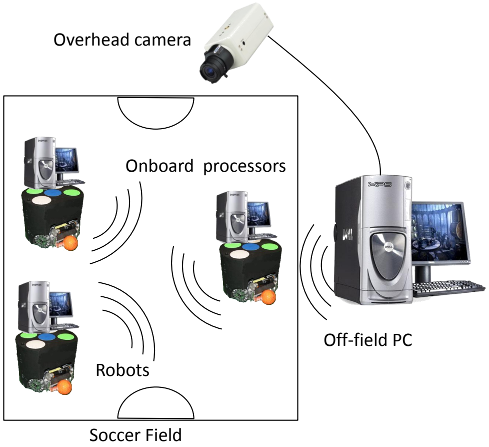


Figure 1.2: Distributed control architecture (DCA) for RoboCup SSL protect(Shopland, 2010; Cincinnati, 2010; Zickler *et al.*, 2010)

1.3 Research Objectives

Following from the motivation discussed above, the general aim of this research is to develop a robotic hardware and software development platform at the SU. The platform is supposed to enable current research in robot soccer as well as future research in MRS. The primary objective is stated below which is sub-divided into four secondary objectives.

Primary Objective

Design and manufacture the first prototype of an omni-directional soccer robot to be used in the RoboCup SSL competition, specifically making provision for a distributed control architecture in order for the robot to also serve as a hardware and software development platform for research in robotics and MRS.

Secondary Objectives

- Design the robot chassis to operate as an ODM that will allow the robot to be capable of holonomic movement. The chassis must ensure enough space for all the motors, a kicker and dribbler device and all electronic boards, while still remaining within the specification set out by the RoboCup rules.
- Design the electronic system that will control the ODM. The system should enable the robot to function in a CCA as well as a DCA. Evidently, the robot must also be able to communicate directly with an off-field computer as well as team robots. Furthermore, the electronic system must be expandable to make provision for future development and research.
- Develop the firmware and software that will implement the control algorithm for speed control on the motors and also make provision for position control. The software drivers should give access to all the hardware components on the robot and provide an application interface for further software development. Furthermore, the software should provide a communications platform for multiple robots in a distributed environment.
- As an extra objective, if time permits, develop a second and third prototype and test the functionality of the robots in a distributed environment by performing a certain cooperative task.

1.4 Research Boundaries

The broad scope of this research has been presented through the discussion on the motivation and objectives in the previous sections. However, the objectives are subject to certain boundaries. Firstly, the hardware development of the robot does not include the development of the kicker and dribbler device. This task was appointed to a third party. Secondly, the software development does not go beyond the level of an application interface. Any further development regarding artificial intelligence and high-level behaviour control was left as a future research topic. Thirdly, the development of a vision system for the SSL robots is a complete research field on its own and was also appointed to a third party.

1.5 Document Outline

The purpose of this document is twofold. Firstly it aims to give a report of the work that has been done to achieve the above-mentioned objectives. Secondly, it is meant to serve as a user manual to the robot platform that was developed. The document should provide enough information to a future researcher to be able to reproduce the design as well as to built on top of the current platform and continue the work. The report was written with these goals in mind.

The document follows the following structure. Chapter 2 gives an introduction to RoboCup and a literature review of the work that was relevant to the field of interest. The next three chapters, Chapter 3 to 5 consists of a detailed documentation on the design of the platform. These chapters also correspond to the first three secondary objectives. A systems engineering approach was followed throughout the design chapters. Chapter 3 outlines the mechanical design of the robot platform. Chapter 4 gives a full report of the electronic system design and development, aiming to give a future developer a good understanding of the operation of the electronic system. Chapter 5 discusses the software and firmware that has been developed for the robot platform. Thereafter, Chapter 6 describes the procedure that was followed in the performed tests and also gives the results thereof. Finally the research is summarised and concluded in Chapter 7. The significance of this work will be enlightened in Chapter 7 which also discusses possible future work. Additional technical information can be found in the appendices. Appendix A is a compilation of the mechanical manufacturing drawings of the robot, whereas Appendix B gives the electronic circuit diagrams. Thereafter, Appendix C is a summary diagram of the software system. Finally, Appendix D lists the contents of the CD included with this document.

Chapter 2

Robot Soccer and Literature Review

In the previous chapter a brief overview of RoboCup was given as an introduction to the research field of this thesis. In this chapter, the RoboCupSoccer event is discussed in more detail. Thereafter, an analytical derivation of the inverse kinematic equations for the omni-directional drive mechanism (ODM) is presented. Thereafter, a comprehensive survey on middleware is given. Finally, the last section discusses related research found in the literature.

2.1 RoboCupSoccer Leagues

Robot soccer is considered as an excellent test bed for MRS. It sets the field for development in autonomous multi-robot cooperation and collaboration in a highly dynamic environment. The RoboCupSoccer event is divided into five competition leagues, each allowing for different aspects of robotics research. These leagues are formally known as the SSL, which was described in Chapter 1, Middle Size League (MSL), Simulation League, Humanoid League and Standard Platform League. A closer look at each of these leagues follows in this section.

2.1.1 RoboCup Small Size League

SSL robots are quite small, 180 mm in radius and 150 mm in height. In this league, a game consists of two teams of five robots per team, playing 10 minutes on a side. The game is played on a carpeted field that is 6.05 m long and 4.05 m wide. The game is played with an orange golf ball. A typical robot consists of a chassis, driving system, kicker and dribbling mechanism and control electronics.

The chassis is simply a base plate on which the motors are mounted and a top plate to fit the electronics. A solenoid is fitted inside the chassis that can extend and retract to create the kicking action. This enables the robot to pass or shoot the ball to a team mate or into the goal net. The ball speed is also controllable by adjusting the power on the solenoid and can reach up to 15 m/s (Zickler *et al.*, 2008). Most robots also have a chip kicking mechanism to chip the ball over the opponent. Moreover, each robot is fitted with a dribbler, which is a horizontal roller in front of the robot connected to a small motor. This roller is then used to give the ball a back spin and thereby enabling the robot to control the ball.

What is significant of these robots is the driving system by which they are moving around the field, the ODM. This drive mechanism consists of small custom designed omni-directional wheels fitted to the base plate. These wheels have little rollers on the circumference of the wheel, enabling the wheel to move forward as well as sideways simultaneously. By spacing four of these wheels around the circular base plate, more or less perpendicular to each other, the ODM is created. This gives the robot the holonomic movement feature described earlier. The holonomic movement allows the robots to be very fast and manoeuvrable, making SSL robot soccer very competitive.

To control the motors, the robots also have onboard electronics and a high speed microcontroller. The ARM 7 core, running at 58 MHz, or field programmable gate arrays (FPGA's) are often used (Kriengwattanakul *et al.*, 2008; Wu *et al.*, 2009). These processors have low power consumption, high performance and integrated interfacing circuitry. Figure 2.1 depicts a typical robot used in the RoboCup SSL.

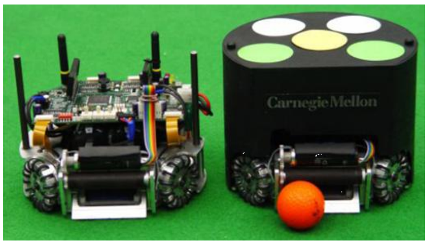


Figure 2.1: Typical Small Size League robot (Zickler *et al.*, 2008)

Apart from the robots, there are two other important components in a SSL team, namely the vision system and off-field computer. Although the robots have onboard processors, the processors are only used to perform local calculations such as motor and dribbler control, sensor data processing and communication from the off-field computer. The actual game control and AI is programmed and executed on a separate computer situated next to the field. The field is also equipped with two high frame rate cameras, 4 meters above the playing field. Each camera observes one half of the field. This vision system provides the global view and determines the position of each robot as well as the ball. The position is calculated on every image frame at a rate of 60 times per second. Each robot is colour coded with stickers to destinguish them from each other.

The position information is fed to the AI system which then determines the next action to take depending on the team strategy. The motion vectors are then sent to each robot via a wireless link. This type of architecture is known as central control architecture (CCA), due to the central off-field computer where the actual control takes place. Figure 2.2 is another illustration of the the CCA, also showing the two overhead cameras and the referee box. The referee computer is used to start and stop as well as monitor the game play. The SSL is designed to allow for hardware development in terms of highly dynamic and fast robots. It also provides an opportunity for software development in designing complex control and strategy algorithms for multi-robot cooperation.

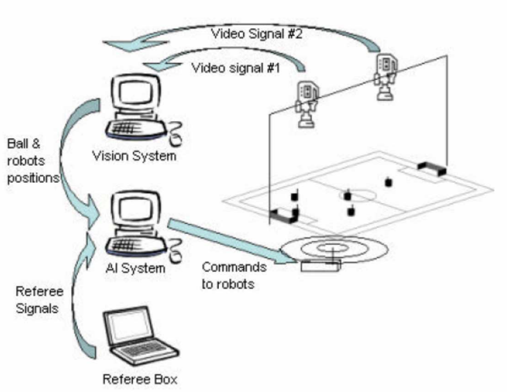


Figure 2.2: Central control architecture (CCA) for Small Size League (SSL) (Martinez-Gomez *et al.*, 2005)

2.1.2 RoboCup Middle Size League

Although the MSL is not directly related to the scope of this project, it is still discussed in this section to give a broader perspective of RoboCup and to point out the differences to the SSL and some similarities to this research. The most obvious difference between the SSL and the MSL is the physical size of the robots, 50 cm × 50 cm × 80 cm. The dimensions of the field is larger and the game is played with a standard soccer ball. The robots make use of an ODM with three wheels instead of four. One similarity is the dribbler and a kicker mechanism which is also apparent on a MSL robot.

The MSL also differs from the SSL in the sense that there is no global vision or off-field computer. It is a requirement in this league that all sensors should be onboard. Therefore, these robots are equipped with a specially designed omni-directional vision systems (Azevedo *et al.*, 2009). The vision system incorporates a hyperboloidal mirror and a vertical camera looking at the lens. Light from a 360 ° radius is then reflected off the mirror into the camera and in this manner, the robots can "see" in all directions at once. Figure 2.3 portrays how the camera can see the whole soccer field.

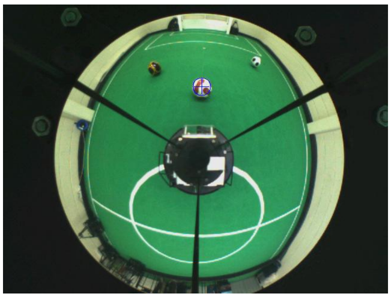
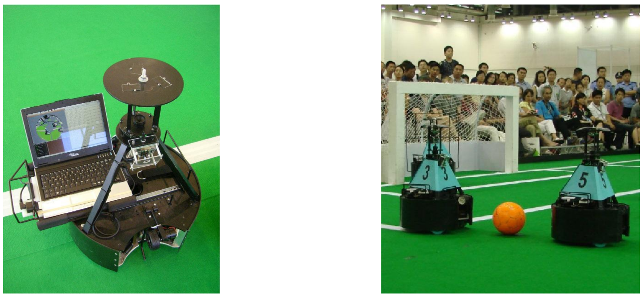


Figure 2.3: Omni vision system on a Medium Size League (MSL) robot (Azevedo *et al.*, 2009)

The MSL also requires that all processing takes place onboard (no off-field PC is allowed). Fortunately there is enough space on the robot to fit a laptop (see Figure 2.4). The laptop is responsible for all high-level control on the robot and image processing, as well as providing communication between the robots through a wireless IEEE 802.11 link. Lower level control, for instance motors, can then be delegated to dedicated microcontrollers which are interfaced to the laptop through a controller area network (CAN). The CAN bus is a real-time communications bus for embedded systems.



(a) Medium Size League (MSL) robot with onboard laptop

(b) Medium Size League (MSL) robots and soccer ball

Figure 2.4: Typical robots for RoboCup Medium Size League (MSL) (Azevedo *et al.*, 2009)

The distributed nature of the MSL and the restrictions that all sensors must be onboard, allows for research in developing fully real-time autonomous systems. Therefore, in the light of the objective to develop a distributed system, the principles used in the MSL is also of interest for this research.

2.1.3 Other Leagues in RoboCupSoccer

RoboCup consists of other leagues that are less relevant to this research, but will be mentioned briefly. The Humanoid League is the most challenging and closest to the actual goal of RoboCup. In this league the robots are fully autonomous and human-like in their physical appearance and operation. A humanoid robot is shown in Figure 2.5. The research issues that are involved in this league, includes dynamic walking, running, and kicking of the ball. All this has to be done while maintaining balance and visual perception of the ball, other players, and the field.

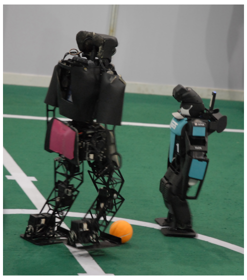


Figure 2.5: Typical robots for RoboCup Humanoid League (Nimbro, 2010)

The Standard Platform League uses a humanoid robot as well, but all the teams are provided with the hardware. The hardware is the same for all participants, hence the name Standard Platform League. The teams therefore only need to focus on the software development. The humanoid robot used for this league is the Nao robot developed by Aldeberan Robotics. The Nao robot, as shown in Figure 2.6, has 25 degrees of freedom and is equipped with a number of different sensors including inertial sensor, force sensitive resistors, Hall effect sensors, and sonar sensors.

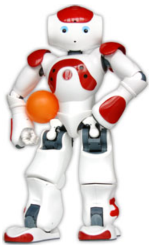


Figure 2.6: The Nao robot for RoboCup Standard Platform League (RoboCup, 2009)

Finally, the Simulation League is a software approach to robot soccer. The game is played using the RoboCup Soccer Simulater software. The server hosts the match and the two teams can join their players as clients. The simulator provides a physical 2-dimensional or 3-dimensional simulation framework in which the virtual soccer players can compete in a dynamic multi-agent environment. An example of a typical game is seen in Figure 2.6. The Simulation League provides a challenging environment for research in AI and multi-agent software systems.

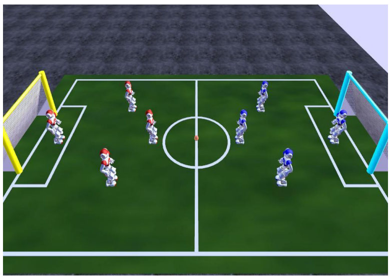
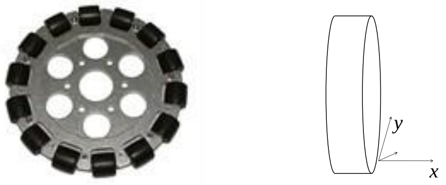


Figure 2.7: Example of a 3-dimensional soccer simulation (Wikipedia, 2011)

2.2 Holonomic Movement of an Omni-directional Robot

Manoeuvrability and speed are two essential requirements in robot soccer and even more so in the SSL. To accomplish the rapid movement and change of direction, the conventional differential drive is not sufficient anymore and omni-directional movement has become an absolute necessity in robot soccer. The ODM gives the robots the significant ability to move very quickly and immediately in any direction. This section will discuss the principle behind the ODM used in soccer robots as well as a simple derivation of a kinematic model for the ODM.

To understand the workings of the ODM, it is necessary to understand the key element in the system, which is the omni-directional wheel (ODW). Independent rollers are fitted around the circumference of the wheel with their degree of freedom perpendicular to that of the wheel itself (see Figure 2.8a). In other words, while the wheel is driven forwards or backwards on the y-axis, seen in Figure 2.8b, the rollers provides an extra degree of freedom in the direction of the x-axis, causing the wheel to drift in a diagonal direction.



(a) Omni-directional wheel (ODW)

(b) ODW that moves diagonal

Figure 2.8: Omni-directional wheel, (a): (RobotShop, 2011a)

Consider a circular base plate with four ODWs mounted on the edge. This is seen in Figure 2.9 where the wheels are numbered one to four from the positive x axis. The wheels are mounted with angles θ_n and, for the sake of simplification, the angles are such that the wheels are perpendicular to each other. Each wheel also has a motor connected to it and can therefore exert a translational force in the positive or negative direction, resulting in a translational velocity component of each wheel denoted by v_n. One can now imagine that if all the wheels are turning in the same direction with velocity v_n, and the positive direction is the direction of the arrow, the platform will just rotate on the spot.

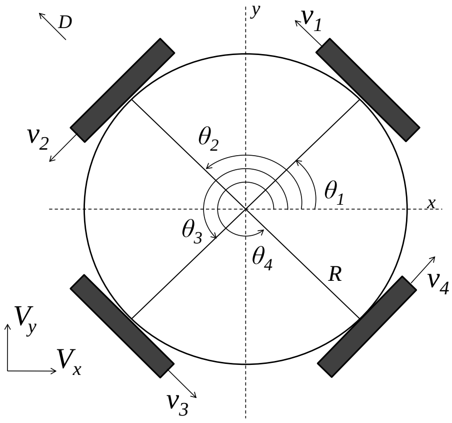


Figure 2.9: Omnidirectional platform capable of holonomic movement

Equally evident, if two opposite wheels, for instance 1 and 3, are turning in opposite directions with both exerting a force in the direction of D, the platform will move in that direction. In this case the two opposite wheels are providing the drive in the D direction, while the other two wheels are stationary and the rollers provide the movement in the D direction. Similarly, if the platform is moving in the direction of the x-axis, all the wheels are drifting diagonally. The wheels provide half of the movement and the rollers the other half. The same can be said for any direction in the $x - y$ plane with the movement supplied by the rollers and wheels changing, depending on the direction.

It is now clear that by driving each wheel individually and combining the forces, the resultant force will drive the platform in any desired direction. It is noteworthy that this 360° movement has been accomplished without any change in orientation of the platform, which is the great advantage of an omni-directional mobile platform. If rotational movement is added to the translational movement of the platform, even more interesting manoeuvres can be achieved, for instance circular and arc formations.

The mathematical problem that now exits is how to make the platform move along a certain path or simply move in a certain direction. Consider Figure 2.9 again, where the platform's global velocity is denoted by capital V_x, V_y and rotational or angular velocity by ω, while small v_n denotes the individual wheel velocities. The question that arises is what should the wheel velocities v_n be in order to obtain a certain global velocity V_x and V_y?

This is not a new problem and many solutions have been proposed in the literature, with Watanabe (1998) being one of the earlier publications on this topic. Williams *et al.* (2002) derived a dynamic model that also includes the effect of slippage on the wheels. Wasuntapichaikul *et al.* (2010) is yet another example which also included slippage on the wheels by modifying the kinematic model with some disturbance parameters. However, if the sensors on the robot are highly accurate and the ground is planar enough, it is accepted to make some assumptions and to discard the effect of slippage and determine only a simple kinematic model (Li and Zell, 2009).

For this study the solution of Rojas and Förster (2006) has been chosen for its simple derivation method. As mentioned above, slippage will be disregarded in the derivation for simplicity reasons. If the platform is now considered to move in the positive x direction with speed V_x, then it is seen in Figure 2.10 that wheel 1 has a translational velocity component $-V_x \sin\theta$, while the rollers have a translational velocity of $V_x \cos\theta$. In the same way, if the platform is moving in the positive y direction, then the wheel translational velocity will be given by $V_x \cos\theta$.

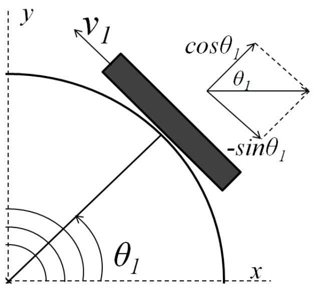


Figure 2.10: Translational velocity components of the wheels

The rollers are not actuated and therefore can be discarded for the kinematic model. Lastly, the rotation of the platform also causes a change in the velocity of the wheel and needs to be added to the equation. The translational velocity caused by the rotation of the platform can easily be calculated as $R\omega$, where R is the radius of the platform and ω the platform's rotational velocity. The following equation then holds for wheel n:

$$v_n = -V_x \sin\theta_n + V_y \cos\theta_n + R\omega \tag{2.2.1}$$

Putting equation 2.2.1 in matrix formation for the four wheels, the inverse kinematic equation to calculate the wheel velocities are given by the matrix equation 2.2.2.

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} -\sin\theta_1 & \cos\theta_1 & R \\ -\sin\theta_2 & \cos\theta_2 & R \\ -\sin\theta_3 & \cos\theta_3 & R \\ -\sin\theta_4 & \cos\theta_4 & R \end{pmatrix} \begin{pmatrix} V_x \\ V_y \\ \omega \end{pmatrix} \tag{2.2.2}$$

or

$$\overline{v} = D\overline{V} \tag{2.2.3}$$

where $\overline{v}$ is the wheel velocity vector, D is the inverse kinematic matrix and $\overline{V}$ is the global velocity vector. The inverse of this equation can also be obtained to calculate the global velocity from the locally measured values. This result is shown in equation 2.2.4:

$$\overline{V} = D^{+}\overline{v} \tag{2.2.4}$$

where D^{+} is the pseudo-inverse of the D matrix.

2.3 Middleware in the Robotics Domain

A distributed robotic system consists of a number of modules, each equipped with certain sensors and actuators. These modules then communicate with each other to make use of their resources in order to accomplish a challenging task. This cooperation between the modules becomes a fairly complex scenario to manage and calls for some sort of integration software to handle the communication and sharing of resources.

Such a software system should provide a framework in which these robot modules are seamlessly integrated, although distributed. It should operate as one single module with many functions. Only then will the advantages of a distributed system be utilised to the fullest. Moreover, the software framework must provide an interface to the available hardware on the robot module and also offer services that will ease the functionality and development of the robot system.

A robot software development framework like this can easily be thought of as analogous to an operating system for a desktop computer, providing an interface to the user through the mouse and keyboard and also offers software packages available to the user. Additionally, such a system has a network connection to communicate to other computers and share their resources. The only difference is that such an operating system for robots is much more difficult to develop than its desktop computer counterpart. Hence, robot operating systems are still in its early stages of development compared to operating systems such as Microsoft Windows, Linux Ubuntu or Apple Mac OS.

2.3.1 What is Middleware?

The type of robotic software framework discussed above is more formally known as *middleware*. However, the term middleware is not restricted to the field of robotics, but actually refers to any software system that provides communication between distributed applications. Like the term suggests, middleware positions itself in the middle between the low-level hardware abstraction layer and the high-level application software. Middleware systems can adopt different architectures and styles in which it realises the communication. These different architectures becomes another field of research. However, for the purpose of this study it is only necessary to know that middleware serves as a communication framework in distributed systems.

A formal definition, according to Goedicke and Zdun (2001), states that *"Middleware extends the platform with a framework comprising components, services, and tools for the development of distributed applications. It aims at the integration, effective development, and flexible extensibility of the application."* From the definition it is clear that middleware not only provides communication in a distributed environment, but also extends the platform with functional tools and services for the development of a distributed system. It is because of these characteristics, that middleware forms the basis of any software development framework in distributed or multi-robot systems as well.

2.3.2 Requirements of a Middleware

While the distributed robotic systems has many advantages, it also generates some challenges like communication and coordination between the different modules. Middleware is the solution to these challenges. The middleware can solve these issues by providing hardware abstractions and add functionality to the system. Other advantages of middleware is the extensibility and reliability of such a system due to its modular architecture (Namoshe *et al.*, 2008). Mohamed *et al.* (2008) also states that data and sensor fusion is another important aspect that middleware offers. Kramer and Scheutz (2007) lists a number of aspects that middleware have in order to solve the challenges at hand. The aspects are simplification, interoperability, heterogeneity, modularity, resource discovery and configuration.

The first and most important task is merely to simplify the development process. Simplification is usually done by providing an easy to use higher-level abstraction to the developer, and by supporting software re-usability. In this way, developers can share their code instead of reinventing the wheel.

A further requirement of middleware is communication and interoperability. Robots are manufactured by numerous different companies with each having a different design. These different devices need to communicate with each other. Therefore, the middleware must provide this interoperability mechanism. Moreover, the robots not only need to communicate, but cooperate with other devices as well. Therefore, the middleware must also offer a high-level application interface to facilitate the collaboration functionality.

Another common issue is the heterogeneity of the robotic devices. If all robots were the same, writing software for them would be as easy as copying and pasting software code. However, this is not the case, and communication between devices with different architectures becomes a non-trivial task. The role of the middleware is to provide an abstraction layer to hide the heterogeneity of the low-level components and create a uniform interface.

A developer often spends a great deal of time writing his own implementation of an existing algorithm, instead of reusing existing code, because it simply does not fit his custom hardware components. If the middleware can provide these functionalities as extra modules to the framework, then the developer can easily choose and add the module he needs, instead of spending his time on rewriting the code.

It has already been said that robots often want to communicate with each other. Additionally, robots want to communicate with external devices, to make use of the extra resources and enhance their functionality. Due to the nature of the situation, the availability of these resources is dynamically changing. This means that the robot must have the ability to automatically discover this newly available resource and also configure itself to make use of it. Thus, middleware also needs to support automatic resource discovery and configuration.

2.3.3 Review of Middleware Solutions

There are several survey papers in the literature that provides an overview of the state-of-the-art robotic middleware solutions. A number of survey publications were reviewed to determine what middleware solution will be appropriate for this research.

The most recent publication known to the author is that of Mohamed *et al.* (2009). It presents an overview of the current solutions and explores the characteristics and roles of a middleware platform for multi-robots systems and provides a criteria to evaluate the solutions. Additionally, Mohamed *et al.*

(2008) gives a short description of several middleware platforms and states the main objective of each with no further discussion.

Furthermore, freely available middleware solutions are discussed in Namoshe *et al.* (2008) together with an example implementation of a multi-robot cooperation algorithm using "Player" as the middleware platform. Another interresting survey was done by Shakhimardanov and Prassler (2007), which provides an overview discussion on three solutions, but moreover, it suggests a comparison and evaluation methodology for middleware solutions.

The most comprehensive survey is done by Kramer and Scheutz (2007). It provides a set of features that sets the basis for evaluation criteria, and then applies these criteria to several middleware platforms or robot development environments (RDE). Note that a RDE is also called robot software development platform (RSDP). Moreover, Kramer and Scheutz (2007) gives a practical usability evaluation on the architecture design, implementation, and execution of the RSDPs on a robot. It also includes the impact of the RSDPs on the multi-robotics field by using the published work as an indicator. These three evaluation sets (conceptual features, practical implementation and the impact on the field) are combined to give an overall score to the RSDPs. For this reason, Mohamed *et al.* (2009) and Kramer and Scheutz (2007) will be considered as the benchmark surveys for selecting a middleware solution.

From the review of the survey papers, nine middleware solutions were selected. These solutions were evaluated against the predefined criteria that was determined by the requirements of this research. A full discussion of the evaluation will be given in Chapter 5. Only three of the alternatives, Robot Operating System (ROS), Mobile and Autonomous Robotics Integration Environment (MARIE) and Player, were studied in more detail and are described below.

Robot Operating System (ROS)

ROS, developed at the Willow Garage Robotics Research Institute (Quigley *et al.*, 2009), provides the services that are typically found in a normal operating system for a computer. However, ROS was designed for robots. Typical services are, for instance, hardware abstraction, low-level hardware control and communication between processes. It also gives higher-level functionality by means of stacks that can be added onto the operating system, much like installing a new program on a normal operating system. Although ROS was originally designed for large-scale service robots, it is now widely used in many robotic applications.

A ROS system is typically a number of nodes that performs the computation, connected in a peer-to-peer network. Each node serves a different function. One node will, for instance, do the vision processing and the other will control the wheels, while a third one does the navigation. This arrangement is called a computational graph. ROS is designed to be multi-lingual by either implementing it in the target language or wrapping it in a library. It currently supports main languages such as C++, Python and Octave. Also, all the algorithms and drivers are implemented in standalone libraries that have no dependencies on ROS. This allows for easier reuse of the code beyond its original application.

Mobile Autonomous Robotics Integration Environment (MARIE)

Various robotic software solutions are being developed and they all implement different communication approaches. This means that most of the programming environments are not compatible with each other. This makes it difficult to integrate the systems to benefit from each of their unique functionalities, or to reuse their components. MARIE is a middleware solution with a different goal in mind. MARIE does not implement yet another way of doing the same thing, but rather, it aims to integrate multiple applications. It is designed with three software requirements in mind: Reusing of available solutions, supporting multiple sets of concepts and abstractions, and supporting a wide range of communication protocols, mechanisms and robotics standards (Côté *et al.*, 2006).

MARIE achieves these goals by creating a mediator interoperability layer (MIL). The MIL is simply a centralised control unit that interacts with each application. The central control unit, called the mediator, contains two different software components, the application adapter and communication adapter. These adapters serve as the bridge between the external applications and the mediator. The mediator in turn serves as the bridge between the different applications. In this way, the applications are able to communicate with each other without having to support other communication protocols other than its own, because the mediator does the translation work. MARIE is, therefore, a programming environment that allows multiple applications, programs and tools, to operate on one or multiple machines and work together to implement a versatile robot software framework (Côté *et al.*, 2004).

The Player Project

The Player Project was initially developed for interfacing and simulation of multi-robot systems (Gerkey *et al.*, 2003). Player has since been greatly extended by various researchers over the world and has become the most widely used multi-robot software development framework. Player implements a client/server architecture and is, in essence, a distributed device server. The server provides clients with network-oriented programming interfaces to access the resources connected to the server (Collett *et al.*, 2005). The Player server runs on the robot and has access to the robots' hardware through a built-in driver.

Player supports a large amount of hardware devices such as cameras, lasers and range finders. The server also makes provision for custom-built hardware through a dynamically loadable plug-in driver. A hardware peripheral that is connected to the player server through a driver, is called a "device". Once the server is started, a client application can connect to the server and subscribe to any of the devices available on the robot. The client then has access to the device through a standard interface. The client application is where the high-level control algorithms are implemented, for instance obstacle avoidance and localization.

Such a client application can be executed on either the same robot that is hosting the server or on any other system connected to the server on a network. This allows a robot to be controlled remotely over a network but moreover, it allows multiple client connection to a server. In this way different robot functionalities can be decoupled and executed on separate systems. For instance, a low-resource robot can have an on-board client programme running the obstacle avoidance algorithm while the localisation is implemented on a remotely connected PC with higher processing power. Player also supports multiple servers to run simultaneously. That means a client can connect to different servers, in essence different robots, and make use of all the resources available. This is the main feature of Player and enables distributed control of multiple robots and sensor fusion.

Furthermore, the server is decoupled from the transport layer or communication layer, which gives Player the ability to use different network transport layers, for instance transition control protocol (TCP) sockets (Vaughan *et al.*, 2001). This feature results that the Player architecture is programming language independent. Thus, because the server's external interface is only a network interface, the client application can be written in any programming language that supports a network transport layer. All together, Player is an established middleware solution and provides good support through its large user community.

2.4 Related Work

The architecture of the RoboCup SSL does not particularly support research in distributed systems, and the MSL is rather more appropriate. Nevertheless, some attempts have been made to incorporate a distributed system in the SSL team. Since this approach is the focus of this research, publications on this topic was reviewed and will be discussed briefly in this section.

Weitzenfeld *et al.* (2007) extended their SSL architecture to be more functional in a real-world situation. More specifically, Weitzenfeld *et al.* (2007) wanted to extend the capabilities of the robots to operate in search and rescue missions. The robots were modified to include a local vision system and ad-hoc networking capabilities. This was achieved by replacing the microcontroller with a Crossbow Stargate single board computer (SBC) and connecting it to a local web camera and 802.11 communications device. The SBC runs an embedded Linux operating system and therefore, all AI software that normally runs on the off-field computer, could easily be transferred to the robot and implemented locally. Though their work went beyond the scope of SSL, it illustrates how a team of SSL robots can be used in a distributed scenario.

Another noteworthy study is that of Aduthaya *et al.* (2006). Their research showed a similar idea of a distributed approach in RoboCup SSL. However, their system is distributed in the sense that each robot has a number of dedicated microcontrollers instead of having one single main controller that takes care of a certain task on the robot. These dedicated microcontrollers or subsystems are responsible for features such as motor control and sensor data collection. The robots also incorporated some interesting on-board sensors to ease the task of the central off-field computer. One such sensor includes an electronic compass that can determine the orientation of the robot. Another innovative concept encountered on these robots is the optical mouse position sensor. Instead of reading the position feedback from the motor encoders, this sensor is mounted in the middle of the robot and detects x and y position movement in the same way as an optical computer mouse.

Aduthaya *et al.* (2006) also made use of a local camera mounted just above the dribbler. The camera is connected to its own controller, forming another subsystem. The camera subsystem is then utilised to detect and track the ball automatically, again sparing processing power on the off-field computer. However, the processing power of a normal computer is now developed to such an extent that the attempts to lighten the weight of the off-field PC have become almost trivial. Still, this research showed innovative approaches towards a distributed architecture.

Lastly, the work of Köker and German (2007) is probably the closest to the objectives of this research. Their research presented an evaluation of the performance of a distributed embedded system in a dynamic environment. Their goal was to shift the control from the centralised system to the robots for local execution of the game play algorithms. This means that while the global vision system is left as it is, the decision-making and strategy algorithms are implemented on the robots and the off-field computer only processes imagery commands.

This approach was achieved by building a small embedded system with a SBC using the PC/104 form factor. This particular SBC is similar to a very small, off-the-shelf desktop computer with a 300 MHz central processing unit (CPU). That means the SBC acts as the main controller and communicates with additional microcontrollers also situated on the robot. In addition to this, an embedded Linux operating system was built to run on the SBC with uClibc, which is a small C library for software development in C. The operating system was patched with a real-time application interface (RTAI) which guarantees a fast and predictable response from the operating system and thus minimises delay. No actual field performance tests were performed, but latency tests on the response times of the system showed that it is a viable solution.

2.5 Conclusion

This research has a particular interest in the RoboCup SSL, although there are aspects in the MSL that are also related, especially the distributed architecture. These features of the MSL can be used to develop the distributed system. Furthermore, the movement of a holonomic robot was investigated and the inverse kinematic equations was derived. These equations will be used later on in the control of the robot. The section on middleware clarified the purpose and the importance thereof in a distributed system. A full evaluation to determine the best alternative will be given in Chapter 5. Finally, the review of related studies showed valuable insights into different approaches to distributed systems in RoboCup SSL.

Chapter 3

Mechanical Design

This chapter gives an account of the mechanical design of the robot platform. A preliminary design was first performed. The preliminary design was a conceptual level design consisting of setting up the requirements and building a mechanical model with computer aided design (CAD) software. Thereafter, the detailed design was performed which included the manufacturing of the prototype and cycling through the design iterations. Finally a conclusion to the mechanical design is given at the end of this chapter.

3.1 Preliminary Mechanical Design

The preliminary design section starts with the mechanical design requirements. Thereafter, the CAD model of all the mechanical parts is described.

3.1.1 Mechanical System Requirements

The first secondary objective of this research is to design the chassis of the robot. In order to achieve this objective, the mechanical design requirements were derived from the objectives. The requirements are numbered and will be referred to as mechanical requirements (MR) and its corresponding number in the rest of the text, for instance MR 1. The mechanical system is required to:

MR 1. Consist of an omni-directional drive mechanism that provides holonomic motion to the robot.

MR 2. Minimize the weight of the robot platform.

MR 3. Ensure enough space to fit all motors, the kicker, the dribbler and the electronics.

MR 4. Be within the maximum dimensions of 180 mm in diameter and 150 mm in hight.

3.1.2 CAD Model of the Robot Platform

The mechanical robot platform was first modelled in the CAD software package, Inventer 2010. The mechanical design was based on a previous SSL robot design, made available by the University of Bremen in Germany. The basic concept of that design was kept, and a few changes were made to fit the requirements of this research.

The new design is seen in Figure 3.1. The design consists of a circular base plate with four indents on the sides where the wheels can be exposed to the surface. On each indent a mount plate is attached perpendicular to the base plate. A shaft is attached to each mount plate that holds the omni-directional wheels (ODW) and spur gear. Additionally, the motors are also attached to the mount plates on the opposite sides of the wheels. Note that the front wheels are mounted further apart than the back wheels to make space for the kicker and dribbler devices.

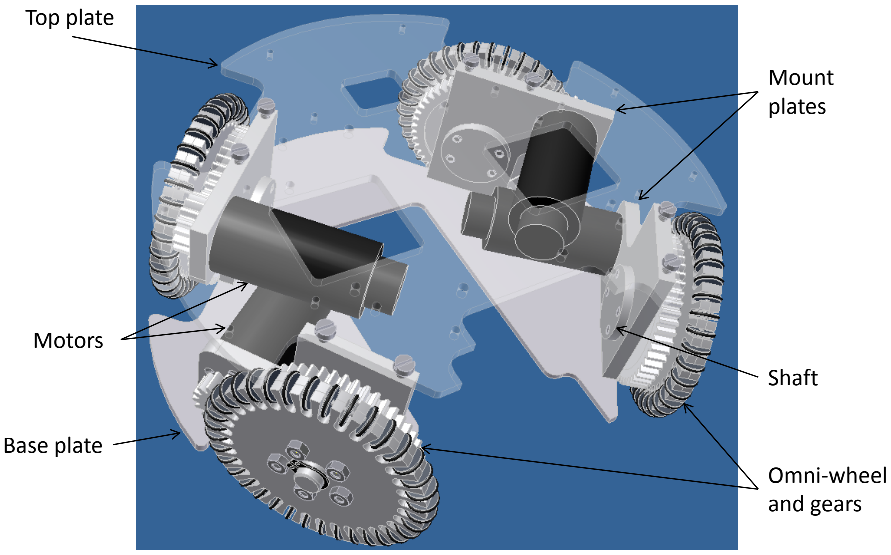
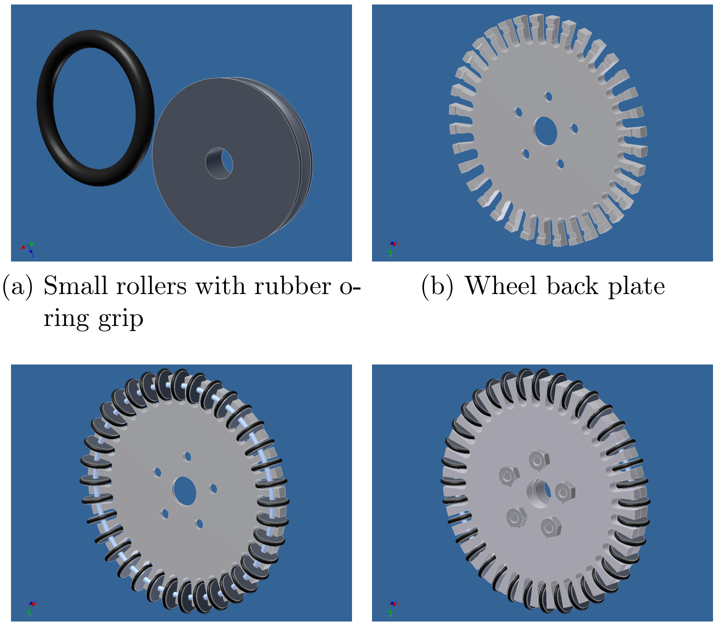


Figure 3.1: CAD model of the omni-directional mechanism (ODM)

Furthermore, the pinion gear attached to the motor goes through a hole in the mount plate to mesh with the spur gear. All these parts together form the ODM. Another plate is inserted on top in order to cover the drive mechanism and to separate it from the electronics that are fitted above it. The top plate, shown transparent, was designed with three holes that are cut away, through which the wires can connect the motors to the electronic system.

Some parts of the original design from the University of Bremen, for instance the gears, were not locally available. The gear system and its surrounding parts were, therefore, redesigned. Another adjustment that was made from the original design, was the layout of the top plate to fit the custom electronics. Furthermore, the original ODW were completely redesigned in order to fit the needs of this study and to simplify manufacturing. The new ODW design is seen in Figure 3.2a. First, two circular plates, the back and front wheel plates, are designed with teeth on the circumference. Figure 3.2b shows the back wheel plate. Then, small rollers were designed as a disc with a groove on the edge. The rubber o-ring fits into the groove. Each roller has a hole drilled through the middle and is strung on a wire ring. The ring fits into a small slot in the wheel back plate so that the rollers are spaced between the teeth (Figure 3.2c). Finally, the front wheel plate holds everything together. The complete CAD model of the ODW is seen in Figure 3.2d. This design aimed for simplified manufacturing and easy assembly.



(a) Small rollers with rubber o-ring grip

(b) Wheel back plate

(c) Back plate with wire ring and rollers

(d) Assembly of the ODW

Figure 3.2: CAD model of the omni-directional wheel (ODW)

Each ODW is driven by a motor and gear system, known as the wheel drive mechanism (WDM). A new WDM was also designed, using the locally available gears. The purpose of the gear system is to reduce the high speed of the motors to a reasonable speed for the wheels. To achieve this, two spur gears were used, as seen in Figure 3.3. The spur gear attached to the wheel is seen on the left and the pinion gear attached to the motor is shown on the right. Also visible in Figure 3.3 is the spacer that extends the spur gear hub. Moreover, a ball bearing is placed within the space between the shaft and spacer. The bearing ensures that the wheels are running smoothly and freely. Detail specifications will be discussed in the following section.

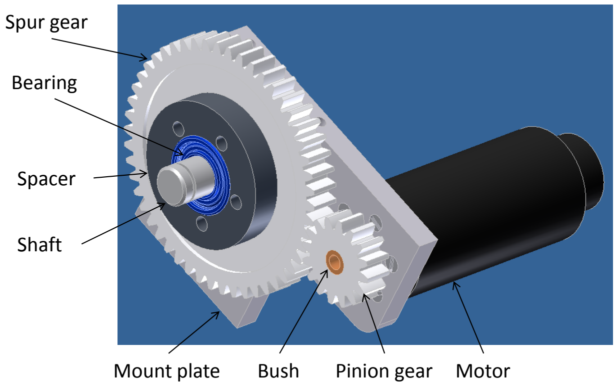


Figure 3.3: CAD model of the wheel drive mechanism (WDM)

3.2 Detailed Mechanical Design

This section describes the implementation of the CAD model. Two iterations were performed during the detailed design phase and the problems that were encountered are discussed. This section also gives the assembly instructions for the drive mechanism. The detail manufacturing drawings for the mechanical system can be found in Appendix A.

3.2.1 First Iteration of the Robot Platform

The first iteration was a prototype manufactured in the local workshop. This prototype highlighted some problems. The problems were mainly caused by the adjustments made to the gear system. The different gears caused the wheels to extend a few millimetres further from the base plate, making the dimensions of the robot larger than allowed. This was improved by shortening the shaft and embedding the spur gear into the wheel. The gear hub can also be machined off in order to gain some extra space.

Another unexpected issue that was encountered was that the o-rings used on the rollers were too thick. It prevented some of the rollers from rolling and therefore, the wheels could not freely slide sideways. Thinner o-rings solved this problem. Lastly, when the electronic system was designed, it was found that there was not enough space to fit the battery and all the electronic modules on the top plate. This problem was also resolved by mounting an extra plate on the top plate with spacers in between. Figure 3.4 shows the addition of the extra plate to the CAD model.

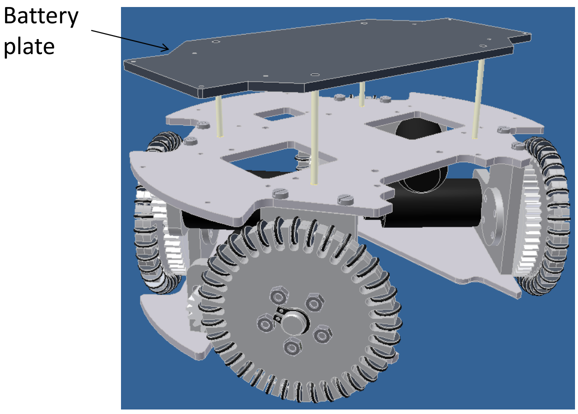
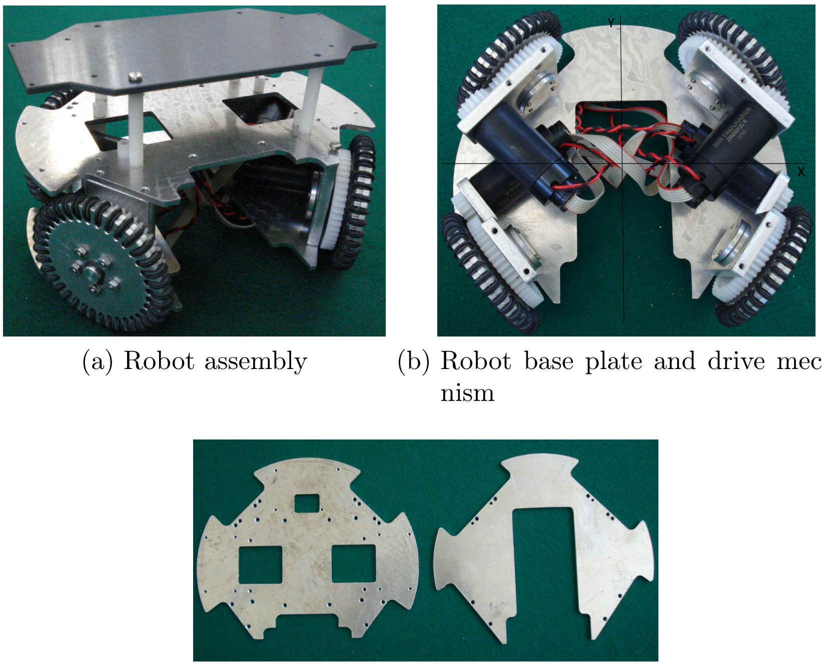


Figure 3.4: CAD model with battery mount plate

3.2.2 Second Iteration of the Robot Platform

The improvements were incorporated in the second iteration of the mechanical design. Figure 3.5 displays the final design of the robot. The complete assembly is seen in Figure 3.5a while Figure 3.5b shows only the chassis and drive system. The base and top plate, seen in Figure 3.5c, were laser cut from 3 mm aluminium. Aluminium was used to minimise the weight, ensure cost effectiveness and ease of manufacturing. Furthermore, the rear wheels in Figure 3.5 are spaced at 45 ° from the imaginary horizontal x-axis through the middle of the base plate. The angle between the front wheel shaft and the x-axis is 37 °. This difference in angle provides the extra space in the front of the robot to fit the kicker and dribbler. Figure 3.5 also shows the additional PVC plate mounted above the top plate with 35 mm spacers to make provision for the electronics and battery.



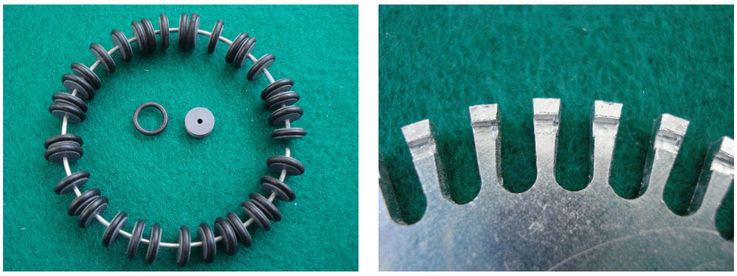
(a) Robot assembly

(b) Robot base plate and drive mechanism

(c) Top plate (left) and base plate (right)

Figure 3.5: Final mechanical system

Figure 3.6 and 3.7 displays the ODW in more detail. The rollers and o-rings can be seen in Figure 3.6a. The 36 PVC rollers were machined in the workshop of the department of Mechanical and Mechatronic Engineering. A rubber o-ring, thickness 0.6 mm, with an inner diameter of 7 mm was slipped onto each roller before it was stringed on the wire ring with a cross-sectional diameter of 1.5 mm. Both the back and front plates were also laser cut from aluminium. Thereafter, they were machined to create a 0.75 mm (half of the wire diameter) groove across the teeth of each of the plates. This is seen in 3.6b.

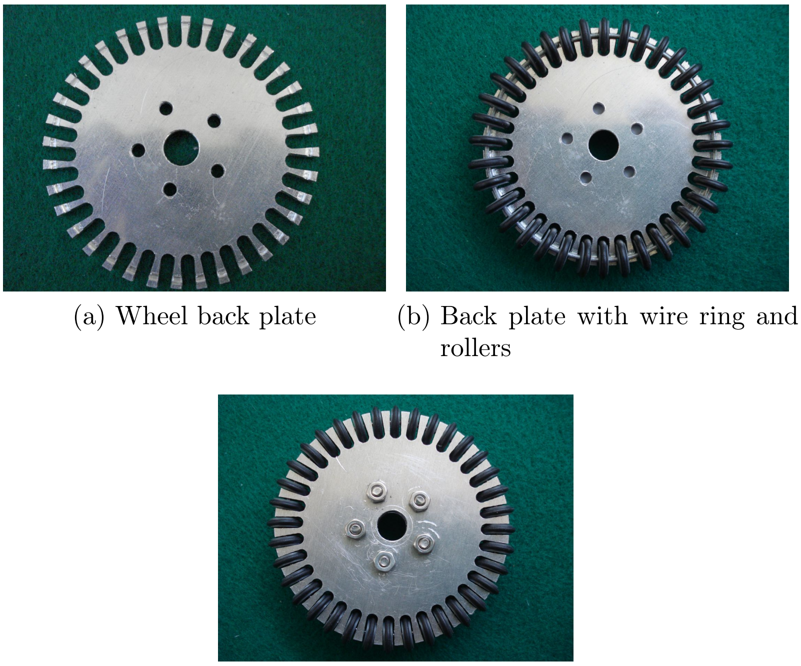


(a) Small rollers with rubber o-ring grip

(b) 0.75 mm Grove in both wheel plates

Figure 3.6: Final omni-directional wheel design

Figure 3.7a shows the back plate. The wire ring fits neatly into the groove on the back plate, shown in 3.7b, and the front plate holds the ring into place. Figure 3.7c shows the assembled ODW.



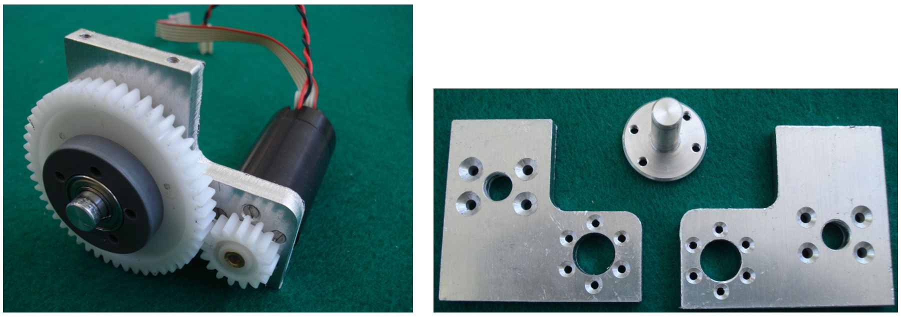
(a) Wheel back plate

(b) Back plate with wire ring and rollers

(c) Assembled omni-directional wheel

Figure 3.7: Final omni-directional wheel design (continued)

The WDM consists of the motor and the two spur gears attached to the wheel mount plate, as seen in Figure 3.8a. The Delrin moulded spur gears were sourced from local suppliers. The large spur gear has 50 teeth and a pitch diameter of 50 mm, while the pinion gear has 16 teeth with a pitch diameter of 16 mm resulting in a gear ratio of 1:3.125. Figure 3.8a also shows the PVC hub extension and spacer, holding the 5 mm × 16 mm ball bearing. The motor, which is also seen in the 3.8a, will be discussed in the electronic design chapter. The wheel shafts were manufactured from aluminium in the workshop at the Department of Mechanical and Mechatronic Engineering, while the rear and front wheel mount plates were laser cut from aluminium. This is seen in Figure 3.8b. Refer to Appendix A for the full mechanical design drawings.



(a) Wheel drive mechanism assembly (b) Rear and front wheel mount plates, and shaft

Figure 3.8: Wheel drive mechanism (WDM)

3.2.3 Assembly Instructions of the Wheel Drive Mechanism

The robot's mechanical system was designed for easy assembly. The assembly can mostly be done intuitively by observing the assembled chassis. However, the order of the construction of the drive mechanism is not so clearly visible from pictures and drawings. Therefore, the exploded view in Figure 3.9 depicts the procedure to assemble the WDM.

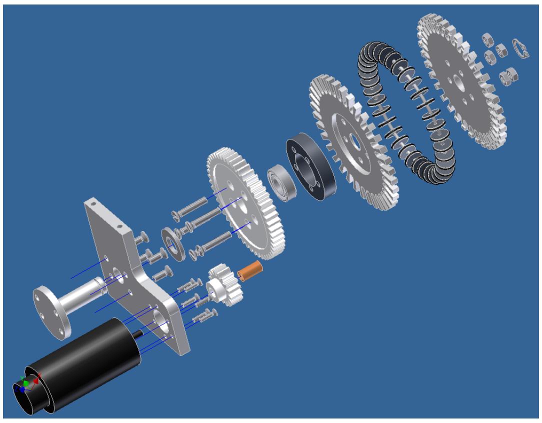


Figure 3.9: Disassembled wheel drive mechanism (WDM)

The procedure is split into two phases. The first phase is to assemble the wheel itself. This includes everything on the right side of the large spur gear. Starting from the left, the bearing is first placed inside the spacer. Then the spacer and bearing slips into the groove in the back wheel cover. Thereafter, the wire ring with the rollers is placed into the slot on the opposite side of the back wheel cover. The ring is held into place with the front wheel cover. Only now can the large spur gear be attached to the wheel with 12 mm M3 screws. These screws go through all the parts and are tightened with nuts on the outer side of the wheel.

The second phase in the assembly process is to attach everything to the mount plate. The motor is fastened to the plate with M2 × 8 mm screws. Thereafter, the pinion gear and bush slides onto the motor shaft and is fastened to the shaft with super glue. Next, the wheel shaft is inserted from the motor's side and fastened as seen in Figure 3.9. An 8 mm washer is placed onto the shaft, thereafter, the complete wheel assembly also shifts onto the shaft, and kept in place with a circlip.

3.3 Conclusion

To ensure that all the requirements have been met, a requirements allocation table was drawn up. The first requirement (MR 1), to develop an ODM, was met with the design of the ODWs and chassis base and top plate. MR 2, to minimise the weight was accomplished by using light materials such as aluminium. MR 3, which was to leave enough space for the kicker and dribbler, was realised by shifting the front wheels further apart. Finally, MR 4 set the dimensional limits of the robot platform to be 180 mm by 150 mm. This was attempted by designing according to the dimensions in the CAD model. However, due to different gears used, the final measurements showed that the robot is still 184 mm in radius and 115 mm in height. This is to be considered as a future improvement and will be discussed further in the chapter 7.

Table 3.1: Requirement allocation table for the mechanical system

Requirement	Method	Result
MR 1	Using four omnidirectional wheels and motors	Achieved
MR 2	Using aluminium parts and Delrin gears	1.48 kg
MR 3	Spacing the front wheels further apart	50 mm × 50 mm × 105 mm
MR 4	Custom design	184 mm × 115 mm

Chapter 4

Electronic Design

This chapter will describe the design process of the electronic system. The process is divided into a preliminary design and a detailed design section. In the preliminary design section the system is described on a conceptual level. In the second part of this chapter each functional unit is again described, this time in full detail in terms of the final design specifications. Where applicable, a trade-off study of possible solutions is discussed and justification for the final selection is given in the detail section. This chapter concludes with a requirements allocation table in which each functional unit in the electronic system is linked to the electronic requirements.

4.1 Preliminary Electronic Design

This section first discusses the requirements of the electronic system that corresponds to the second secondary objective of this research. Thereafter a predesign functional analysis is conducted. This analysis consists of a description of the electronic system in the form of a functional architecture diagram which describes the function of each sub-component in the system.

4.1.1 Electronic System Requirements

Following from the objective related to the electronic system in Chapter 1, the following requirements for the electronic system were defined. Each electronic requirement is numbered and will be referred to as ER and its corresponding number. The electronic system is required to:

ER 1. Be capable of driving the wheels of the omni-directional platform individually.

ER 2. Be able to implement both speed and position control on each wheel.

ER 3. Consist of an on-board processor capable of high-level, autonomous control.

ER 4. Enable the robot to communicate wirelessly to other robots as well as an off-field computer.

ER 5. Make provision for the attachment of additional sensors or modules, either custom or industry standard, that might be required for further development.

ER 6. Have as low as possible power consumption to operate continuously for at least one half of a match, which is 10 minutes

4.1.2 Functional Architecture

In order to meet the requirements stipulated above, the electronic system was first designed at a conceptual level by defining electronic functional units (EFUs). Figure 4.1 displays how the EFUs are connected in the functional architecture of the robot's electronic system.

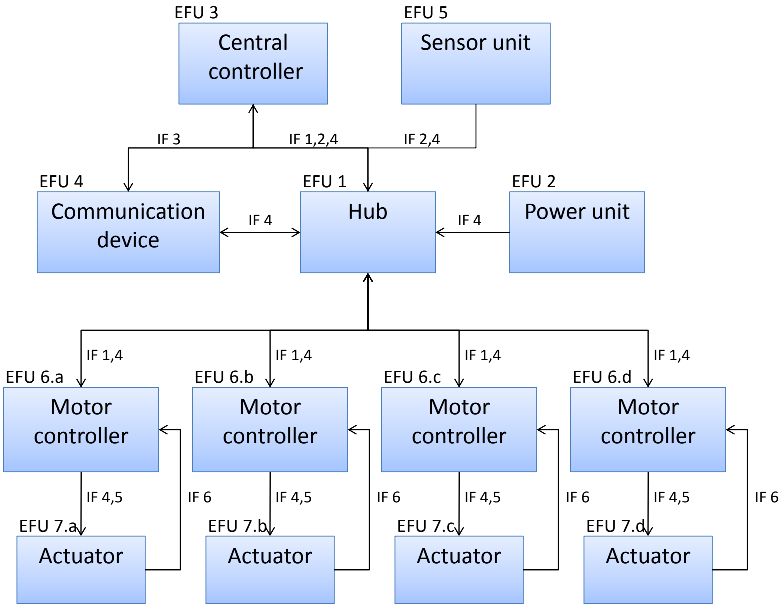


Figure 4.1: Functional architecture of the electronic system

Each EFU will perform a set of functions to achieve the requirements. Also shown in Figure 4.1 are the different interfaces between each of the functional units. A full description of the interfaces is given in Table 4.1. Each EFU will be discussed in more detail below.

Table 4.1: Electronic system interface definitions

Interface	Definition
IF 1	Serial peripheral interface bus (SPI)
IF 2	Inter integrated circuit bus (I^2C)
IF 3	Mini peripheral component interconnect bus (MiniPCI)
IF 4	Electrical power connection
IF 5	Pulse width modulated signal (PWM)
IF 6	Electronic signal connection (Digital)

Hub - EFU 1

The hub, as shown in Figure 4.2, serves as a central connection board. All the functional units are connected to and from the hub via the appropriate interfaces. Furthermore, the hub includes an expansion port that brings all the interface buses together. This enables any external module to be connected to the selected interface bus. The hub also provides a connection to the main controller's general purpose input and output (GPIO) pins that further contribute to the expandability. Lastly, it contains the main power switch from where all power can be cut off.

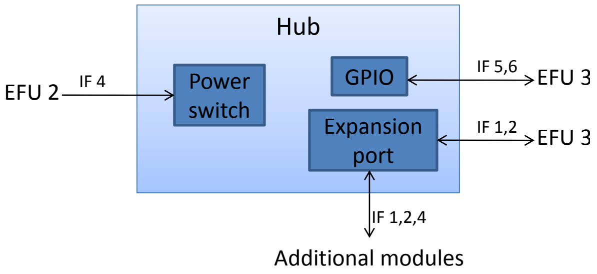


Figure 4.2: Hub - EFU 1

Power Supply Unit - EFU 2

The power supply unit consists of the battery and voltage regulator. It also includes a safety circuit to protect the circuit against over voltage as well as the battery from over-discharge. This can be observed in the diagram in Figure 4.3. All the functional units are supplied with power from this unit through IF 4. To minimise repetition, the power supply unit will be referred to as EFU 2 in all the remaining diagrams and will not be mentioned in the descriptions.

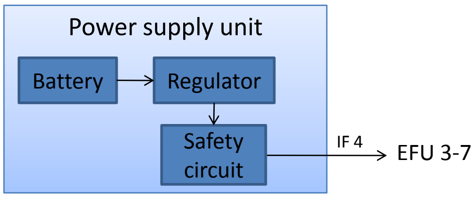


Figure 4.3: Power supply unit - EFU 2

Central Controller - EFU 3

The central controller (CC) unit, which is shown in Figure 4.4, is the heart of the robot's electronic system. The controller is responsible for all high-level control and it is also interfaced to low-level control modules. The CC connects to the communications device (EFU 4), sensor unit (EFU 5) and motor controller (EFU 6) through interfaces 3, 2 and 1, respectively. Note that EFU 5 and 6 are connected via the Hub (EFU 1).

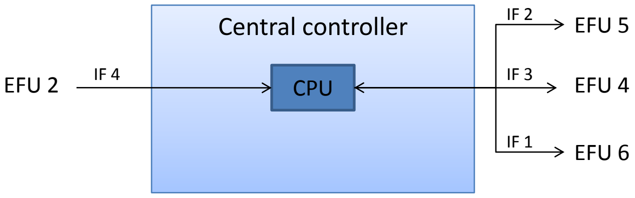


Figure 4.4: Central controller (CC) - EFU 3

Communication Device - EFU 4

The communication device provides wireless communication to other robots as well as the off-field computer. EFU 4 also allows remote control and easy debugging during development. This functional unit consists of a wireless module which is interfaced to the central controller (EFU 3) via the MiniPCI (IF 3) slot on the controller. See Figure 4.5 for an illustration of EFU 4.

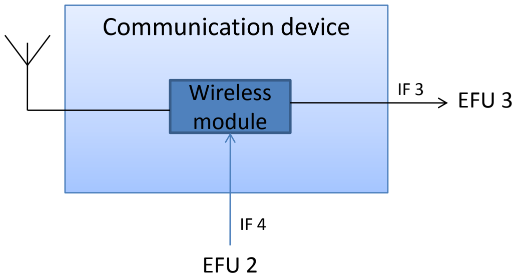


Figure 4.5: Communication device - EFU 4

Sensor Unit - EFU 5

The sensor unit consists of three sensor modules and an analogue to digital converter (ADC) input. The sensor unit makes provision to attach an accelerometer and a compass sensor. Both these sensors are connected to the main controller through the I^2C bus IF 2. An infra-red distance sensor is also included to measure distance to an object in the vicinity in front of the robot. Figure 4.6 depicts EFU 5 and its sub-components.

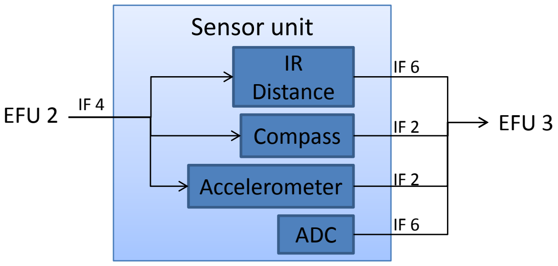


Figure 4.6: Sensor unit - EFU 5

Motor Controller - EFU 6

The motor controller (MC) unit consists of a microcontroller and an H-bridge (see Figure 4.7). The microcontroller can be programmed in various ways to drive the H-bridge, which in turn drives the motor through a PWM signal (IF 5). The microcontroller also makes provision for feedback to implement closed-loop control on the motor. Furthermore, a communication link is established between the microcontroller (EFU 6) and the central controller (EFU 3) through an SPI interface (IF 2).

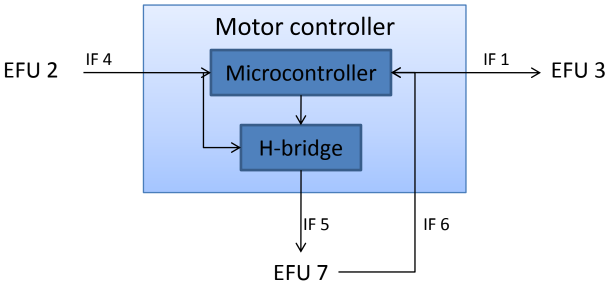


Figure 4.7: Motor controller (MC) - EFU 6

Actuator - EFU 7

The actuator unit, in Figure 4.8, consists of the motors and feedback module connected to the MC (EFU 6). The robot has four motors and also makes provision for connecting additional motors and controllers through the hub.

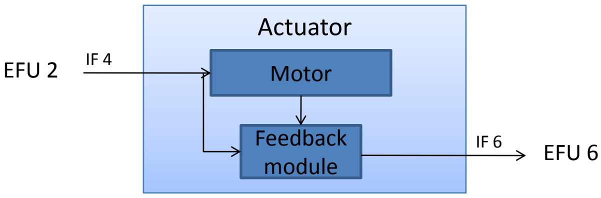
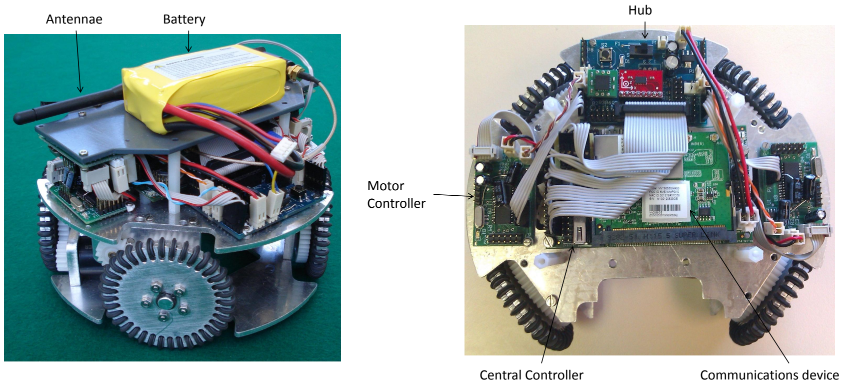


Figure 4.8: Actuator - EFU 7

4.2 Detailed Electronic Design

The task of the preliminary design was to make sure that all the required functionality is included in the design. The next step was to determine the detailed specifications of each functional unit and implement the design. Figure 4.9a is a picture of the fully assembled robot and Figure 4.9b shows all the functional units of the electronic system mounted on the top plate. The detailed design process commenced with a study to determine the state of the art of the current RoboCup teams. Thereafter, each functional unit was designed in detail to fit the requirements for the RoboCup SSL. A description of the detail design of the units is given in this section, with a trade-off study for alternative solutions where applicable.



(a) Fully assembled robot (b) Electronic system mounted on the top plate

Figure 4.9: Robot and electronic system

4.2.1 State of the Art in RoboCup SSL

A review was performed in order to get up to date with the most recent technology used for RoboCup SSL. For this review, the designs of the top teams in the recent SSL and MSL competitions were used. A summary of the design specifications is given in Table 4.2.[1] From this summary, some insights and ideas were gathered before the final design decisions were made. Note that N/A indicates that the information was either not available or not applicable.

The first thing noticeable is that most teams used four wheels, even though only three wheels are sufficient to carry the robot. It is also evident that brushless DC motors are the most popular. Only one team used normal brushed motors. Furthermore, most of the SSL teams implemented an embedded solution using a processor, such as the ARM7 linked to an FPGA, or an FPGA with an internal processor. The FPGA is then configured to implement PI control and other functions. Team 6, however, made use of a Fox board which

[1] Team 1: Zickler *et al.* (2010), Team 2: Wasuntapichaikul *et al.* (2010), Team 3: Maeno *et al.* (2008), Team 4: Nakajima *et al.* (2010), Team 5: Kriengwattanakul *et al.* (2008), Team 6: Laue *et al.* (2009), Team 7: Neves *et al.* (2010), Team 8: Käppeler *et al.* (2010)

is an embedded Linux processor with USB and Ethernet networking capabilities. In contrast, the MSL robots have a laptop onboard with microcontrollers, such as the AVR ATmega128 and PIC 18F258, as secondary processors.

All the SSL robots made use of a proximity sensor to measure distance. No other sensors are installed, except for Team 4 who used a gyroscope sensor. SSL robots rely mostly on the global vision system for feedback, whereas the MSL robots have a number of additional sensors including a local camera, accelerometer and compass. It is also interesting to note that the SSL robots mostly make use of radio frequency (RF) communication because it is easy to implement on a microcontroller. In contrast, the MSL robots all use a wireless ethernet network because of the onboard laptop. The last observation is that Lithium Polymer (LiPo) batteries seemed to be very popular because of their high power to weight ratio.

Table 4.2: Most recent technology used by SSL and MSL RoboCup teams

	RoboCup teams							
Electronic part	Team 1 SSL	Team 2 SSL	Team 3 SSL	Team 4 SSL	Team 5 SSL	Team 6 SSL	Team 7 MSL	Team 8 MSL
Number of wheels	4	4	4	4	4	4	3	4
Motor type	Brushless	Brushless	Brushless	Brushless	Brushless	Brushed	Brushless	Brushless
Motor power output(Watt)	30	30	30	11	N/A	20	150	200
Controller	ARM7	FPGA	FPGA	Renesas	FPGA	Foxboard	Laptop	Laptop
Processor speed	58 MHz	N/A	N/A	28 MHz	N/A	100 MHz	2.2 GHz	2.2 Ghz
Secondary Controller	FPGA	No	No	No	No	AVR	PIC	N/A
Onboard camera	No	No	No	No	No	No	Yes	Yes
IR proximity sensor	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Gyroscope	No	No	No	Yes	Yes	No	Yes	Yes
Accelerometer	No	No	No	No	No	No	Yes	Yes
Compass	No	No	No	No	No	No	Yes	Yes
Wireless communication	RF	RF	RF	RF	RF	WLAN	WLAN	WLAN
Frequency band	902 MHz	2.4 GHz	2.4 GHz	2.4 GHz	900 MHz	N/A	N/A	N/A
Battery type	LiPo	N/A	N/A	LiPo	LiPo	LiPo	NiMh	NiMh
Battery Voltage (V)	N/A	N/A	N/A	14.8	14.8	11.1	12	12
Battery capacity (mAh)	2200	N/A	N/A	N/A	2200	2000	3700	9000

4.2.2 Detailed Specification

This section describes the EFUs that were designed in the preliminary design phase on the basis of their detailed specifications.

Hub - EFU 1

The hub was designed mainly to provide a central connection for all the modules. The hub also exposes the main controller's GPIO pins and communication ports. Hereby, the hub provides an interface for any external modules through the expansion port. Figure 4.10 shows the GPIO circuit and the expansion port circuit that form part of the hub. Furthermore, the hub circuit board also includes the power supply unit (EFU 2) and the sensor modules (EFU 5) in order to save space.

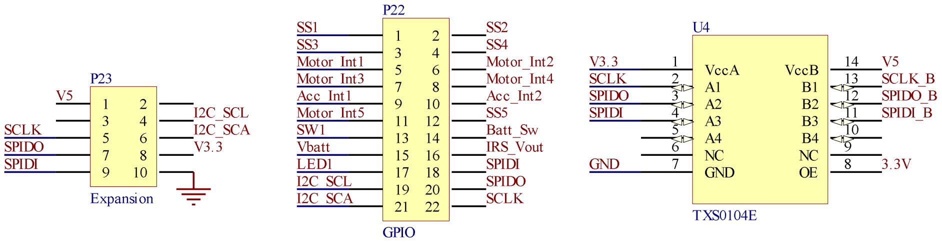


Figure 4.10: Expansion port and GPIO circuit diagram on EFU 1

Some additions were made to the EFU 1 after the first prototype of the design. It was found that the central controllers' SPI port operates at 3.3 V, whereas the motor controllers have a 5 V SPI port. Therefore, a voltage level shifter was needed to convert the 3.3 V pins to 5 V and vice versa. The TXS0104 voltage level shifter was used for this purpose. Note that this was only an additional attachment to EFU 1 and not a second prototype. The voltage shifter was soldered on its own printed circuit board (PCB) and connected to the hub with a ribbon cable. The photo in Figure 4.11 shows the circuit board of the hub and indicates where each subcomponent is situated. In Figure 4.12 the voltage level shifter circuit board can be seen.

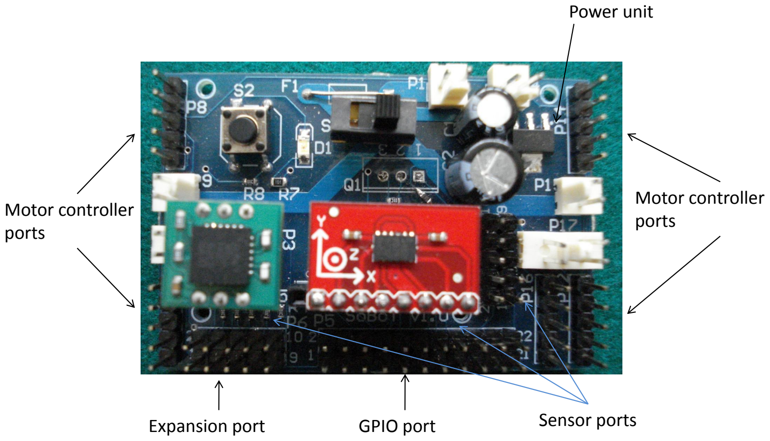


Figure 4.11: Hub - EFU 1

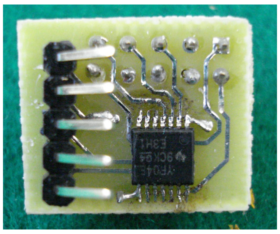


Figure 4.12: TXS0104 voltage level shifter circuit board

Power Supply Unit - EFU 2

Figure 4.13 displays the circuit for the power supply unit. This circuit supplies 3.3 V to the rest of the system by means of the LM 1117 voltage regulator. Furthermore, it includes a MOSFET that is part of a safety mechanism. This safety mechanism can be used to switch off power to the motors in case of an emergency, while the main controller stays powered. Moreover, the battery voltage is fed to the ADC on the central controller in order to monitor the battery level. An LED and a push button are also included to indicate system status and can be used for debugging.

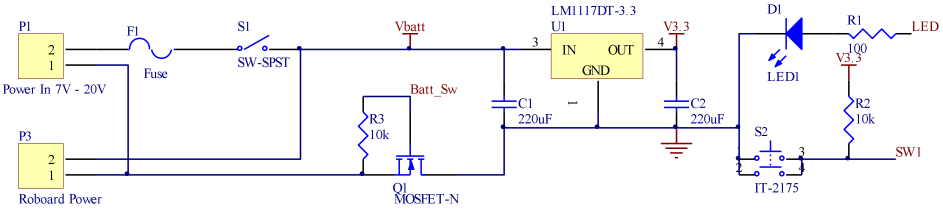


Figure 4.13: Power supply unit circuit diagram - EFU 2

The EFU 2 was constructed as described above. However, the first iteration of the EFU 2 showed some shortcomings. The safety mechanism did not function properly because of a wrongly connected MOSFET, which caused a floating ground level when the MOSFET was not switched off properly. Another problem was that the 12 V from the battery needed to be scaled down with a voltage divider to 5 V, so as not to damage the ADC when reading the battery level. It was also found that a 5 V regulated supply was needed. Therefore a LM 7805 voltage regulator was added to the circuit. These problems were resolved with additional circuits attached to the circuit board and were not included onto the original PCB. However, a second iteration of the power supply unit is recommended to include these components. Figure 4.14 depicts the suggested diagram for the improved power supply unit.

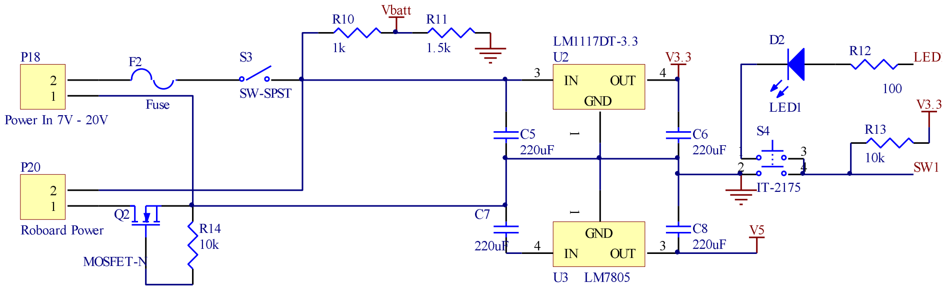


Figure 4.14: Improved power supply unit circuit diagram

In order to fullfil ER 6 to maintain the robot's operation time for at least 10 minutes, the power consumption design was done in the following manner: The motor controller has a maximum supply current of 25 mA, and the three sensor modules together only consumes 11.65 mA. The only components on the robot that draw a significant amount of power from the battery are the central controller and the motors. The central controller consumes 400 mA while idling and 1 A while performing tasks. The maximum continuous current consumption, according to the datasheet, is 1.4 A for the wheel motors and 0.35 A for the dribbler motor. Finally, the communication device has a current consumption of 80 mA. All together it adds up to 7066.65 mA, as shown in Tabel 4.3.

Table 4.3: Power consumption

Component	Maximum current (mA)
Motor controller with H-Bridge	25
Sensors	11.65
Motors	5950
Central controller	1000
Communication device	80
Total	7066.65

Equation 4.2.1 shows how the power consumption was calculated if a 2000 mAh battery is used. According to the calculation 4.2.1, a 2000 mAh battery will supply the robot with 7.06 A continuously for almost 17 minutes. Note that this value is calculated with the peak current consumption and not continuous

consumption. The reason for using the peak value is to over-compensate in the design. The battery would, therefore, perform slightly better than was calculated, since the robot does not draw the maximum of 7.06 A continuously. Also note that the kicker, which will be added in the future, is not taken into consideration yet. However, there is enough space available on the robot to add an additional battery for the kicker if needed.

$$\text{Operating time (min)} = \frac{\text{Battery capacity}}{\text{Continuous current}} \times 60 \tag{4.2.1}$$

$$\frac{2000}{7066.65} \times 60 = 16.98\,\text{min}$$

Central Controller (CC) - EFU 3

For the CC, a number of possible solutions were considered. It is evident from the state-of-the-art review that an embedded solution with a FPGA is very popular for the SSL robots. An alternative to that, also found in the review, is to use a single board computer (SBC). A SBC is a complete computer on one small circuit board with all the functionalities of a normal desktop computer. The alternatives that were considered for the main controller will be discussed and thereafter, the final solution will be described in more detail.

Looking first at the FPGA in combination with a PIC32 microcontroller, this solution has the advantages of high-speed and reliable performance due to the stability of the FPGA. However, the microcontroller, although it is a powerful processor, does have limitations in terms of high-level control. The SBC on the other hand, being a small-size computer, opens up a whole new field of opportunities in robot development. The SBC offers high processing power and the advantage of connectivity through standard interfaces such as USB and Ethernet. Moreover, high-level programming languages including C++ and Java allow the developer to easily implement high-level control, such as path planning or behaviour learning on the SBC. Therefore, five different SBC alternatives were considered, listed in Table 4.4.[2] All the SBCs are Linux compatible and also have storage capability through a microSD card. Furthermore, the operating voltages are all within the desired range of $5 - 12\,\text{V}$. The rest of the specifications in Table 4.4 increase in performance from the left column to the right. Starting from the left, the Foxboard has an ARM9 400 MHz processor and 64 MB of RAM. The Foxboard is a convenient small-size board with standard interfaces which is also listed in the Table. Moving to the next column, the Gumstix has a 600 MHz processor and 128 MB RAM. The Gumsitx

[2] Images in Table 4.4 from left to right: ACME (2011), Gumstix (2010), Roboard (2011), and ADLINK (2011)

is a very small computer on module (COM) solution and also offers a whole range of expansion boards to extend the functionality with Wi-Fi, for instance Bluetooth and GPS. However, the drawbacks of these two solutions are that both lack an SPI interface, and the Gumstix needs an additional expansion board for networking capabilities.

The Roboard is the next alternative with a competitive 1 GHz processor and 256 MB RAM. The Roboard is specifically designed for robotics applications. It provides a wide range of connectivity, from low-level communication with SPI and I^2C bus, to interfaces such as USB and miniPCI. The Roboard offers all these functionalities while still being within 90x90 mm, which is the maximum space available on the robot. The last alternative, the PC/104, is a popular SBC solution with a standard form factor of 90 mm × 96 mm. The standard form factor allows several of these boards to be linked together, stacked on top of each other. However, even though the PC/104 has high processing resources, it is not truly suited for a small-sized robotic application. The PC/104 exceeds the space limit and lacks low-level communication interfaces such as SPI and I^2C. As a result, after analysing all the alternatives, the Roboard was chosen as the best solution having all the necessary resources and fits within the space limitation.

Table 4.4: Alternative solutions for the central controller (CC)

	Alternative solutions				
	PIC32 with FPGA	FoxBoard G20	Gumstix Verdex pro	Roboard RB100	CoreModule 745 PC/104
Properties					
Processor Type		Atmel ARM9	PXA270	Vortex86DX	Atom N450
Processor speed	40 − 80 MHz	400 MHz	600 MHz	1 GHz	1.6 GHz
Memory - RAM	16 − 32 KB	64 MB	128 MB	256 MB	2 GB
Memory - ROM	32 − 256 KB	MicroSD	Micro SD	MicroSD	SATA
Power input (V)	5	5	1.5-6.5	6-24	5
Dimensions (mm)	N/A	66x72	80x20	96x56	90x96
Operating system	RTOS	Linux	Linux	Windows Linux	Windows Linux
Interfaces	USB SPI I^2C CAN QEM	USB Serial I^2C LAN GPIO	USB Serial I^2C GPIO	USB Mini-PCI Serial SPI I^2C GPIO PWM LAN	USB Serial GPIO LAN

The Roboard executes the main control software which is installed on an 8 GB microSD card, together with the operating system. It communicates with the other modules through the various interfaces, which is shown in Figure 4.15. The SPI bus is used to send and receive commands to and from all the motor controllers, while sensors are connected to the I^2C interface to separate the different types of internal communication. The ADC is used to monitor the voltage level on the battery as well as the analogue signal from the distance sensor. Furthermore, the Wi-Fi communications device is connected through the miniPCI slot on the Roboard. This leaves the USB port open for any extra devices, for example a local camera.

Lastly, some of the ports on the Roboard can be utilised during development or debugging of the robot. For instance, a monitor can be connected to the Roboard by means of a miniPCI VGA card, together with a keyboard and mouse. This enables easy development directly on the robot. However, an easier solution is to link the Roboard to the local network through either the Ethernet port or the Wi-Fi device. Then the central controller can be accessed remotely from a PC, also connected to the network, through remote desktop or secure shell (SSH). Finally, the Roboard also provides a functionality to redirect all onboard activities to the serial port and enables a console login mode over the serial port. This is especially helpful in emergency cases where there is no other way to access the controller.

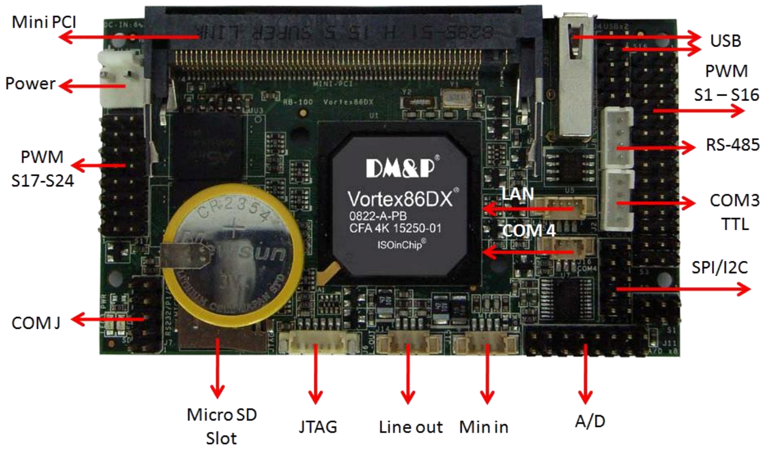


Figure 4.15: Roboard RB 100 for the central controller (CC) - EFU 3 (Roboard, 2011)

Communications Device - EFU 4

The Roboard uses IEEE 802.11 (Wi-Fi) for external communication. The miniPCI slot on the Roboard is used to connect a miniPCI Wi-Fi card, enabling full networking capabilities. The robot can then easily be connected to any ad-hoc or infrastructure wireless network. Figure 4.16 is a picture of the miniPCI Wi-Fi card connected to the Roboard.

Figure 4.16: Communications device - EFU 4 (RobotShop, 2011*c*)

Sensor Unit - EFU 5

The system was designed to be modular and interchangeable. Thus, the sensors were not integrated directly into the circuit design. Sensor modules that can be connected to the central controller via the sensor ports, are rather used. The I^2C bus is also reserved for the connection of sensors. The circuit makes provision for a compass, an accelerometer and an infrared distance sensor. The diagrams below in Figure 4.17, displays the circuits used to connect sensor modules to the system.

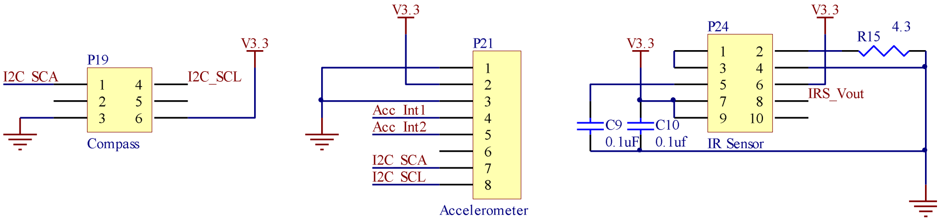


Figure 4.17: Sensor connection circuits - EFU 5

The HMC6352 compass module, displayed in Figure 4.18a, is one sensor module that fits easily into the port. The compass can be used to give information regarding the robot's orientation. Furthermore, a triple axis accelerometer, the ADXL345 (shown in Figure 4.18b), was also connected to the robot through the I^2C bus. This accelerometer is capable of detecting acceleration up to 16 g on all three axes. These sensors can give extra information on the robot's movement in a feedback control system. Lastly, the sensor unit also includes the Sharp GP2Y0D805Z0f distance sensor that can measure distances of up to 60 mm. This can be used to detect whether the ball is in the vicinity of the dribbler. The distance sensor can be seen in Figure 4.18c.

(a) Compass module (b) Accelerometer module (c) Sharp distance sensor

Figure 4.18: The sensors used on the robot, left to right (RobotShop, 2011*b*; Sparkfun, 2011*b*; Pololu, 2011*c*)

Motor Controller (MC) - EFU 6

The specification for the MC must comply with the rest of the components on the robot since most of them were already chosen. For instance, the controller must be able to control a brushed DC motor since that is the motors that is used. The MC is also required to perform closed-loop control. Therefore, the controller must be capable of taking the quadrature encoder feedback from the motors as an input. It must also be capable of supplying the motors with a maximum current of 1.4 A, which is the motor's maximum current consumption. Furthermore, the controller should be able to operate at any voltage below 12 V. Space on the robot is limited and therefore the controller must be as small as possible. The final requirement was that the controller should be able to connect to the SPI interface since that interface was reserved for this purpose and provides fast communication.

Several motor controllers that are available on the market were considered. Table 4.5 shows the controllers that were compared with each other.[3] The first controller that was considered, is the Faulhaber MCDC 3003. This controller is specifically designed to control the Faulhaber DC motors range. The MCDC fullfils all the requirements listed above, but some features were not satisfactory. For instance, the controller only supports two communication interfaces, a serial (RS232) interface and a CAN bus. The serial interface is not suited for connecting more than one motor and therefore, it cannot be used. The CAN bus protocol however, supports several motor connections on one bus, but the Roboard used for the main controller does not support the CAN bus. Furthermore, the physical dimensions of the MCDC, which is 40 mm × 62 mm, are too large considering that four controllers must be fitted onto the robot. Lastly, the cost of this controller at R 2000 was too expensive, again keeping in mind that four controllers are needed. For the above-mentioned reasons this

[3] Images in Table 4.5 from left to right: Phidgets (2010*a*), Pololu (2011*b*), Phidgets (2010*b*), Sparkfun (2011*a*), Pololu (2011*a*), and Faulhaber (2011)

controller, even though it is specifically designed for the Faulhaber motors, was rejected.

Other alternatives were also investigated. The Phidget 1060 has the advantage of two channels on one board, thus being capable of driving two motors. However, the drawbacks are that its continuous current output is too low and it does not have any input for feedback. The Polulo Qik 2s12v10 and Polulo MC33887 were also ruled out for not having a feedback input, even though they are within the parameters of the rest of the specifications. Only two of the remaining controllers are capable of receiving a feedback signal, including the Polulo jrk and Phidget 1064. However, they can only accommodate speed control (no position control). Moreover, the feedback signal type was not matching that of the motor's quadrature encoder. In conclusion, none of the controllers found on the market fulfilled the requirements. It was therefore decided to design a custom motor controller for the robot.

Table 4.5: Alternative solutions for the motor controller (MC)

	Alternative solutions					
	Phidget 1060	Polulo jrk	Phidget 1064	Polulo Qik 2s12v10	Polulo MC33887	Faulhaber MCDC 3003
Specifications						
Feedback	No	Speed	Speed	No	No	Position Speed
Feedback signal	N/A	Tachometer	Phidget encoder	N/A	N/A	Quadrature encoder
Channels	2	1	2	2	2	1
Interfaces	USB	USB serial	USB	Serial	PWM	Serial CAN
Operating Voltage (V)	5-12	6-16	6-15	6-16	5-28	12-30
Continuous Current (A)	1.5	12	14	13	2.5	3
Peak Current (A)	N/A	30	19	30	5	6
Dimensions (mm)	48×56	34×47	76×91	47×54	40×32	40×62
Cost (R)	334	697	816	535	162	2000

The MC was designed to fit the exact requirements mentioned above. It has a very compact design, with dimension of only 45 mm × 25 mm, in order to guarantee enough space on the robot. The PIC 18F2431 microcontroller, shown in Figure 4.21a, was chosen specifically because of its onboard quadrature encoder module and high clock frequency of 40 MHz. The PIC 18F2431 also has an SPI interface so that the motor controller can connect to the central controller through the SPI bus. The PIC also features a serial (RS232) port that can be used for debugging.

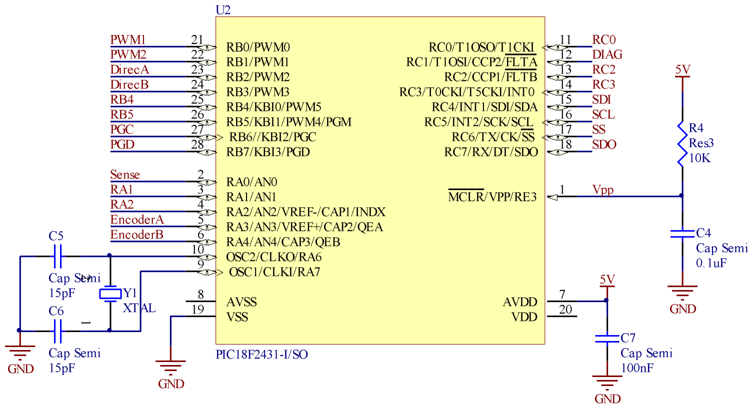


Figure 4.19: PIC18F2431 microcontroller

Furthermore, the microcontroller is connected to a VNH2SP30 H-bridge integrated circuit (IC). The microcontroller drives the H-Bridge with a 20 kHz PWM signal, and the H-bridge in turn drives the motors directly from the battery. The H-bridge is capable of supplying the motors with a continuous current of 12 A, although the copper tracks on the small-size PCB can only endure up to 6 A. The motors, however, have a maximum current consumption of 1.4 A. The H-bridge is also capable of sensing the current consumption in the motor which can be used as a protection mechanism. Figure 4.20 shows the circuit diagram of the H-bridge.

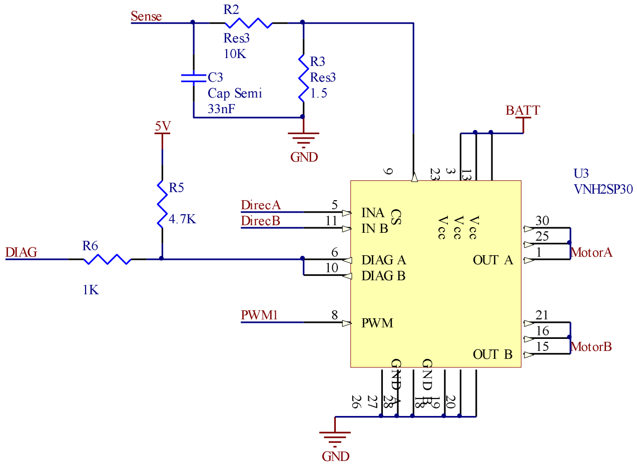
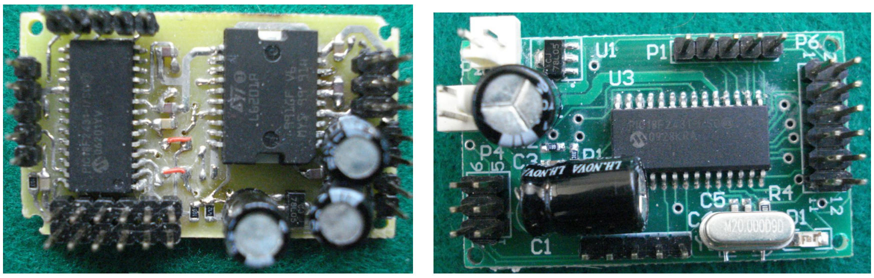


Figure 4.20: VNH2SP30 H-Bridge

The first MC was only a rapid development prototype. The purpose of the first iteration was to build a low cost version of the controller as quick as possible in order to test the concept and identify problem issues with the design. The first prototype used the L6201 H-bridge IC which was changed to the VNH2SP30 in the second version, because of a higher current rating. Figure 4.21a is a photograph of the first prototype and the second iteration is shown in Figure 4.21b. The second iteration also experienced some problems. The electromagnetic field, or "back EMF", from the motors caused the PIC to reset. This was solved with two additional capacitors placed on the PCB. It is recommended that these capacitors are included in the next iteration of the design. Refer to the circuit diagram in Appendix B for the improved circuit diagram.



(a) First prototype of the motor controller

(b) Second prototype of the motor controller

Figure 4.21: Motor controller (MC) - EFU 6

Actuator - EFU 7

The motors need to be small enough to fit into the framework, yet strong enough to accelerate the weight of the robot. The original mechanical design from Bremen University made use of the Faulhaber 2342CR brushed DC motors. These motors were therefore already sized for the framework and, to simplify the design process, they were used for the new design as well. The brushed DC motors are also easy to control by using a PWM signal and an H-bridge. The Faulhaber motors have a maximum output power of 17 W, a maximum speed of 7000 rpm and operate at 12 V. They also have a built-in 512 lines per revolution magnetic encoder, with a quadrature signal output. Figure 4.22 is a photo of the Faulhaber 2342CR motor.

4.3 Conclusion

To ensure that all the requirements for the electronic system are met, a requirement allocation table was set up, which is displayed in Table 4.6. The table maps each one of the functional units to the requirement it is supposed

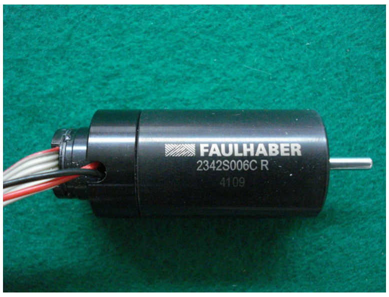


Figure 4.22: Faulhaber 2342CR brushed DC motor - EFU 7

to fullfil. The left column shows the electronic requirements (ER) as mentioned in the beginning of this chapter. The rest of the columns display each of the functional units in the system architecture, shown in Figure 4.1. A star in the column indicates that the functional unit in that column fullfils the corresponding requirement.

The hub (EFU 1), apart from bringing all the connections together, fulfils the purpose of making the system extendible, according to ER 5. The power unit (EFU 2), with the battery included, fulfills ER 6 of having an operating time of at least 10 min. EFU 3, which is the main controller, provides the onboard processor according to ER 3 and also adheres to the extendability requirement in ER 5. Furthermore, the communications device (EFU 4) fulfils the fourth requirement by enabling networking capabilities. The sensor module EFU 5 serves the purpose of providing connectivity to additional sensors required in ER 5. It is also required to control each wheel individually (ER 1) and implement closed-loop control (ER 2), which is fulfilled by the motor controllers (EFU 6a-d) and the actuators (EFU 7a-d). Lastly, the reader is referred to Appendix B for detail electronic diagrams.

Table 4.6: Requirement allocation table for the electronic system

	Electronic Functional Unit						
Requirement	EFU 1	EFU 2	EFU 3	EFU 4	EFU 5	EFU 6	EFU 7
ER 1						*	*
ER 2						*	
ER 3			*				
ER 4				*			
ER 5	*		*		*		
ER 6		*					

Chapter 5

Software Development

The software system consists of three main components: the firmware, software driver and middleware. Each of these software components will be discussed with respect to their development and function in the system. This chapter first states the requirements for the software system. Thereafter, the software architecture gives an overview of the whole system and the different software layers. The three components are then described. Finally, a discussion on the distributed design of the system is given. The chapter concludes with a requirement allocation.

5.1 Software Requirements

Requirements for the software system were derived from the third secondary objective of this research. The requirements are numbered and will be referred to as software requirement (SR) and its corresponding number. The software system is required to:

SR 1. Implement the speed control algorithm that controls the motors (on the firmware).

SR 2. Execute a communication protocol that enables communication between the motor controllers and the central controller.

SR 3. Provide a software application interface to the hardware for easy future development.

SR 4. Provide a communication platform for a multi-robot distributed environment.

5.2 Software Architecture

From an overview perspective the software system consists of a number of layers. Each layer implements a different level of abstraction. The blocks in Figure 5.1 represents these abstraction levels from the lowest to the highest, with each level building on top of the previous one. All the blocks coloured in blue represents software layers. Some of the layers were built from already available software libraries. These layers are shown in the light blue blocks, while the dark blue blocks represent the software layers that has been developed for this research and will be described in this chapter.

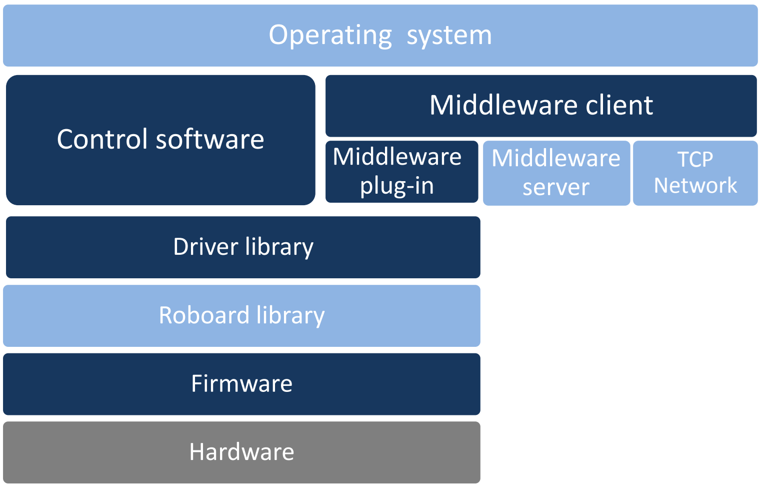


Figure 5.1: Software architecture

The first level of abstraction is found within the firmware. The firmware runs on the microcontroller situated on each of the MCs. The firmware grants access to the motor position and speed through a communication protocol. The next software level is the Roboard library. This library is shipped with the Roboard (EFU 3) and is developed specifically for the Roboard's hardware components. It consists of a set of functions that can be used to access all the communication interfaces on the Roboard. Built on top of the Roboard library is the driver class library. This class provides an abstraction to all the functionalities on the robot, including the motors, kicker, dribbler and sensors. It can be seen as a software application interface (API) which a developer can use to control the robot and implement high-level control algorithms.

The high-level control algorithms form the next level of abstraction indicated by the control software block. Apart from the operating system, this is the highest level of abstraction. Algorithms such as path planning, obstacle avoidance, localisation, etc., are implemented at this level. Lastly, the levels mentioned up to now would be sufficient for autonomous control of a single robot, but one more component is needed in a MRS environment, namely the middleware.

The middleware is presented alongside the control software, indicating that they are at the same level of abstraction, yet, they are used for different purposes. The control software would be used to control a single robot, for instance in a testing scenario, whereas the middleware is required in a multi-robot distributed environment. The middleware includes a server application that hosts two main components. The first is the middleware plug-in, which is another abstraction level above the driver library. The plug-in simply provides the middleware with an interface to the driver library. The second component in the middleware is the client software. The client software replaces the control software in a distributed environment and operates at the same level of abstraction. The important component of the middleware is the network layer. This enables the system to communicate with the outside world and thus creates the possibility of a distributed environment.

Finally, the operating system is depicted as the highest level of abstraction. In reality however, the operating system plays a role in each one of these abstraction layers, behaving as a mediator between the levels. The operating system installed on the robot is Linux Ubuntu 10.04. Evidently, all the other software layers are also Linux-based. Linux was specifically selected as the development platform because it is open source and has good support for software development. The remaining sections of this chapter will describe each software layer in more detail.

5.3 Firmware Design

The firmware was written in C using the Custum Computer Services'(CCS) C compiler v4 and Microchip MPLAB v8.36 integrated development environment (IDE). The main purpose of the firmware is to control the velocity of the motor, using the feedback from the quadrature encoder on the motor. It also implements a simple communication protocol over SPI in order to receive commands from the central controller (CC) and send requested data back to the CC. The firmware was designed to be simple and efficient to ensure a stable and robust control loop. The program was implemented in a state machine, which is shown in Figure 5.2.

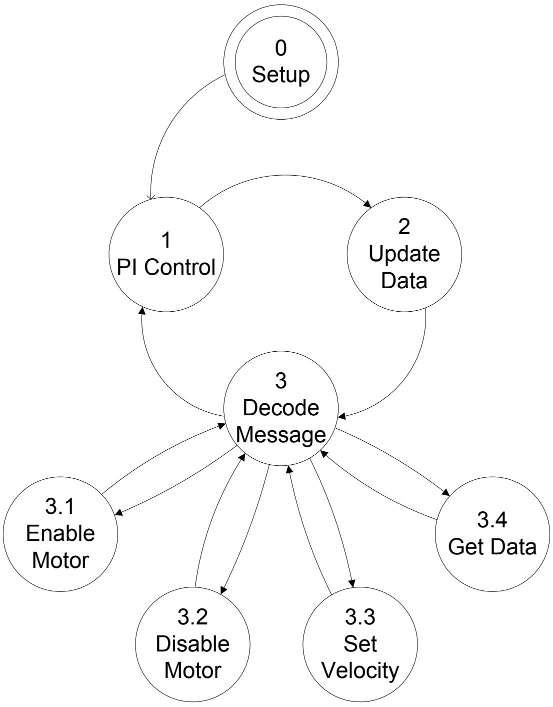


Figure 5.2: Firmware state diagram

The state machine has a starting state, a three-state main loop, and four sub-states. The *Setup State* (State 0) is only executed once, as soon as the controller is powered up. The setup process consists of declaring the global and local variables, configuring the I/O pins and initiating the PWM signal parameters and the PI gain constants. It also calculates the conversion constants that are used to convert between speed and PWM signal as well as converting the counts on the encoder to actual motor speed and position. By calculating these constants beforehand, the performance of the main control loop is increased. The constants are calculated from the geometry parameters of the motor and wheel which can be set in the firmware.

After the *Setup State*, the program goes into the main loop which will keep on executing as long as the power is turned on. The main loop has three states: namely, *PI Control*, *Update Data* and *Decode Message*. State 1, *PI Control*, executes the control function that measures the current motor speed and adjusts the output accordingly to reach the set point. The PI gain constants were determined beforehand by trial and error until satisfactory results were obtained and set in the firmware as fixed parameters. For this trial and error process, an extra functionality was built into the firmware which allows the tuning of the PID gains. Once the tuning is done, this function is disabled in the firmware and will not execute at runtime. More detail on the tuning of the control loop will be given in the next chapter.

The firmware then moves on to the *Update Data* (State 2). In this state the current speed and position of the motor is read, converted and stored in the output buffer. Finally, the third state that the program enters into, is *Decode Message* (State 3). In order to understand the workings of the Decode Message state, the protocol used for communication between the main controller and the motor controllers must first be explained. The protocol has a master/slave architecture over an SPI bus. The motor controller is the slave device and only responds to commands sent by the main controller. Commands are sent in a package of up to seven bytes. Regardless of the number of data bytes in the package, it is always sent in the same order and always initiated by the master. The use of the SPI interface allows full duplex communication; meaning that for every byte sent by the master the slave sends one byte back simultaneously. However, the firmware can also operate in half duplex mode when necessary. Nevertheless, the data in the packet stays the same. Table 5.1 shows the contents of each byte in the data packet.

Table 5.1: Data packet byte allocation

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Bytes sent by CC	Command	Speed	Speed	Direction	0	0	0
Bytes received by CC	0	Position	Position	Direction	Speed	Speed	Direction

Byte 1 is the command byte sent from the master. The command determines what action the MC must perform. Four different actions are defined as sub states of the *Decode Messages* state. These actions are *Enable Motor*, *Disable Motor*, *Set Velocity* and *Get Data*. As soon as the program enters the *Decode Message* state, it will check whether a data packet has been received. If that is the case, the task according to the command byte will be performed. In the case of *Motor Enable* and *Motor Disable* commands, bytes 2 to 7 are discarded. These commands ensure that the CC always has full control over the motors. If the motors are enabled during the start-up process of the CC, glitches on the data lines can cause the motors to malfunction, leading to motor runaway. The Enable and Disable functions prevent this and act as a safety mechanism.

The remaining two available commands are the *Set Velocity* and *Get Data*. If a *Set Velocity* command is received, bytes 2 and 3 contain the speed, which is the input to the PI control loop, and byte 4 sets the direction. Note that even though bytes 5 to 7 contain no data, it is still sent because there is data flowing in the opposite direction in the case of full duplex operation. For half duplex operation these bytes are discarded. For every byte received, the MC loads the current measured position and speed data and sends it to the CC in full duplex mode. This minimizes communication overhead, while having to wait for a response after data is requested. Thus, in full duplex mode, when the *Set Velocity* command is called in the CC the *Get Data* is also executed at the same time. However, in half duplex mode the *Get Data* command must be called separately. This is used in cases where the master only needs to read the current speed and position and not to set a new set point.

5.4 Driver Class Library

The robot hardware driver was written in a C++ class library. The library contains all the functions necessary to control the robot in the public member functions of the class. The use of a class library also makes it convenient for the developer to add the class into an existing application. All the functions that are available to the developer in the software application interface (API) are shown in the unified modelling language (UML) diagram in Figure 5.3.

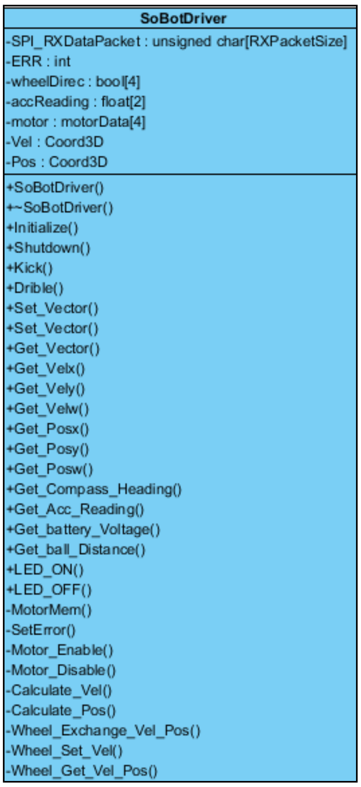


Figure 5.3: Driver class UML diagram

The name of the class is SoBotDriver (SoBot is short for Soccer Robot). The smaller section at the top as well as the other entries with a minus sign next to it, are the private class parameters and functions. These are only used internally by the class. The lower section shows the public member functions that are available for use. The functions called SoBotDriver are the class constructor and destructor. The constructor is invoked automatically when the class object is created and opens all the communication ports on the CC. The destructor will also automatically close the ports at the end of program execution.

Only four functions are essential to the control of the robot: *Initialise*, *Shutdown*, *Set Vector* and *Get Vector*. The *Initialise* member function enables all the hardware on the robot, for instance the motors and sensors. It is very important to call this function before any other functions are called. The *Initialise* function is separate from the constructor so that the developer can control whenever the hardware, for instance the motors, are enabled. It also prevents motor runaway caused by glitches on the data lines during the start-up sequence. The *Shutdown* function should always be called at the end of program execution to ensure proper disabling of the hardware.

The most important function is the *Set Vector* function. This function takes a 2-dimensional(x and y) translational velocity vector and a rotational velocity as parameters. From these values the individual wheel velocities are calculated using the formula in Equation 2.2.2 derived in Section 2.2. These velocities are then sent to the corresponding motors to drive the robot in the desired direction. Note that the vectors are relative to the local coordinate system of the robot. Moreover, the function does not take a speed value as input, but calculates the speed from the resultant vector between the x and y parameters. To stop the robot, the developer has two options. The first is to simply call the function with all the parameters set to zero. The second option is to use the override function with an additional time limit parameter. The robots will then stop automatically after the time elapsed.

Furthermore, the *Get Vector* function is used to retrieve the velocity as well as distance travelled from all four MCs. The values are stored in a data structure from which the robot's current velocity and position is calculated and also stored in the memory. This calculation is done with the kinematic equation 2.2.4, also derived in Section 2.2. It is important to remember that the *Get Vector* function only calculates these values and does not return anything. To retrieve the calculated values, the *Get Vel* and *Get Pos* functions can be used to get the velocity and position, respectively.

The rest of the functions in the class are self-explanatory by their names. These functions can be used to access the sensors on the robot or activate other digital outputs such as the kicker and dribbler. For further detailed explanation of each function, the reader is referred to Appendix C and the accompanied CD for the complete software documentation. The full UML diagram of the software system is shown in Appendix C . A final remark on the drive class library is that the functions only act as the interface to the robot platform and no control is implemented. The implementation of position control and high-level behaviour control is left to the developer.

5.5 Middleware Selection

A survey was done to investigate the availability of middleware in the field of robotics. The reader is referred to Section 2.3 on middleware for a detailed description of the function of middleware. Even though a vast number of middleware solutions are currently being developed, the technology still has some shortcomings in the domain of robotics. Nevertheless, the use of a middleware still improves the functionality of a MRS and is almost unavoidable in a distributed scenario.

Recalling from Section 2.3, middleware is found in any area that is distributed in nature, and not only in robotics. However, this survey only considered middleware in the robotics field since that is the topic of interest. In robotics, middleware can be seen as a sub-division of the broader field of robot software development platform (RSDP). RSDPs are usually developed with certain software functionality and services, depending on the application domain of the target robot. One such functionality is a distribution mechanism, which is enabled by the use of middleware. Middleware, in turn, accomplishes a distribution mechanism through the use of different network transport protocols such as sockets, Common Object Request Broker Architecture (CORBA) language or Internet Communications Engine (ICE). However, this goes beyond the scope of this research and for the middleware selection it was only required that the RSDP supports some form of distribution mechanism.

Table 5.2 gives a summary of the RSDP that was considered in this study and its level of support for a particular functionality. A 0 indicates no support at all, a 1 indicates partially supported and a 2 represents very good support for that particular aspect. All the candidates were measured against the criteria that were determined for this research which is shown in the criteria column. This column indicates whether that aspect was considered as a requirement or not with a Yes or No, or a specification where relevant.

Before each RSDP is considered, it is necessary to understand what is meant with the different aspects in the criteria. Only the ones that might seem am-

Table 5.2: Comparison of middleware solutions

		Middleware								
	Criteria	Aria	Player	Carmen	Miro	Marie	Fawkes	ROS	MRS	URBI
Operating system	Linux	Linux Windows	Linux Windows	Linux	Linux Windows	Linux	linux	linux	Windows	Linux
Programming language	C++	C++	C++	C	C++	C++	C++	C++,Java	C++	
Open source	Yes	Yes	yes	yes	yes	yes	yes	yes	No	Yes
Active development	Yes	Yes	Yes	No	Yes	yes	yes	yes	Yes	Yes
High-level language	Yes	Yes	No	No	yes	Yes	yes	Yes	Yes	Yes
Hardware support	No	0	2	2	1	2	1	2	2	2
Simulator	Yes	1	2	1	1	2	1	2	2	2
Logging facilities	Yes	2	2	2	2	1	2	2	N/A	N/A
Debugging facilities	Yes	1	2	2	2	2	2	2	N/A	N/A
Distribution mechanisms	Yes	1	1	1	2	2	2	2	2	2
Scalability	No	1	0	1	1	1	1	2	N/A	N/A
Documentation	Yes	2	2	2	0	1	0	2	2	2
Graphical interface	Yes	0	2	1	1	2	1	2	2	2
Software integration	Yes	0	1	1	1	2	1	2	1	2
Predefined components	Yes	1	2	1	1	1	2	1	2	2
Score(%)		41	70	46	42	61	N/A	N/A	N/A	N/A

biguous are discussed. Firstly, for the operating system, it is evident from the previous section that RSDP must support the Linux platform and the desired programming language was chosen to be C++. Furthermore it was required that the RSDP is still actively being developed and improved with regular updates. The next requirement, high-level language, refers to the RSDP's support for an integrated behaviour language used for robot behaviour control, in addition to the standard robot hardware control functionality.

The hardware support aspect is an indication of built-in drivers for third-party devices, for example cameras and other sensors. This was investigated in the study but not considered as a requirement since most hardware used on the robot is custom-made which will not be supported anyway. A simulation environment allows testing of complex control algorithms or even analyses of rigid body dynamics. This was also included as a requirement to improve future development opportunities. Logging facilities enables the RSDP to gather information of the system at runtime, such as execution errors and performance statistics, whereas debugging tools can be used to evaluate an application while developing, for instance, graphical representation of sensors.

The distribution mechanism is the component that enables a RSDP to function as a middleware. This should certainly be part of the criteria. Scalability determines how the system performance will change once the number of nodes or robots is increased. In most cases this effect is only considerable in the range of hundreds of nodes in a system. Therefore it is not such an important factor for this study since the number of robots will not be in that range. A software product without proper documentation is often very difficult to implement, especially when it is desired to customise the source code. Proper documentation of the RSDP was considered as an important aspect.

Software integration is the RSDP's ability to integrate with external software for sharing and reuse of existing software. This is a notable aspect that can greatly improve development time and effort. Some RSDPs also include predefined components or software libraries for localisation, path planning or neural networks which also simplifies the development considerably. The last aspect that was considered in the study is an overall score value. This was not part of the criteria but was used as a guideline. The score value was obtained from a comprehensive survey that was done by Kramer and Scheutz (2007) on a number of RSDPs. This value was calculated by investigating the RSDPs on three aspects, namely the support of different features, the practical usability and implementation, and the impact of the system in the field, corresponding with the number of published works.

Amongst the many RSDPs currently available, it was found in the literature that the ones listed in Table 5.2 were the most popular and widely used. These middleware alternatives will be discussed briefly. Firstly Carnegie Mellon Robot Navigation Toolkit (CARMEN), is a robot control software system written for basic single robot navigation (Montemerlo *et al.*, 2004). CARMEN was the first to be ruled out because of its lack of C++ language support and not being actively developed any more. Miro, or Middleware for Robots (Utz *et al.*, 2002) is a distributed object-orientated framework for mobile robot control. However, the documentation proved to be unsatisfactory. Miro was also difficult to implement and only partially supports most of the criteria aspects.

Aria (Advanced Robot Interface for Applications) is basically a set of C++ classes for mobile robot control. Aria was developed for MobileRobots Inc. robots, but is freely available (Activmedia, 2005). Aria also only partially supports, or has no support at all, for most of the required criteria. Alternatively, Fawkes is a component-based software framework for real-time robotic applications (Niemueller, 2009). It is also used by a RoboCup team at the University of Cape Town in South Africa. Although Fawkes seemed to be promising, the documentation was, once again, the stumbling block and therefore, it could not be implemented successfully.

Microsoft Robotics Studio (MRS) is a comprehensive robotics development environment. MRS might have been a viable solution, but it is a commercial product that needs a licence, and can only run on a Microsoft Windows based platform (Jackson, 2008). Therefore MRS was also discarded from the options list. URBI, or Universal Robot Body Interface, is a partially open source robotics development software package from Gostai (Baillie, 2005). URBI is, in reality, a C++ based programming language extended with a behaviour scripting language called urbiscript. URBI was developed particularly to enable event-driven and parallel code execution in robotics applications. Although URBI itself is open source, the simulation environment and other packages must be purchased. This alternative was also rejected since its application focus is different from that of this research.

Finally, only three options remained which were selected as the top three candidates: namely, Player, MARIE (Mobile and Autonomous Robotics Integration Environment) and ROS (Robot Operating System). These middleware solutions were investigated in more detail. The reader is reminded that Table 5.2 contains the detailed information on these alternatives. The full description can be found in Section 2.3. All three RSDPs are competing against each other in all of the aspects that were considered in the criteria. Even though they all showed to be a viable solution, Player was finally selected. From the literature review it was found that Player was the most widely used and established RSDP in the robotics field, which makes the support for development very good. Player also received the highest overall score according to the survey done by Kramer and Scheutz (2007).

5.6 Distributed System

The middleware, Player, forms the basis of the distributed system used for this project. Player is based on a client/server software model. Everything in Player is built around the Player server. The server is sandwiched between four additional abstraction layers and acts as the host that manages all communication above and below it (see Figure 5.4). Additionally, the client and the driver both play an important part, while the proxies and the plug-in layer are the only interfacing components that "glue" everything together.

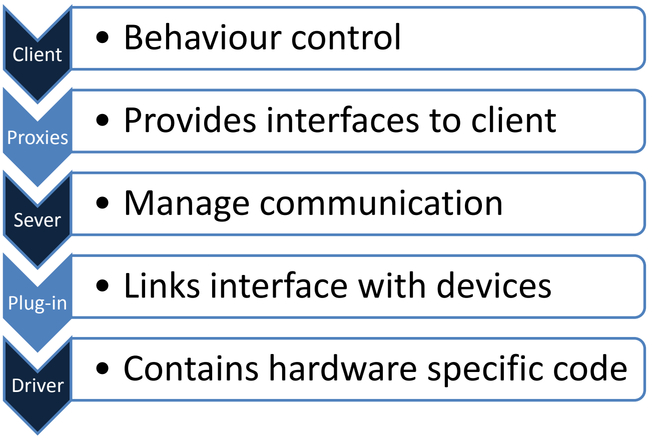

Client
• Behaviour control
Proxies
• Provides interfaces to client
Sever
• Manage communication
Plug-in
• Links interface with devices
Driver
• Contains hardware specific code

Figure 5.4: Software layers above and below the middleware

The driver, as discussed in the previous section, contains the hardware-specific code necessary to access and control all the devices on the robot, whereas the client library contains the high-level robot behaviour control code. Furthermore, the Player server supplies a number of predefined software interfaces, such as a position control interface. The plug-in links each device on the robot with a corresponding interface which grants access to the device. These interfaces are then shared via the server through the so-called proxy. In other words, the driver is linked to the interfaces supplied by the server with the plug-in, while the client is linked to the server through the proxies that provide access to those interfaces.

The server manages all communication by means of Transfer Control Protocol (TCP) sockets as the distribution mechanism, in other words, it uses a network connection. This means that the communication between the client and the server does not necessarily have to be on the same physical platform or even in the same programming language. It operates in the same way as multiple computers on a network. The computers can have different operating systems but still communicate with each other through the TCP network. It is this attribute of Player that was used to implement the distributed system. Figure 5.5 shows two robots and an off-field computer. Each runs a separate instance of the Player server and a client application. All the nodes in the system are connected through a wireless network using TCP sockets and the proxies from Player.

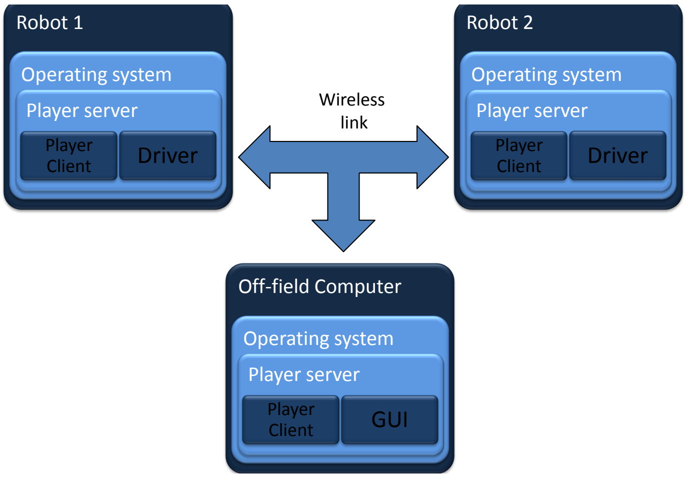


Figure 5.5: Distributed system architecture

The client program on Robot 1 can control itself as well as all on-board devices, and it also has access to the resources on Robot 2. For instance, Robot 1 can access the position sensors or retrieve the current velocity of Robot 2. Additionally, the off-field PC has the same connection to the robots and can also access all the resources on Robot 1 and 2. The off-field computer can, therefore, be used to do runtime debugging or remotely control the robots.

5.7 Conclusion

All the components in the software system were selected, and developed where necessary, in fulfilment of the requirements that were set out in the beginning of this chapter. These requirements are briefly mentioned again and mapped to the software component that fulfils it. The firmware was developed to implement the speed controller as well as to provide a communication link to enable the CC to access the motors. This was in fulfilment of SR 1 and 2. Furthermore, the driver class library was written according to the requirements of SR 3. Finally, the middleware serves the purpose of providing a communication platform amongst several robots. Therefore, the middleware was used to fulfil the requirements of SR 4.

Table 5.3: Requirement allocation table for the software system

	Software component		
Requirement	Firmware	Driver library	Middleware
SR 1	*		
SR 2	*		
SR 3		*	
SR 4			*

Chapter 6

Implementation and Testing

In order to demonstrate the basic functionality of the robot, three tests were conducted. The fist test illustrates the operation of the motor controllers (MC) and at the same time is used to determine the proportional, integral and derivative (PID) gain values for the speed controller. The second test shows how the electronic system controls the omni-directional drive mechanism (ODM) to move the robot in any direction without turning. This test also determines the current accuracy of the movement in an open-loop system. The third test was performed to verify whether non-complex communication can be achieved with the middleware. This chapter concludes with final comments on the test results.

6.1 Speed Control Test

The speed controller on each motor ensures that the robot moves at the desired speed, regardless of the resistance on the wheels due to inertia and friction. This test consisted of speed measurements that were taken to determine the PID gains for the speed controller. The gains had to be tuned such that the robot will reach the target speed in the shortest possible time with a reasonable amount of overshoot. This section describes the procedure that was followed during the testing.

For the speed control test the robot was set up for normal operation with the test application running on the central controller (CC). Furthermore, the PID test option in the firmware was enabled. This activates a built-in function that will take speed measurements at a pre-determined sampling rate when the motors are turning. The built-in function also enables the serial communication port on the MC. The MC's serial port was connected to a PC that is running a custom Matlab script. The script captures the speed measurements sent from the MC and plots the data after each test run. Only one motor controller was used to take the measurements. The firmware programmer was also connected to the MC in order to program the gain values and start the test runs. An iterative tuning method was used, which requires several test runs and changing of the gain values. It would be unpractical to let the robot move with the serial cable and programmer connected. The robot was, therefore, rather placed on a stand to allow the wheels to run freely in the air. The speed control test setup can be seen in Figure 6.1.

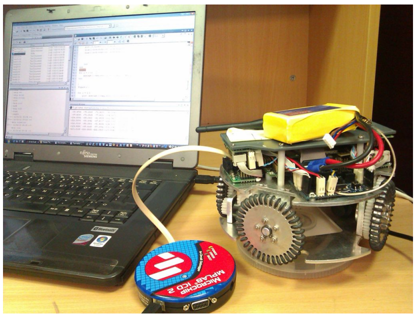


Figure 6.1: Setup for speed control tests

For each test run, the desired PID gains and target speed were set in the firmware and programmed onto the MC. Thereafter, the controller was powered on and fed with a step input command from the CC. This activated the motor and started the measurements. While the motor was running freely, samples were taken every 5 ms with a total of 200 samples, resulting in a 1 sec test run. The target speed was chosen to be 50% of the maximum wheel speed, which is 2000 rpm. Thus, the set point was set to 1000 rpm throughout all the tests. By repeating this procedure several times and adjusting the gains each time, starting with the proportional and then adding the integral gain, the PID controller was tuned to give the desired response.

Figure 6.2 shows the results of five tests with different gain values. The motor speed, in rpm, is plotted on the y-axis while the x-axis shows time in seconds. The noise that is visible on the graph is caused by the tolerances on the encoder measurements. However, the error is small enough to ignore and smooth operation was still achieved despite of the noise. Starting with the bottom red line, the proportional gain was set to 1, while the integral and derivative gains were left at 0. This resulted in an offset in the settling speed at 760 rpm. The next two lines, green and cyan, show how the output shifts closer to the set point as the proportional gain is increased to 2 and 3, respectively. As expected with proportional gain only, the motor reacts faster but does not settle at the desired set point.

The remaining offset was removed by adding the integral gain. The black line represents the motor speed with the proportional and integral gain set to 3 and 0.01, respectively. This gain setting results in the motor speed reaching the desired set point in 0.12 s and settles almost immediately. In order to let the robot have a quick starting speed the controller was tuned to allow a small overshoot. The overshoot was achieved with a further increase in the integral gain to 0.013, leaving the proportional gain at 3. The top blue line shows the motor speed response with a quicker rise time of 0.1 sec, but settling time of 0.4 s due to the 4% overshoot. A further addition of a derivative gain did not improve the results considerably and since the derivative gain normally amplifies the noise on the measurements, it was left at 0. Sufficient results were obtained with only proportional and integral gains of 3 and 0.013, respectively.

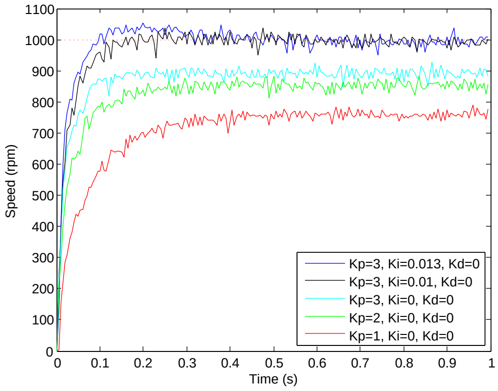


Figure 6.2: Step response for different PID gains

Due to the practical reasons mentioned above, the test was simplified by letting the wheel run freely. However, in order to confirm the validity of these results, a single comparison test was also conducted with the robot running on the floor without changing the gain values. The test was executed in the same manner as the previous ones. In this case, the serial cable and programmer was disconnected and the robot was allowed to spin on the floor. The measurements were recorded on the robot and extracted through the serial cable afterwards. Figure 6.3 shows that the overshoot was reduced to 2.7% if compared to the blue line in Figure 6.2. The reduction in the overshoot could be due to the fact that all four motors are now contributing to the spinning of the robot and the PI controller does not need to push the motor as hard. Nevertheless, this result shows that tuning the PID gains with free running wheels did not compromise the performance of the speed controller considerably. Furthermore, the speed tests revealed that only PI control is sufficient to control the speed of the motors.

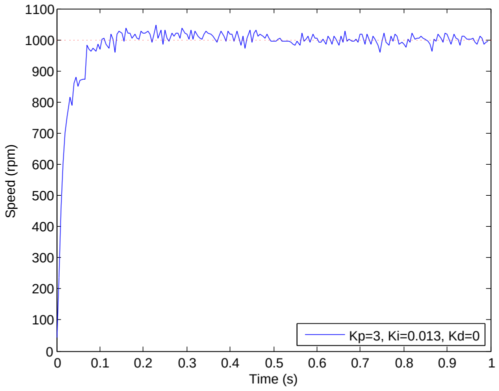


Figure 6.3: Speed measurements for different PID gains

Finally, a last test was performed to investigate the speed response of the robot while running in a straight line on the same surface. The robot was accelerated from standstill to 0.5 m/s with a step input command. This test was repeated 8 times, each in a different direction, and then averaged to get the results shown in Figure 6.4. It is clear that the response differs considerably from the previous tests. The response shows a 28% overshoot, whereas the previous test had a 4% overshoot. The sudden high overshoot can be explained by the fact that more inertia and friction is present when the robot is moving forward, instead of spinning on the spot. This causes the controller to have a slower response time and a higher overshoot to compensate for the inertia. It is therefore concluded that more accurate results will be obtained if the controller is tuned while the robot is moving forward instead of spinning. However, this is a tedious process since all four MC will have to be reprogrammed for every change in the gain values. Another recommendation is to do the initial tuning with a lower overshoot if it is desired. Nevertheless, the current response still shows satisfactory performance for development purposes and demonstrates the functionality of the speed controller.

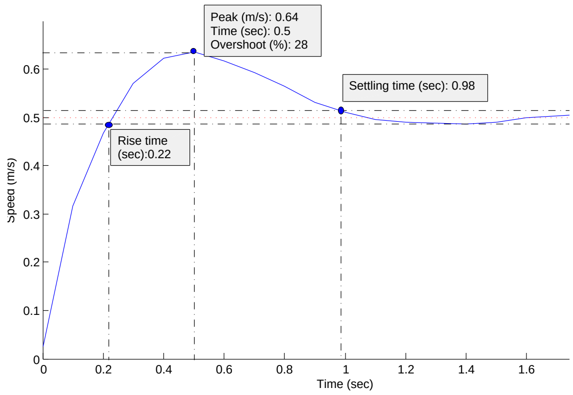


Figure 6.4: Step response for robot with 0.5 m/s set point

6.2 Directional Accuracy Test

The omni-directional robot is capable of moving in any direction, directly from the starting point. However, due to the geometry of the wheels, accuracy in each direction will differ. The purpose of the directional accuracy test was to determine the amount of deviation as the robot moves in different directions.

The second purpose of this test is to demonstrate the integrated functionality of the robot's mechanical, electronic and software systems. A simple movement of the robot in a straight line illustrates the matrix calculation to determine the individual wheel speeds. It also puts the motor controllers, as well as the ODM, to the test. At the same time, the movement of the robot was recorded and used for the last test in the previous section. Note that only speed control was implemented on the robot and not position control. It is thus an open-loop system in terms of position control. Therefore, this test determines the open-loop accuracy of the robot when moving along a straight line.

The experiment was conducted on the felt surface of a standard RoboCup SSL soccer field. The robot was placed in the centre of the field and orientated so that the y-axis was in the length of the field and the x-axis in the width. First, the robot was started up normally with the test application running. Then the robot was connected to a laptop through the Wi-Fi network from where it was remotely controlled. Furthermore, using the driver library, a function was written which takes samples of the robot's position and speed during the test, while saving the data on the microSD card. This position and speed information was gathered from the on-board motor encoders. Note that there was not a vision system available and the external measurements were performed manually with a measuring tape. Figure 6.5 shows the setup for this test.

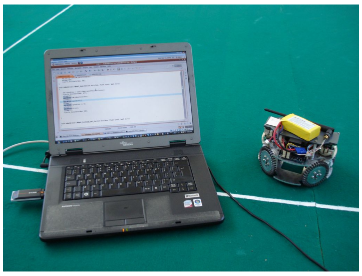
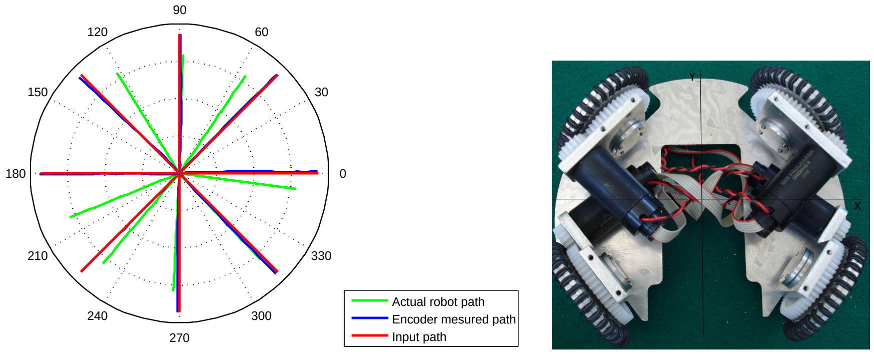


Figure 6.5: Direction accuracy test setup

For each test, the robot received a unity direction vector and a speed value as input. The robot then moved from the origin, which was the centre of the field, in the direction of the vector for 2 s. During this period measurements were taken every 100 ms. The speed input was kept constant at 0.5 m/s throughout the test. The reason for this speed value was to minimise the effect of slip on the wheels, while still maintaining a reasonable speed. Furthermore, the direction vectors were spaced 45 ° appart, starting at the positive y-axis and moving clockwise. After each test, the robot's position from the origin was measured with the measuring tape and manually inserted into the data set. The data was then copied by means of the Wi-Fi connection to the laptop and plotted in Matlab. Finally, before and after every test the robot test application was restarted to reset the position data in the memory. Each direction test was performed six times and the average values are reflected in Figure 6.6a. Figure 6.6b indicates how the robot was orientated during the tests which corresponds with the polar graph in Figure 6.6a (x-axis at 0 °).



(a) Robot direction deviation in degrees

(b) Robot orientation corresponding to the polar graph

Figure 6.6: Robot deviation and orientation

The polar graph shows the results of tests in the eight different directions. The red line represents the input direction vector sent from the central controller, while the blue line represents the path measured by the encoders. The green line is the actual path that the robot followed. From the graph it is clear that the robot deviates from its desired path as it moves further and further, indicated by the offset between the red lines and the green lines. The deviation error, for each direction, is given by the angle between these two lines. Note that the length of the lines are normalised and does not represent the radial distance of the robot's travel. The distance is not of any significance for the direction test since the robot moves in a straight line and the error in direction would stay the same for any distance travelled.

If the lines in Figure 6.6 are investigated, it is very difficult to spot a trend in the behaviour of the robot. One would also expect a more symmetrical graph around the y-axis due to the symmetry in the geometry of the robot. This is not the case. However, these results still remain valid and some conclusions can be drawn from them. The most probable reason for the unevenness is slip occurring between the wheels and the surface. Another reason could be small imperfections in the motor controllers and motors, causing them to be asynchronous. Thus, not starting at exactly the same time causes a small initial jerk in a different direction at the starting point.

The deviation errors for the robot path are shown in Table 6.1. Note that a negative sign implies a deviation in the clockwise direction and a positive sign implies an anti-clockwise deviation. It is evident that the robot is most accurate in the direction of the y-axis, 90 ° and 270 °. The robot only deviates 1.71 ° and 2.97 ° in these two directions respectively, whereas the deviation in the direction of the x-axis is significantly larger, 7.77 ° and 21.88 ° in the 0 ° and 180 ° direction, respectively. The explanation for this behaviour lies in the geometry of the robot wheels.

Recall that the wheels are not perpendicular to each other, but are spaced closer to the x-axis. The wheels are thus more parallel to the y-axis than to the x-axis. If the robot moves in the direction of the y-axis, the wheels are facing the direction of movement and experience more grip on the surface. On the other hand, if the robot moves in the direction of the x-axis, the wheels are almost perpendicular to the direction of movement, having less grip. This means slip can occur more easily and there is less control over the robot, which causes the error in direction.

Table 6.1: Robot path direction and encoder deviation

	Deviation in degrees							
Direction (deg)	0	45	90	135	180	225	270	315
Robot path deviation (deg)	-7.77	10.85	-1.71	-13.22	21.88	4.65	-2.97	-1.74
Encoder deviation (deg)	8.42	-10.54	1.25	14.39	-21.51	-4.68	2.38	0.64

Furthermore, the bottom row of Table 6.1 shows the error in the encoder measurements. Note that the encoder deviation (blue line) is calculated relative to the path deviation (green line). In other words, these values in the third row of Table 6.1, shows the deviation of the encoder measurements from the actual robot path. From 6.6 it is clear that according to the encoders the robot is moving with practically 100% accuracy, even though it is not. This confirms that the major reason for the accuracy deviation of the robot is slippage on the wheels, since the wheels are turning correctly according to the encoders but not relative to the ground.

A last important remark is that during the tests that were performed above, the duration of the battery life was recorded. Over a total of five tests, consisting of 0.5 m/s speed tests, the average battery life was 2.5 h. When the robot was left only on stand-by, the battery lasted 4 hours. Unfortunately no tests were performed to determine the battery consumption during maximum motor operation. However, from the results above, it can be assumed during the competition that the battery would last at least more than 10 minutes.

6.3 Static Communication Test

In order for several robots to operate in a distributed environment, communication between them is vital. It was already discussed in Chapter 5 how Player is used as a middleware to accomplish this task. The purpose of the static communication test is to demonstrate how the middleware accomplishes a communication link between the robots.

In order to perform such a test, at least two robots are necessary. However, only one robot was functional. Nevertheless, since it is only a static test and since the controller on the robot is nothing other than a small computer, a normal PC can be used to obtain similar results. Therefore, a laptop was used to simulate the second robot. The same operating system (Ubuntu) that runs on the robot was installed on a virtual machine on the laptop, together with the middleware server and client application. A wireless link was also established between the laptop and the robot by using Player. Note that this setup is the same as in all the previous cases where the robot needed to be remotely controlled. In this case the middleware is used for communication, instead of a direct wireless link. Figure 6.7 depicts the setup for this test. The reader is referred to Section 5.6 for a more detailed description of the distributed system using Player.

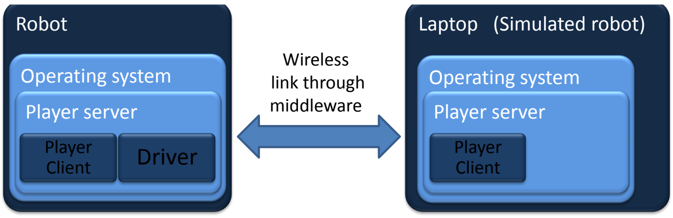


Figure 6.7: Static communication test setup

The client application on the laptop can access the functions in the driver class library through the middleware. Thus, the test was performed by calling the *Set Vector* function from the client application on the laptop (simulted robot), commanding the robot to move. With the *Get Vector* function data was requested from the robot and sent to the laptop. This test proved to be successful and demonstrates a simple data transfer from one robot to another using the middleware.

6.4 Conclusion

The first test showed that speed control was successfully achieved. It also revealed that only a PI controller is sufficient to control the motors. The controllers were initially tuned to have an overshoot of 4%, which resulted in a 28% overshoot due to the inertia. This overshoot will give the robot a faster starting time, bringing the robot up to the set point speed within 0.22 seconds. Furthermore, the direction accuracy test gave some valued insights into the open-loop performance of the robot. The robot has a minimum tracking deviation of 1.71 ° and a maximum error of 21.88 °. It also showed the difference in the accuracy caused by the geometry of the robot. These results will be meaningful to any future development, when implementing a position control system. The battery also showed satisfactory performance by lasting 2.5 h on average during the tests. Finally, a static communication test demonstrated that Player can be used to send and receive data between the robots. A further discussion on these results can be found in the next chapter.

Chapter 7

Conclusion

Chapter 1 stated that the primary objective of this research was to design and manufacture a first prototype of the omni-directional robot for RoboCup SSL for Stellenbosch University. It was also mentioned in the objective that the design should make provision for a distributed control architecture (DCA), in order for the robot to be used as a development platform for research in multi-robot systems (MRS). To accomplish this objective, the research was devided into four secondary objectives. Each of these objectives will be discussed in this chapter by recalling the means by which it was achieved throughout the research. Then the conclusions and the implications thereof are discussed. Also, since the robot platform is meant for further research, recommendation for future work will be given where it is applicable. Finally the chapter will conclude with a final remark on the significance of this work.

The first secondary objective relates to the mechanical design of the robot platform in Chapter 3. It stated that the robot should be capable of holonomic movement. This feature is absolutely essential for the robot to compete in the RoboCup SSL. Therefore, the omni-directional drive mechanism (ODM) was designed in order to achieve this objective. An existing design was used for this purpose and adjusted where necessary. However, the omni-directional wheels (ODWs) were redesigned for easy manufacturing and assembly. The tests in Chapter 6 demonstrated how the drive mechanism enables the robot to move in any direction without changing orientation, thus being a holonomic platform as required from the objective.

Additionally, it was said that the platform should ensure enough space for a dribbler and kicker device. The platform makes provision for these devices with an open space of 50 mm × 50 mm × 105 mm. It is also left open at the rear of the robot to give the opportunity to use a solenoid kicker, since these devices usually need the extra space for the pull back spring. However, other kicker designs, such as a linear actuator, should also fit easily into the availible space. The construction of a kicker and dribbler should be considered for further development. Nevertheless, the extra space was achieved by widening the front wheel placement to have a 37 ° angle with the x-axis, instead of 45 ° like the rear wheels. This adjustment caused some degradation in the performance of the ODM as it was found in the accuracy tests in Chapter 6. The robot had an average deviation of 21.88 ° in the direction of the x-axis. However, it is important to note that this was only an open-loop test. The accuracy will be greatly improved once a feedback system is used to correct this error. This issue will be discussed later on in this chapter with some suggestions on how to improve the accuracy.

Furthermore, the RoboCup rules require that the SSL robots may not exceed a radius of 180 mm and a height of 150 mm. The robot is within the limit regarding the height, at 110 mm without the battery, and only 135 mm including the battery and electronics. This even leaves space for an extra battery if required. However, some problems were experienced during the redesign and manufacturing of the omni-wheels and gears, which caused a few extra millimetres on the sides of the robot. This was improved in the second iteration of the design, although the problem still persisted. The result was that the robot has a radius of 184 mm. This problem can be solved by machining off some of the parts to gain that extra 2 mm on a side. Due to time limitations, it was not done since it did not influence any of the other requirements. However, this is recommended as a future consideration. Finally, although the weight limit was not an initial requirement, it was added as an additional design consideration to maximise the performance of the end product. A total weight of 1.48 kg was achieved by using aluminium as the primary material for the robot.

Secondly, the objective in Chapter 1 that corresponds to the electronic design in Chapter 4, stated that the electronic system is needed to control the ODM. In order to control the ODM, the system should provide functionality for speed as well as position control on each motor. A number of motor controllers (MCs) were considered, although none of the alternatives were satisfactory. The most probable solution was the Faulhaber MCDC motion controller which was designed specifically for the motors that were used in this research. Yet, this alternative was also rejected due to the cost and lack of an SPI interface. Therefore, the MC was designed with the requirements of this research in mind. The first consideration was to design for the position and speed control. The PIC18F2431 contains a quadrature encoder module which simplified this task. It is also capable of having a very high sampling rate with a maximum clock frequency of 40 MHz. Moreover, the MC was designed to have very small dimensions, only 45 mm × 25 mm, in order to fit four of the PCBs on the platform. The speed tests in Chapter 6 also showed how the MCs were successfully used to accomplish PI speed control. The controller was deliberately tuned to have an overshoot, to give the robot a quick reaction time. However, this response can easily be adjusted if desired in future applications. Another future recommendation was to manufacture a third version of the MC and include the capacitors, as mentioned in Chapter 4, which is used to counter the effect of the motor back-EMF.

Furthermore, a very important aspect that was stated in the objectives, is the provision of a distributed architecture. Also, the robot platform should provide the opportunity for further research and development, since it is intended as a test bed for MRS. The former and the latter statements go hand in hand. It was also mentioned in the motivation in Chapter 1, that by using a hybrid system, not only can research in RoboCup SSL be conducted but additional opportunities for future research are created. It was, therefore, a major design consideration to design the electronic system accordingly. Three design guidelines were followed to achieve these objectives. The guidelines were: design for high-level programming language compatibility, design for a wide range of connectivity and design for modularity.

High-level programming languages, such as C++ or Java as opposed to C or Assembler, offers a vast amount of reusable libraries and classes for robot development, especially for research in AI and autonomous behaviour. Therefore, the Roboard RB 100, which is a SBC, was chosen for the central controller to support these languages. Moreover, the SBC supports a number of different connectivity options, such as USB, RS232, SPI and PCI. This wide range of standard interfaces creates the opportunity to expand the system easily, which further contributes to the robot being a research platform. One example of this, that has been implemented for this research, was to use the PCI interface to give the robot Wi-Fi capabilities, thereby, also adhering to the objective that requires communication to the off-field computer, as well as directly between the robots.

Furthermore, the guideline to design for modularity also contributes to the research platform, by making it easy to add or change any of the modules, without compromising the rest of the system. For instance, if at a later stage it is decided to use brushless DC motors rather than the current brushed motors, the MCs can simply be replaced with the desired controller and connected to the CC via the appropriate interface. This is also a recommendation that can be considered for future development. The brushless DC motors can offer a better power output performance over the physical size of brushed motors. Nevertheless, the modular design approach does have some disadvantages. Because the modules are separate from each other, they need to be connected with wires or ribbon cable. These wires can get tangled and cause problems with the communication, especially when long power and signal lines run together, and the impedances are not matched. There are, however, precautions that can be taken to avoid these problems. Therefore, the modular approach remains a more desirable solution.

Thirdly, it was an objective of this research to develop the firmware and software that will utilise the electronics, to implement basic control on the robot. The first level of control, before any other form of control can be implemented, is speed control. Chapter 5 described how the control algorithm for a PI speed controller was implemented in the firmware on the MC. The technique of using a state machine, ensured that the firmware is robust and reliable, since a failure on this level can easily send the robot off track. Again, the speed tests in Chapter 6 demonstrates the successful implementation of the speed controller. The next level of control is position control. In order to accomplish position control, some form of position feedback is essential. The firmware also includes a function capable of reading the position from the motor encoders, thereby determining the position of the robot. However, the results from the accuracy tests in Chapter 6 showed that these measurements were inaccurate between 0.64 ° and 21.51 °. The reason for this is not because of faulty encoders, but mainly because of slippage on the wheels as it was concluded in Chapter 6. Therefore, in order to implement position control, the slippage must be minimised or another means of feedback must be used.

This can be achieved in one, or a combination of the following ways. A different set of PI gains can be determined to slow down the response of the speed controller and thereby reduce slippage. However, this will be a compromise to the quick response of the robot. A more desirable approach would be to use a ramp input instead of a step input. This would gradually increase the speed of the robot and reduce slippage. Another more advanced technique is to implement the algorithms mentioned in Williams *et al.* (2002) and Rojas and Förster (2006), to detect when slippage occurs on the wheels. This information can then be used to discard the faulty readings and improve the measurements from the encoders. Additionally, the encoder measurements can be supplemented with data from the accelerometer. Sensor fusion techniques can then be applied to determine a more accurate position measurement. The most viable solution would be to acquire an external measurement, such as from a vision system. This would give the exact position of the robot even when slippage occurs. Nevertheless, the firmware makes provision for position control, and it is recommended for future development to consider these methods for a position control system.

In addition, the objective required that the software should provide a software application interface to the robot platform. This goes hand in hand with what was stated in the primary objective, that the robot is supposed to be a software development platform for research in robotics. Therefore, the software driver class library was developed as described in Chapter 5. The library offers a number of functions that can be used by a developer to access and control all the hardware on the robot. It is evident that this leaves plenty of space for future development. One suggestion is to implement a path planning and path following algorithm. These algorithms and high-level behaviour control can be implemented with a behaviour language such as XABSL (Extensible Agent Behaviour Specification Language). Furthermore, the implication of the class being written in a high-level programming language, is that a developer can make use of the libraries, algorithms and resources that are already available for robotic applications. For instance, for this research Player was implemented together with the driver library, to accomplish communication for multiple robots in a distributed environment. This was also stated as part of the objectives. Chapter 5 described how Player utilises a TCP network and client/server model to achieve communication between the robots and the off-field computer. The significance of this is, firstly, that it contributes to the distributed approach of this research by means of the TCP network. Secondly, that it adds to the robot being a development platform by means of the client/server model. The latter is because the client/server model allows the robot (server) and the application software (client) to be totally separate from each other, in terms of programming language and physical location, thus giving the researcher more freedom of choice and opportunity for development.

Fourthly, an extra objective was given to, if time permits, develop an additional robot and test the distributed system as a whole, by performing a cooperative task in a dynamic distributed environment. One example of such a task would involve two or more robots to move from one formation to another. For instance, from standing in a line to forming a circle. This is typically what will be required from the robots in RoboCup SSL competitions or most MRS assignments. Evidently, this task must be performed without any support from the off-field computer, other than supplying the vision. Such a task would require the robots to be aware of their own locations, as well as there fellow teammates' positions relative to them. However, this level of control was not possible, since it would require either an overhead vision system or very accurate position feedback from local sensors. Even though a second prototype was indeed developed, neither of the formerly mentioned necessities were available. The distributed test could not be performed and this objective was, therefore, only partially achieved. A simple static communication test was, however, performed. This test demonstrated that data can easily be transferred between the robots in a static environment through Player, thus sharing their resources. In the same manner it would be possible to transfer data between the robots in a dynamic environment to achieve some cooperative task, once a position control system is also available. It is therefore recommended as another future research opportunity to investigate the performance of Player as a middleware in a distributed environment using the developed robot platform.

In conclusion, it is evident that all the required objectives for this research were achieved successfully, with the exception of the additional objective being partially met. However, several suggestions were made on how to improve the position measurements in order to continue this objective in future work. Each design requirement was carefully considered and implemented using a system design approach. This complete design process was documented to give the developer a comprehensive understanding of the robot platform. Furthermore, lessons were learned during the development of this first prototype, and recommendations were made on how to improve it. Also, through this research a number of future research opportunities arose, which was mentioned in this chapter. All in all, this research encompassed three technological fields, mechanical, electronic and software, integrating them to create a versatile development platform for future research in robotics and MRS.

List of References

ACME (2011). Fox board g20. Last visited: January 2011.
Available at: `http://eshop.acmesystems.it`

Activmedia (2005). robotics mobilerobots developer support. Last visited: January 2011.
Available at: `http://robots.mobilerobots.com/`

ADLINK (2011). Coremodule 745. Last visited: January 2011.
Available at: `http://www.adlinktech.com`

Aduthaya, J., Charitkhuan, C. and Bhuripanyo, J. (2006). Distributed control system for small-sized robocup. In: *2006 IEEE Conference on Robotics, Automation and Mechatronics*, pp. 1–7.

Asada, M., Kitano, H., Noda, I. and Veloso, M. (1999). Robocup: Today and tomorrow–what we have learned. *Artificial Intelligence*, vol. 110, no. 2, pp. 193 – 214. ISSN 0004-3702.

Azevedo, J., Cunha, M., Lau, N., Neves, A., Corrente, G., Santos, F., Pereira, A., Almeida, L., Lopes, L., Pedreiras, P. *et al.* (2009). Cambada 2009: Team description paper. Tech. Rep., Transverse Activity on Intelligent Robotics, IEETA/DETI University of Aveiro, 3810-193 Aveiro, Portugal.

Baillie, J. (2005). Urbi: Towards a universal robotic low-level programming language. In: *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 820–825. IEEE. ISBN 0780389123.

Burkhard, H., Duhaut, D., Fujita, M., Lima, P., Murphy, R. and Rojas, R. (2002). The road to robocup 2050. *IEEE Robotics & Automation Magazine*, vol. 9, no. 2, pp. 31–38.

Cincinnati (2010). Video camara. Last visited: January 2011.
Available at: `http://www.uc.edu/ucii/monitor/images/`

Collett, T., MacDonald, B. and Gerkey, B. (2005). Player 2.0: Toward a practical robot programming framework. In: *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005)*. Citeseer.

Côté, C., Brosseau, Y., Letourneau, D., Ra
"ıevsky, C. and Michaud, F. (2006). Robotic software integration using marie. *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, pp. 55–60.

Côté, C., Létourneau, D., Michaud, F., Valin, J., Brosseau, Y., Raievsky, C., Lemay, M. and Tran, V. (2004). Code reusability tools for programming mobile robots. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings*, pp. 1820–1825.

Faulhaber (2011). Faulhaber. Last visited: January 2011.
Available at: `http://www.faulhaber.com/n40966/n.html`

Gerkey, B., Vaughan, R. and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In: *Proceedings of the 11th international conference on advanced robotics*, pp. 317–323. Citeseer.

Goedicke, M. and Zdun, U. (2001). A key technology evaluation case study: Applying a new middleware architecture on the enterprise scale. *Engineering Distributed Objects*, pp. 8–26.

Gumstix (2010). verdex pro coms. Last visited: January 2011.
Available at: `http://www.gumstix.com`

Jackson, J. (2008). Microsoft robotics studio: A technical introduction. *Robotics & Automation Magazine, IEEE*, vol. 14, no. 4, pp. 82–87. ISSN 1070-9932.

Käppeler, U., Zweigle, O., Rajaie, H., Häussermann, K., Tamke, A., Koch, A., Eckstein, B., Aichele, F., DiMarco, D., Berthelot, A., Walter, T. and Levi, P. (2010). Rfc stuttgart team description 2010. Tech. Rep., IPVS, University of Stuttgart, Germany.
Available at: `http://robocup.informatik.uni-stuttgart.de/rfc/www/`

Kitano, H. and Tadokoro, S. (2001). Robocup rescue: A grand challenge for multiagent and intelligent systems. *AI Magazine*, vol. 22, no. 1, p. 39. ISSN 0738-4602.

Köker, K. and German, R. (2007). The robocup f-180 league dedicated system design for performance analyses of distributed embedded systems. In: *Control & Automation, 2007. MED'07. Mediterranean Conference on*, pp. 1–4.

Kramer, J. and Scheutz, M. (2007). Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, vol. 22, no. 2, pp. 101–132.

Kriengwattanakul, A., Kriengwattanakul, A., Wattanasarn, S., Suntharasantic, S., Chanpichaigosol, S., Kokaewwichian, C., Wattanavekin, T., Ovatlarnporn, P., Phayong, S., Tongoa, N., Changwichukarn, T., Leesatapornwongsa, T., Wannasuphoprasit, W. and Wongsaisuwan, M. (2008). Plasma-z 2008 team description paper. Tech. Rep., Chulalongkorn University,Phayathai Road, Bangkok, Thailand, 10330.

Laue, T., Burchardt, A., Fritsch, S., Hinz, S., Huhn, K., Kirilov, T., Martens, A., Miezal, M., Nehmiz, U., Schwarting, M. and Seekircher, A. (2009). B-smart extended team description for robocup 2009. Tech. Rep., Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Sichere Kognitive Systeme, Enrique-Schmidt-Str. 5, 28359 Bremen, Germany.

Li, X. and Zell, A. (2009). Motion control of an omnidirectional mobile robot. *Informatics in Control, Automation and Robotics*, pp. 181–193.

Maeno, J., Achiwa, H., Moribayashi, T., Tamaki, J., Hiramoto, N., Nakanishi, R., Narita, R., Otake, K., Tanaka, M., Ueno, S. *et al.* (2008). Robodragons 2008 team description. Tech. Rep., Aichi Prefectural University, Nagakute-cho, Aichi, 480-1198 JAPAN.

Martinez-Gomez, L., Moneo, F., Sotelo, D., Soto, M. and Weitzenfeld, A. (2005). Design and implementation of a small size robocup soccer team. In: *Proc. 2nd IEEE-RAS Latin American Robotics Symposium, Sao Luis, Brasil, Sept*, pp. 20–23.

Mohamed, N., Al-Jaroodi, J. and Jawhar, I. (2008). Middleware for robotics: a survey. In: *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pp. 736–742.

Mohamed, N., Al-Jaroodi, J. and Jawhar, I. (2009). A review of middleware for networked robots. *International Journal of Computer Science and Network Security*, vol. 9, no. 5, p. 139.

Montemerlo, M., Roy, N. and Thrun, S. (2004). Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit. In: *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 3, pp. 2436–2441. IEEE. ISBN 0780378601.

Nakajima, M., Kimura, T. and Masutani, Y. (2010). Odens 2010 team description. Tech. Rep., Osaka Electro-Communication University,1130-70, Kiyotaki, Shijonawate, Osaka 575-0063, Japan.

Namoshe, M., Tlale, N., Kumile, C. and Bright, G. (2008). Open middleware for robotics. In: *15th International Conference on Mechatronics and Machine Vision in Practice, Massey University, Auckland, New Zealand*, pp. 2–4. Massey University.

Neves, A., Azevedo, J., Cunha, M., Lau, N., Pereira, A., Corrente, G., Santos, F., Martins, D., Figueiredo, N., Silva, J. *et al.* (2010). Cambada 2010: Team description paper. Tech. Rep., Transverse Activity on Intelligent Robotics, IEETA/DETI University of Aveiro, 3810-193 Aveiro, Portugal.

Niemueller, T. (2009). *Developing A Behavior Engine for the Fawkes Robot-Control Software and its Adaptation to the Humanoid Platform Nao*. Ph.D. thesis, Masters thesis, Knowledge-Based Systems Group, RWTH Aachen University.

Nimbro (2010). Learning humanoid robots. Last visited: January 2011.
Available at: `http://www.nimbro.net/robots.html`

Phidgets (2010*a*). 1060 - phidgetmotorcontrol lv. Last visited: January 2011.
Available at: `http://www.phidgets.com`

Phidgets (2010*b*). 1064 - phidgetmotorcontrol hc. Last visited: January 2011.
Available at: `http://www.phidgets.com`

Pololu (2011*a*). Dual mc33887 motor driver carrier. Last visited: January 2011.
Available at: `http://www.pololu.com/catalog/product/712`

Pololu (2011*b*). Pololu jrk 12v12 usb motor controller with feedback. Last visited: January 2011.
Available at: `http://www.pololu.com/catalog/product/1393`

Pololu (2011*c*). Sharp gp2y0d805z0f digital distance sensor 5cm. Last visited: January 2011.
Available at: `http://www.pololu.com/catalog/product/1131`

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R. and Ng, A. (2009). Ros: an open-source robot operating system. In: *International Conference on Robotics and Automation*.

Roboard (2011). Roboard rb 100. Last visited: January 2011.
Available at: `http://www.roboard.com/RB-100.htm`

RoboCup (2009). Standard platform league. Last visited: January 2011.
Available at: `http://www.tzi.de/spl/pub/Website/WebHome/Nao.jpg`

RobotShop (2011*a*). Aluminum omni wheel. Last visited: January 2011.
Available at: `http://www.robotshop.ca/andymark-6in-alum-omni-wheel.html`

RobotShop (2011*b*). Parallax hmc6352 compass module. Last visited: January 2011.
Available at: `http://www.robotshop.com/ProductSearch.aspx?qs=HMC6352+compass`

RobotShop (2011*c*). Roboard mini-pci wifi card for rb-100. Last visited: January 2011.
Available at: `http://www.robotshop.com/roboard-wifi-card.html`

Rojas, R. and Förster, A. (2006). Holonomic control of a robot with an omnidirectional drive. *künstliche Intelligenz*, vol. 20, no. 2, pp. 12–17.

Shakhimardanov, A. and Prassler, E. (2007). Comparative evaluation of robotic software integration systems: A case study. In: *Intelligent Robots and Systems*. Citeseer.

Shopland (2010). dell computer xps. Last visited: January 2011.
Available at: `http://www.shopsland.org/wp-content/uploads/dell-computer-xps.jpg`

Sparkfun (2011*a*). Qik dual serial motor controller - high power. Last visited: January 2011.
Available at: `http://www.sparkfun.com/products/9210`

Sparkfun (2011*b*). Triple axis accelerometer breakout - adxl345. Last visited: January 2011.
Available at: `http://www.sparkfun.com/products/9156`

Tadokoro, S., Kitano, H., Takahashi, T., Noda, I., Matsubara, H., Shinjoh, A., Koto, T., Takeuchi, I., Takahashi, H., Matsuno, F., Hatayama, M., Nobe, J. and Shimada, S. (2002). The robocup-rescue project: A robotic approach to the disaster mitigation problem. In: *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 4, pp. 4089–4094. IEEE. ISBN 0780358864.

Utz, H., Sablatnog, S., Enderle, S. and Kraetzschmar, G. (2002). Miro-middleware for mobile robot applications. *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 4, pp. 493–497. ISSN 1042-296X.

Vaughan, B., Sukhatme, G., Howard, K. and Mataric, M. (2001). Most valuable player: A robot device server for distributed control. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, pp. 1226–1231.

Wasuntapichaikul, P., Srisabye, J., Onman, C., Damyot, S., Areeprasert, C. and Sukvichai, K. (2010). Skuba 2010 extended team description. Tech. Rep., Faculty of Engineering, Kasetsart University, 50 Phaholyothin Rd., Ladyao, Jatujak, Bangkok, 10900, Thailand.

Watanabe, K. (1998). Control of an omnidirectional mobile robot. In: *1998 Second International Conference on Knowledge-Based Intelligent Electronic Systems, 1998. Proceedings KES'98*, pp. 51–60.

Weitzenfeld, A., Martínez-Gómez, L., Francois, J., Levin-Pick, A., Obraczka, K. and Boice, J. (2007). Multi-robot systems: Extending robocup small-size architecture with local vision and ad-hoc networking. In: *Robotics Symposium, 2006. LARS'06. IEEE 3rd Latin American*, pp. 26–33. Citeseer. ISBN 1424405378.

Wikipedia (2011 January). Robocup 3d soccer simulation league. Last visited: January 2011.
Available at: `http://en.wikipedia.org/wiki/RoboCup3DSoccerSimulationLeague`

Williams, R., Carter, B., Gallina, P. and Rosati, G. (2002). Dynamic model with slip for wheeled omnidirectional robots. *IEEE transactions on Robotics and Automation*, vol. 18, no. 3, pp. 285–293.

Wu, Y., Qiu, X., Yu, G., Chen, J., Rie, X. and Xiong, R. (2009). Extended team description paper of zjunlict 2009. Tech. Rep., Technical report, National Laboratory of Industrial Control Technology Zhejiang University, 2009.

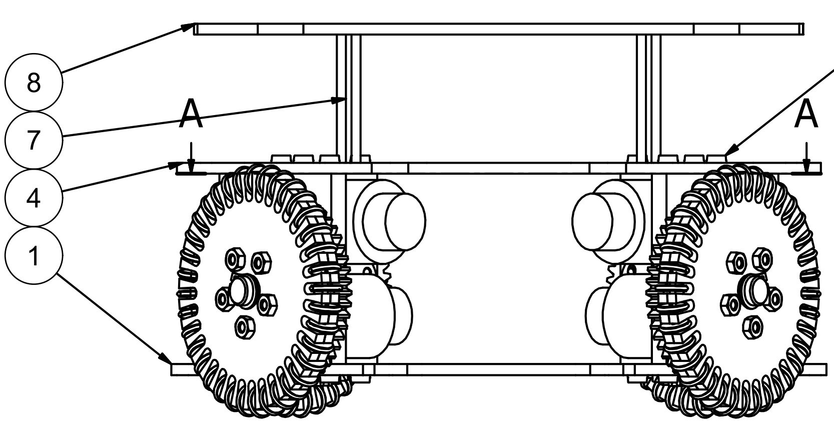
Zickler, S., Biswas, J., Bruce, J., Licitra, M. and Veloso, M. (2010). Cmdragons 2010 extended team description. Tech. Rep., Carnegie Mellon University.

Zickler, S., Vail, D., Levi, G., Wasserman, P., Bruce, J., Licitra, M. and Veloso, M. (2008). Cmdragons 2008 team description. *Proceedings of RoboCup*, vol. 1.

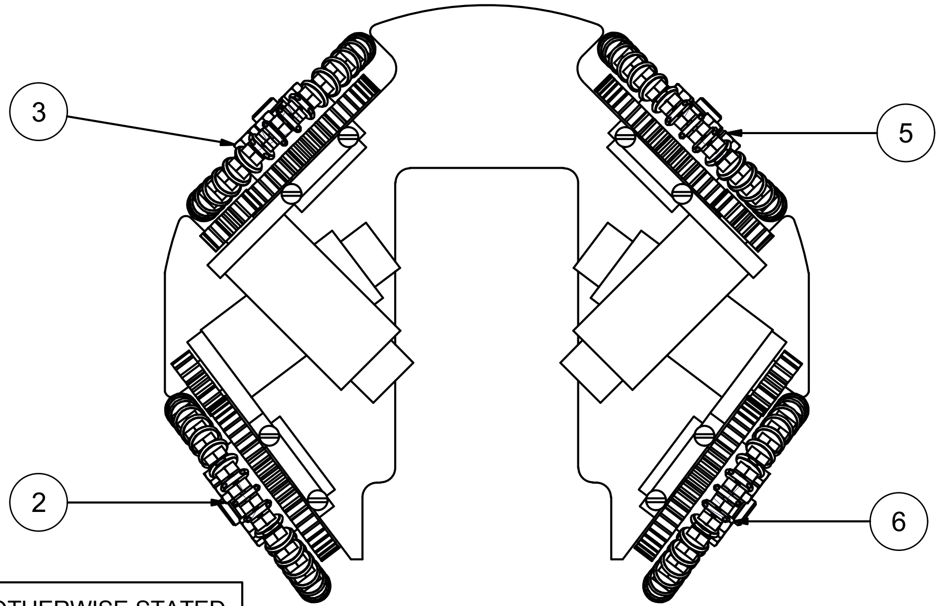
Appendices

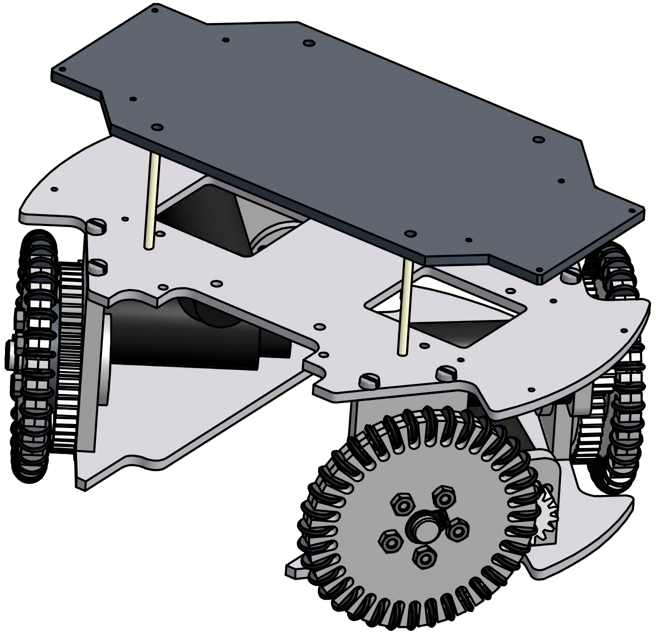
Mechanical Design Drawings

This appendix contains the manufacturing drawings for the mechanical system of the robot.



A-A (1 : 1.5)





ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION
9	Slotted cheese head machine screws	16	BS 4183 - M3 x 8
8	Battery Mount	1	00/00/03
7	Battery Mount Spacer	4	M3 x 40
6	Front Right Drive Assembly	1	04/00/00
5	Rear Right Drive Assembly	1	01/00/00
4	Top Plate	1	00/00/02
3	Rear Left Drive Assembly	1	02/00/00
2	Front Left Drive Assembly	1	03/00/00
1	Base Plate	1	00/00/01

UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

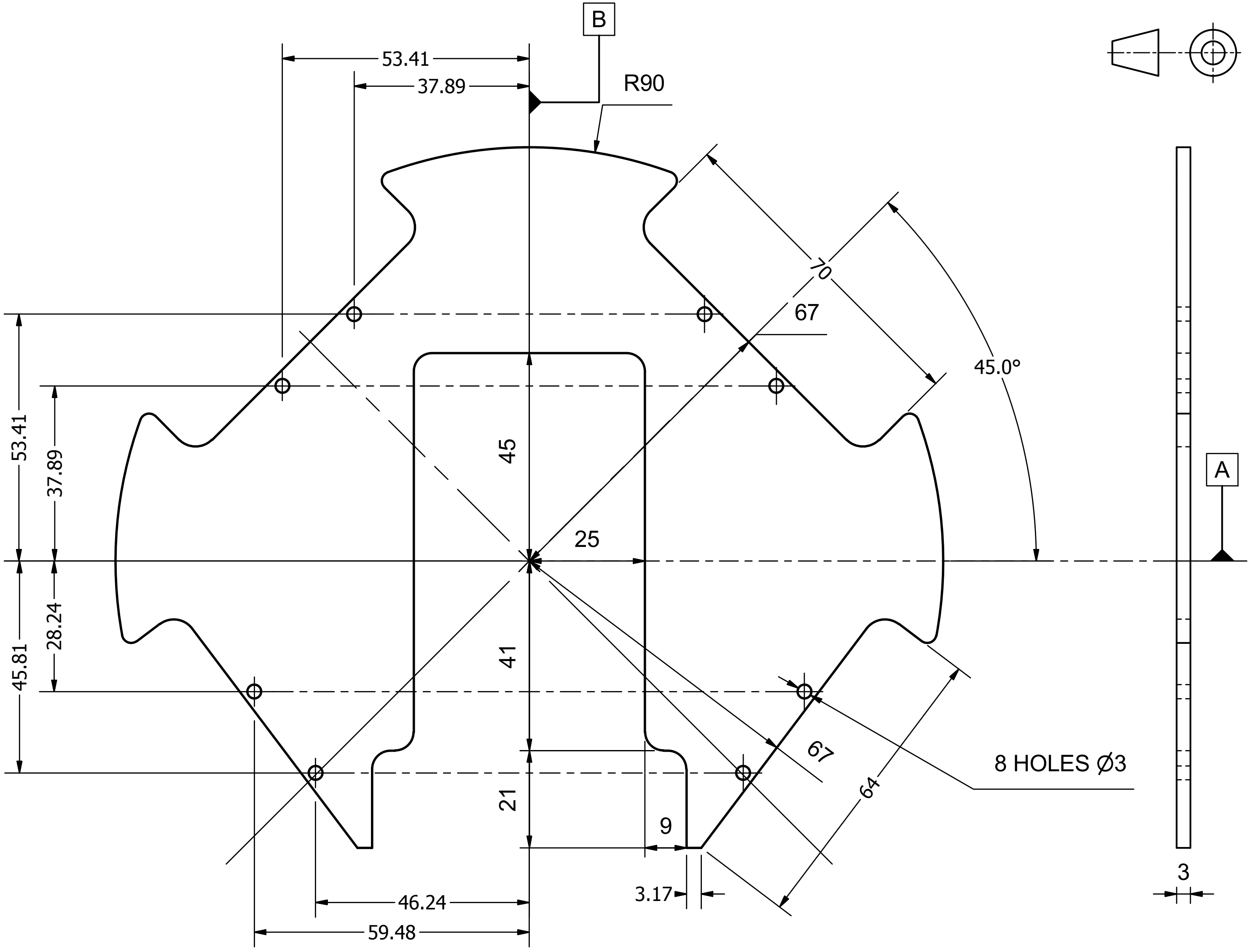
STELLENBOSCH UNIVERSITY

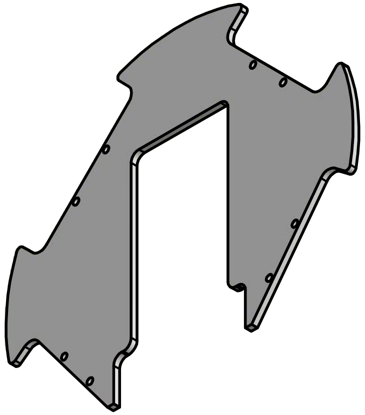
SCALE ON A3 1:1.5 **MEASUREMENTS IN mm**	**TITEL: Robot Assembly**		
STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	
DATE: 25/01/11	**SHEET No. 1 OF 1 SHEETS**	**No. 00/00/00**	

NOTE: THIS PART NEEDS TO BE LASER CUT. DIMENSIONS ONLY FOR ADDITIONAL REFERENCE

NOTE: ALL HOLES TO BE DRILLED WITH RESPECT TO DATUM A AND B

NOTE: HOLES ARE SYMMETRICAL AROUND VERTICAL LINE (DATUM B)





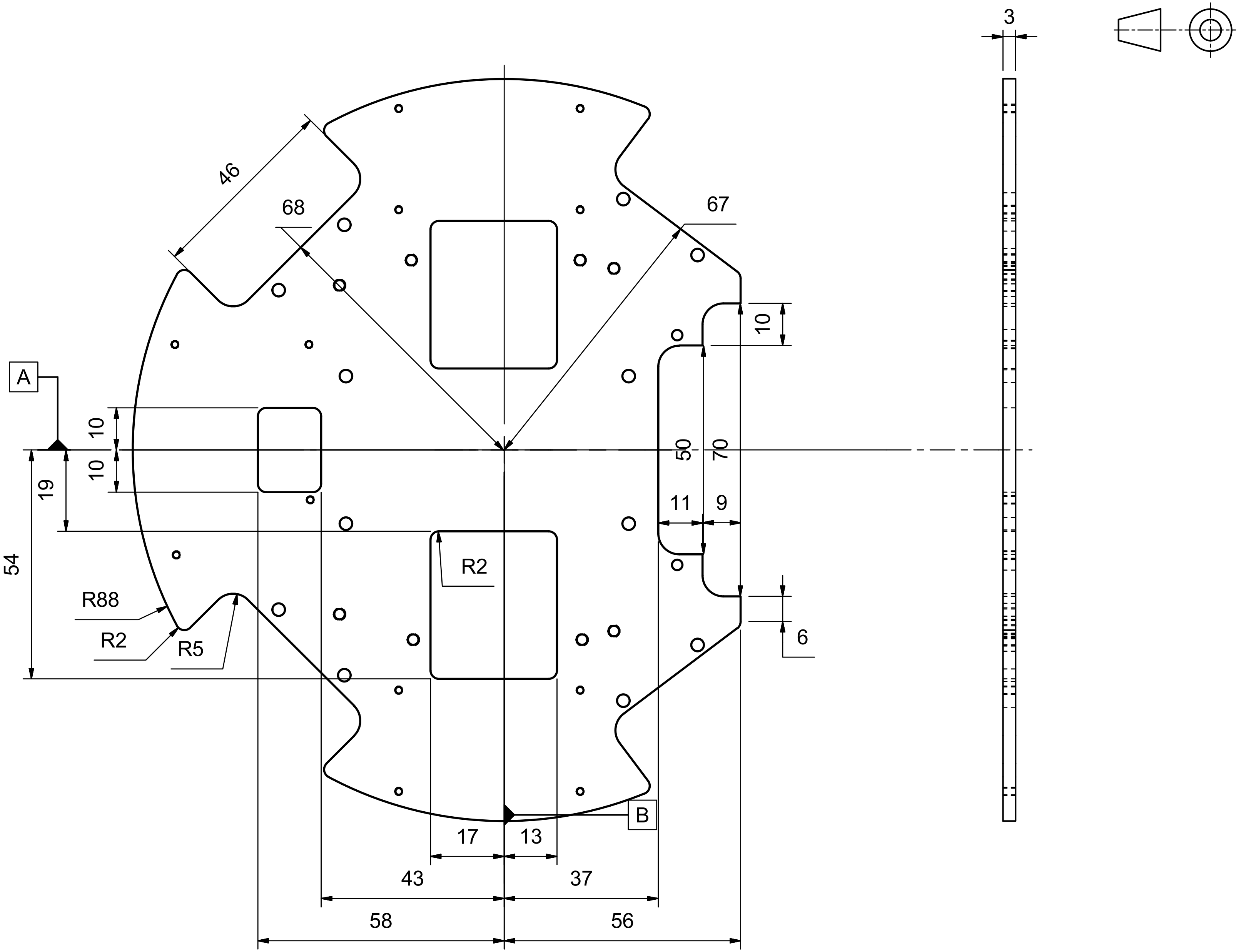
UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1°

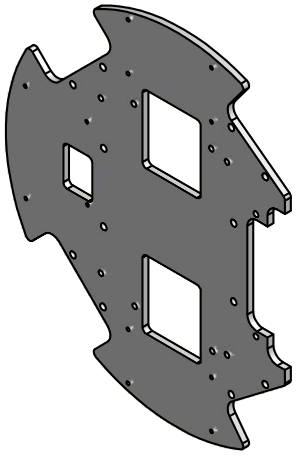
1	Base Plate	1	Aluminium
ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION

STELLENBOSCH UNIVERSITY		**SCALE ON A3 1:1** **MEASUREMENTS IN mm**	**TITEL: Base Plate**
STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	**DATE: 25/01/11**
SHEET No. 1 OF 1 SHEETS	**No. 00/00/01**		

NOTE: THIS PART NEEDS
TO BE LASER CUT.
DIMENSIONS ONLY FOR
ADDITIONAL REFERENCE

NOTE: ALL DIMENSIONS
ARE SYMMETRICAL
AROUND HORIZONTAL
LINE (DATUM A)





UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

1	Top Plate	1	Aluminium
ITEM	**DESCRIPTION**	**QTY**	**MATERIAL / SPECIFICATION**

STELLENBOSCH UNIVERSITY

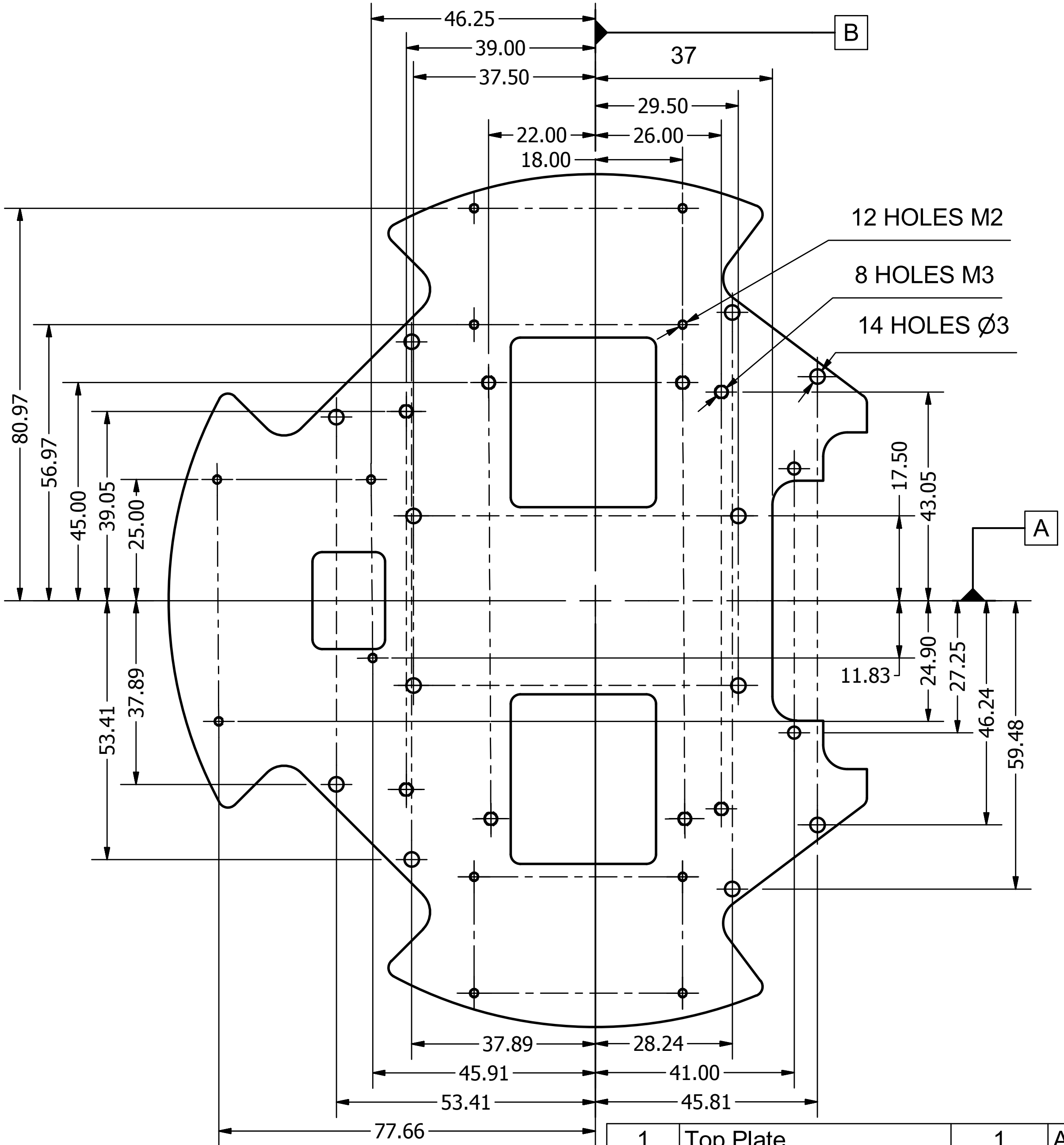
SCALE ON A3 1:1
MEASUREMENTS IN mm

TITEL: Top Plate

STUDENT No. 15979199 | **DRAWN BY: A SMIT** | **REVIEWED:** | **DATE: 25/01/11** | **SHEET No. 1 OF 2 SHEETS** | **No. 00/00/02**

NOTE: ALL HOLES ARE TO BE DRILLED WITH RESPECT TO DATUM A AND B

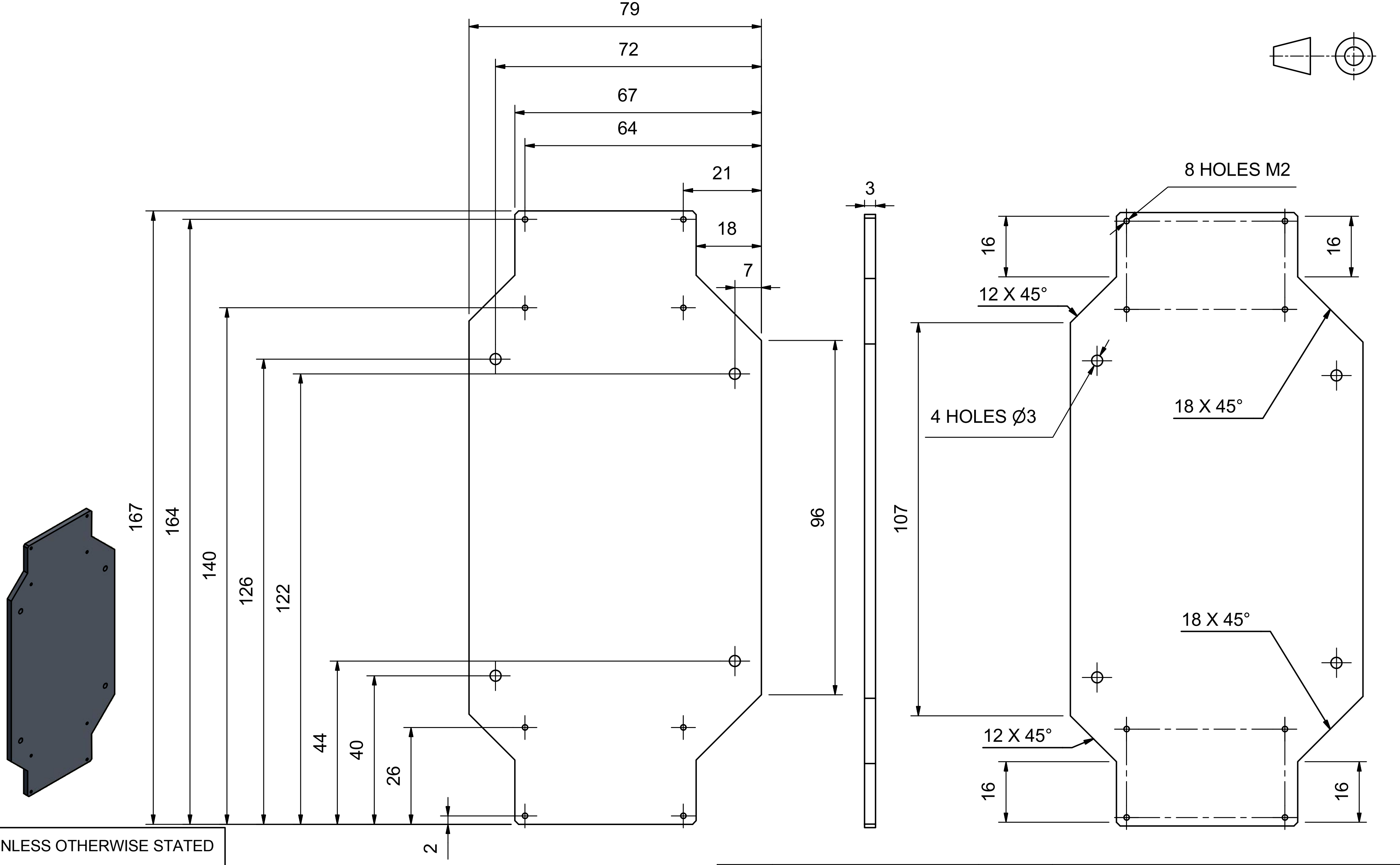
NOTE: ALL DIMENSIONS ARE SYMMETRICAL AROUND HORIZONTAL LINE (DATUM A)



1	Top Plate	1	Aluminium
ITEM	**DESCRIPTION**	**QTY**	**MATERIAL / SPECIFICATION**

STELLENBOSCH UNIVERSITY

SCALE ON A3 1:1 **MEASUREMENTS IN mm**	**TITEL: Top Plate**		
STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	
DATE: 25/01/11	**SHEET No. 2 OF 2 SHEETS**	**No. 00/00/02**	



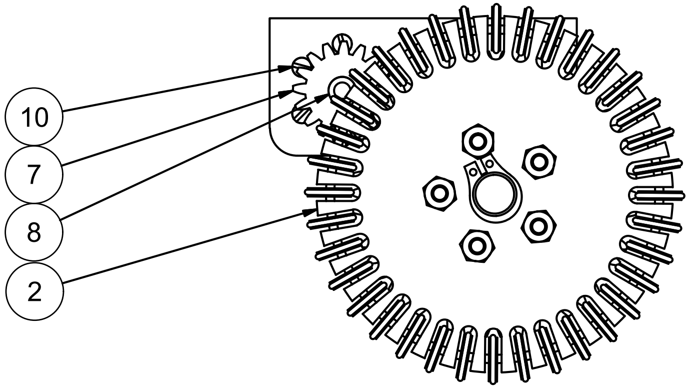
UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

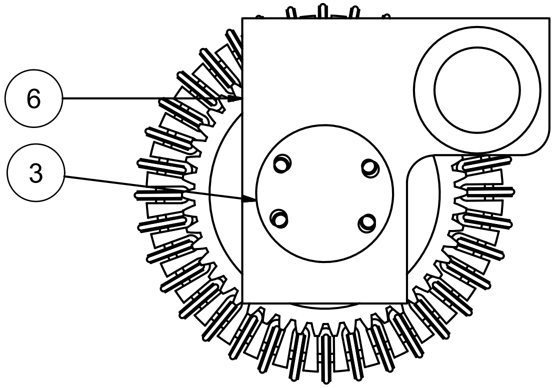
1	Battery Mount	1	PVC
ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION

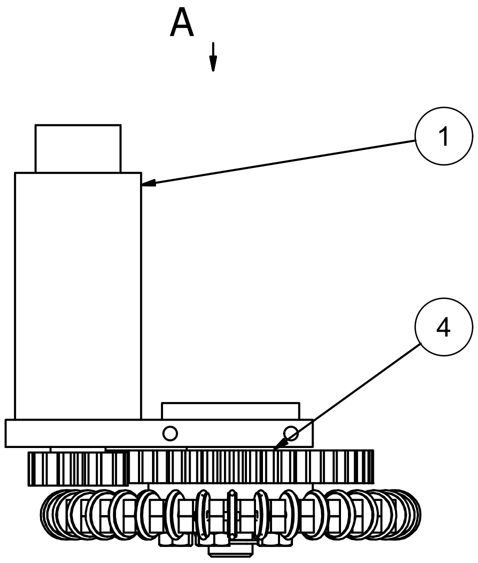
STELLENBOSCH UNIVERSITY

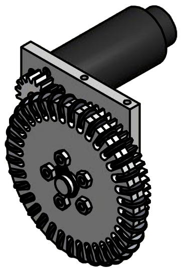
SCALE ON A3 1:1 MEASUREMENTS IN mm	TITEL: Battery Mount		
STUDENT No. 15979199	DRAWN BY: A SMIT	REVIEWED:	
DATE: 25/01/11	SHEET No. 1 OF 1 SHEETS	No. 00/00/03	

A (1 : 1)





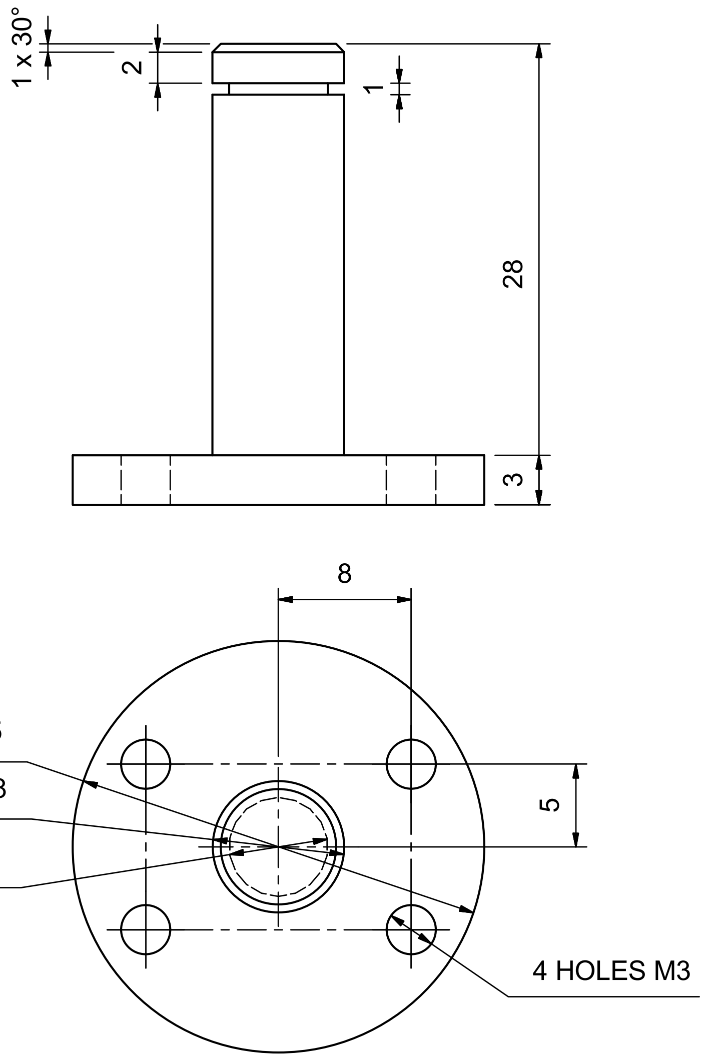


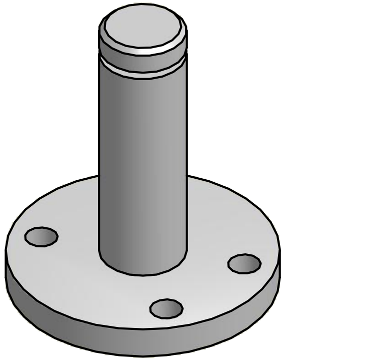


10	ISO metric machine screws	6	AS 1427 - M2 x 8
9	ISO metric machine screws	4	AS 1427 - M3 x 8
8	Bush	1	01/00/04
7	Motor Spur Gear	1	01/00/03
6	Rear Right Drive Mount	1	01/00/02
5	Circlip	1	ANSI B 27.7M - 3AMI-8
4	Washer	1	DIN 125 - A 8.4
3	Wheel Shaft	1	01/00/01
2	Omni Wheel Assembly	1	01/01/00
1	Motor	1	FAULHABER CATALOGUE 2342S012CR
ITEM	**DESCRIPTION**	**QTY**	**MATERIAL / SPECIFICATION**

UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

STELLENBOSCH UNIVERSITY			**SCALE ON A3 1:1** **MEASUREMENTS IN mm**	**TITEL: Rear Right Drive Assembly**
STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	**DATE: 25/01/11**	**SHEET No. 1 OF 1 SHEETS** · **No. 01/00/00**





UNLESS OTHERWISE STATED

TOLERANCES ± 0,1

ANGLES 1˚

1	Wheel Shaft	4	Aluminium
ITEM	**DESCRIPTION**	**QTY**	**MATERIAL / SPECIFICATION**

STELLENBOSCH UNIVERSITY

SCALE ON A3 3:1

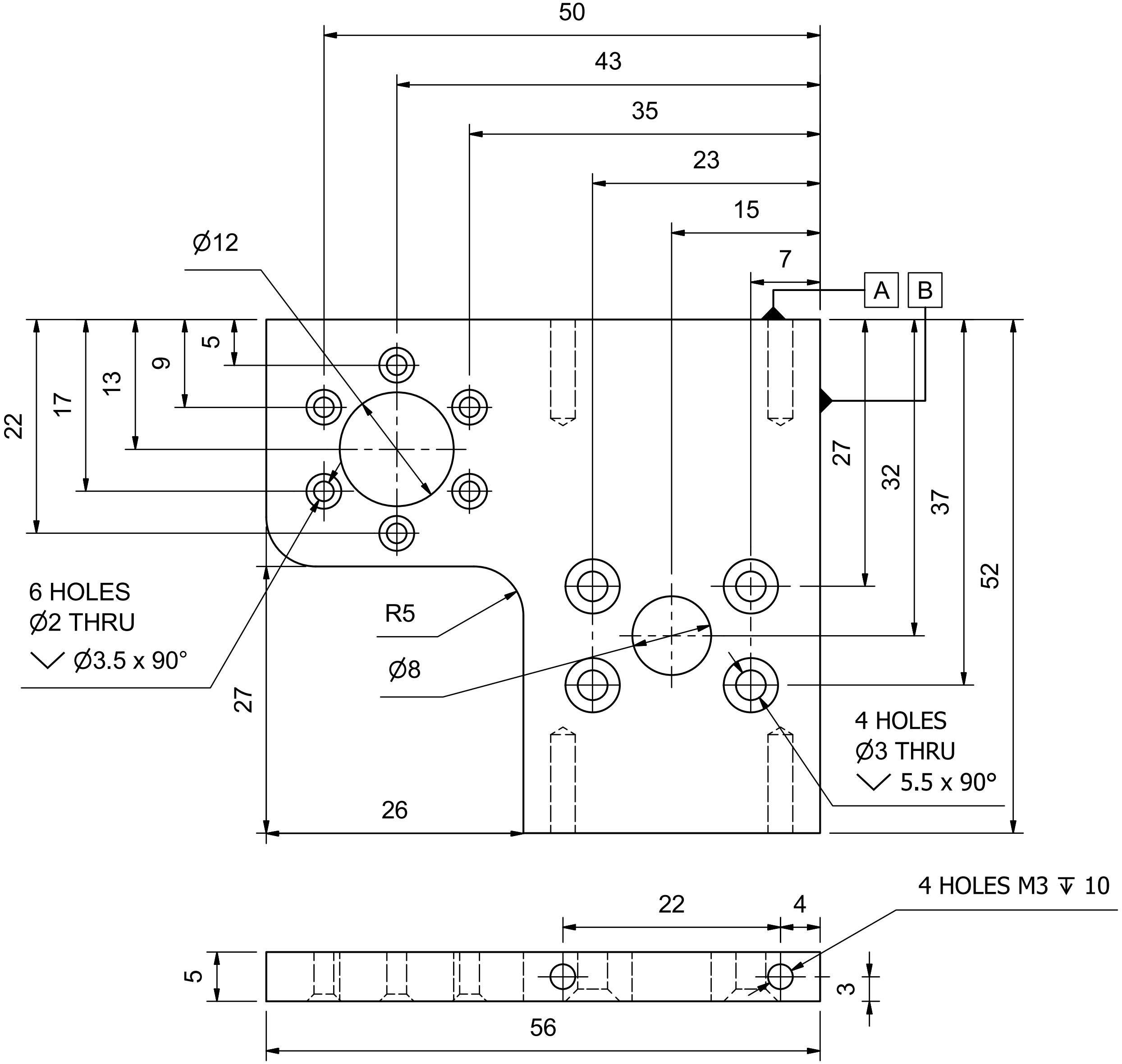
MEASUREMENTS IN mm

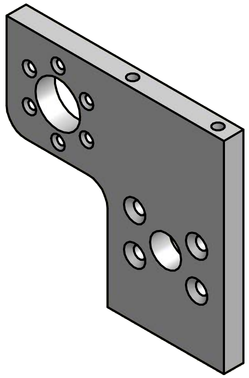
TITEL: Wheel Shaft

STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	**DATE: 25/01/11**	**SHEET No. 1 OF 1 SHEETS**	**No. 01/00/01**

NOTE: THIS PART NEEDS
TO BE LASER CUT.
DIMENSIONS ONLY FOR
ADDITIONAL REFERENCE

NOTE: HOLES TO BE
BORED WITH RESPECT
TO DATUM A and B





UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

1	Rear Right Drive Mount	1	Aluminium
ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION

STELLENBOSCH UNIVERSITY

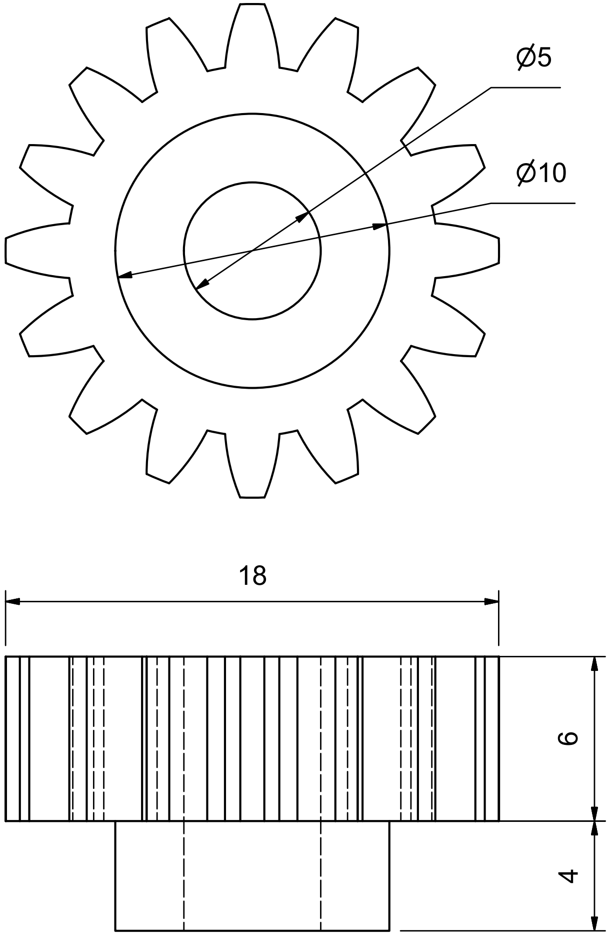
SCALE ON A3 2:1
MEASUREMENTS IN mm

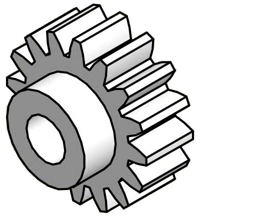
TITEL: Rear Right Drive Mount

STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	**DATE: 25/01/11**

SHEET No. 1 OF 1 SHEETS **No. 01/00/02**

NOTE: THIS PART NEEDS TO
BE MACHINED FROM THE
ORIGINAL PART ACCORDING
TO DIMESIONS

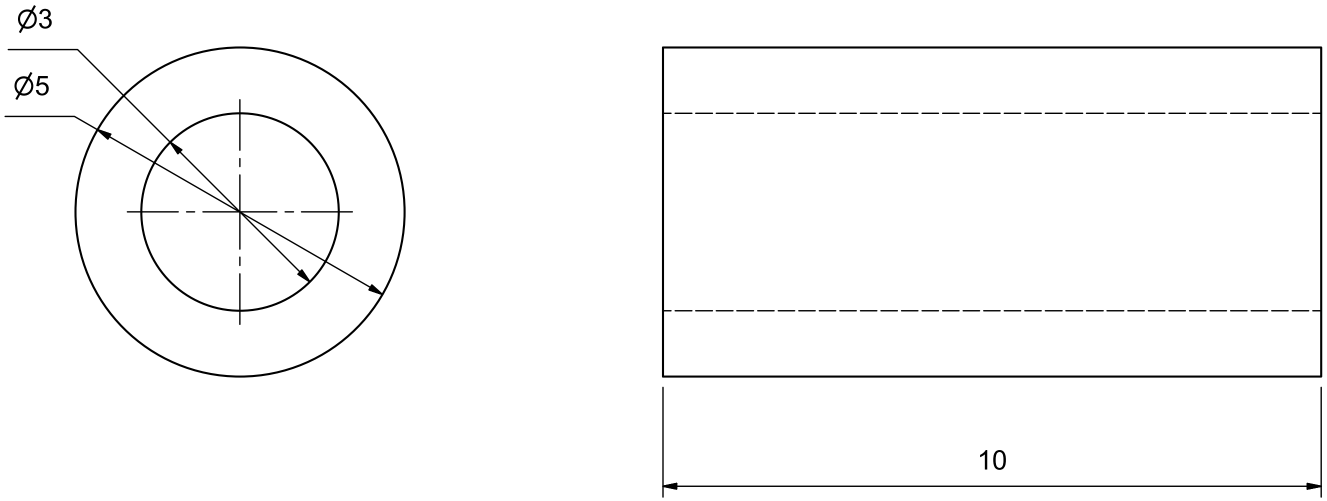


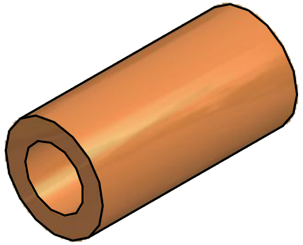


UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION
1	Motor Spur Gear	4	DELRIN 1 MODULE 16 TEETH

STELLENBOSCH UNIVERSITY		**SCALE ON A3 5:1** **MEASUREMENTS IN mm**	**TITEL: Motor Spur Gear**
STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	**DATE: 25/01/11**
SHEET No. 1 OF 1 SHEETS	**No. 01/00/03**		



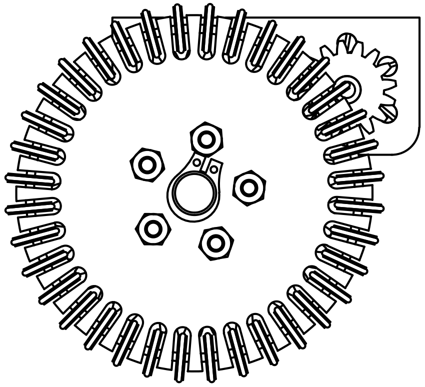


UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

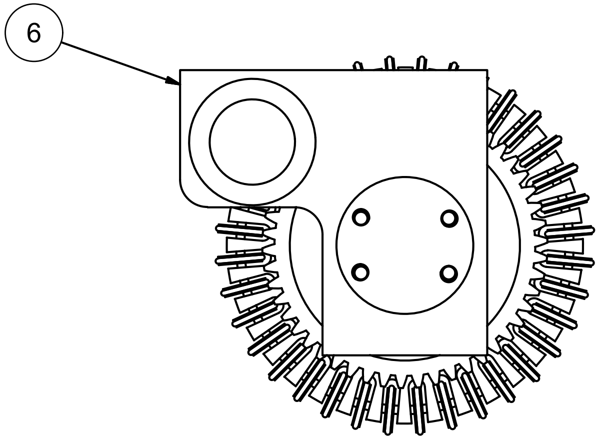
1	Bush	4	Copper
ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION

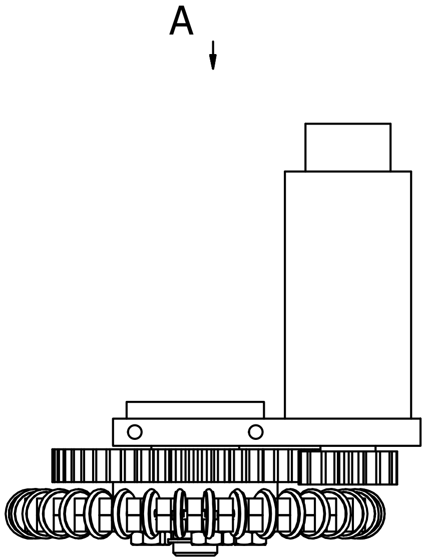
STELLENBOSCH UNIVERSITY	SCALE ON A3 12:1 MEASUREMENTS IN mm	TITEL: Bush

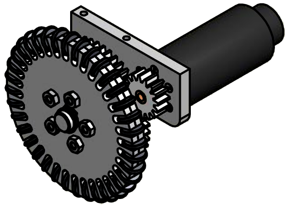
STUDENT No. 15979199	DRAWN BY: A SMIT	REVIEWED:	DATE: 25/01/11	SHEET No. 1 OF 1 SHEETS	No. 01/00/04



A (1 : 1)







10	ISO metric machine screws	6	AS 1427 - M2 x 8
9	ISO metric machine screws	4	AS 1427 - M3 x 8
8	Bush	1	02/00/04
7	Motor Spur Gear	1	02/00/03
6	Rear Left Drive Mount	1	02/00/02
5	Circlip	1	ANSI B 27.7M - 3AMI-8
4	Washer	1	DIN 125 - A 8.4
3	Wheel Shaft	1	02/00/01
2	Omni Wheel Assembly	1	02/01/00
1	Motor	1	FAULHABER CATALOGUE 2342S012CR
ITEM	**DESCRIPTION**	**QTY**	**MATERIAL / SPECIFICATION**

UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

STELLENBOSCH UNIVERSITY

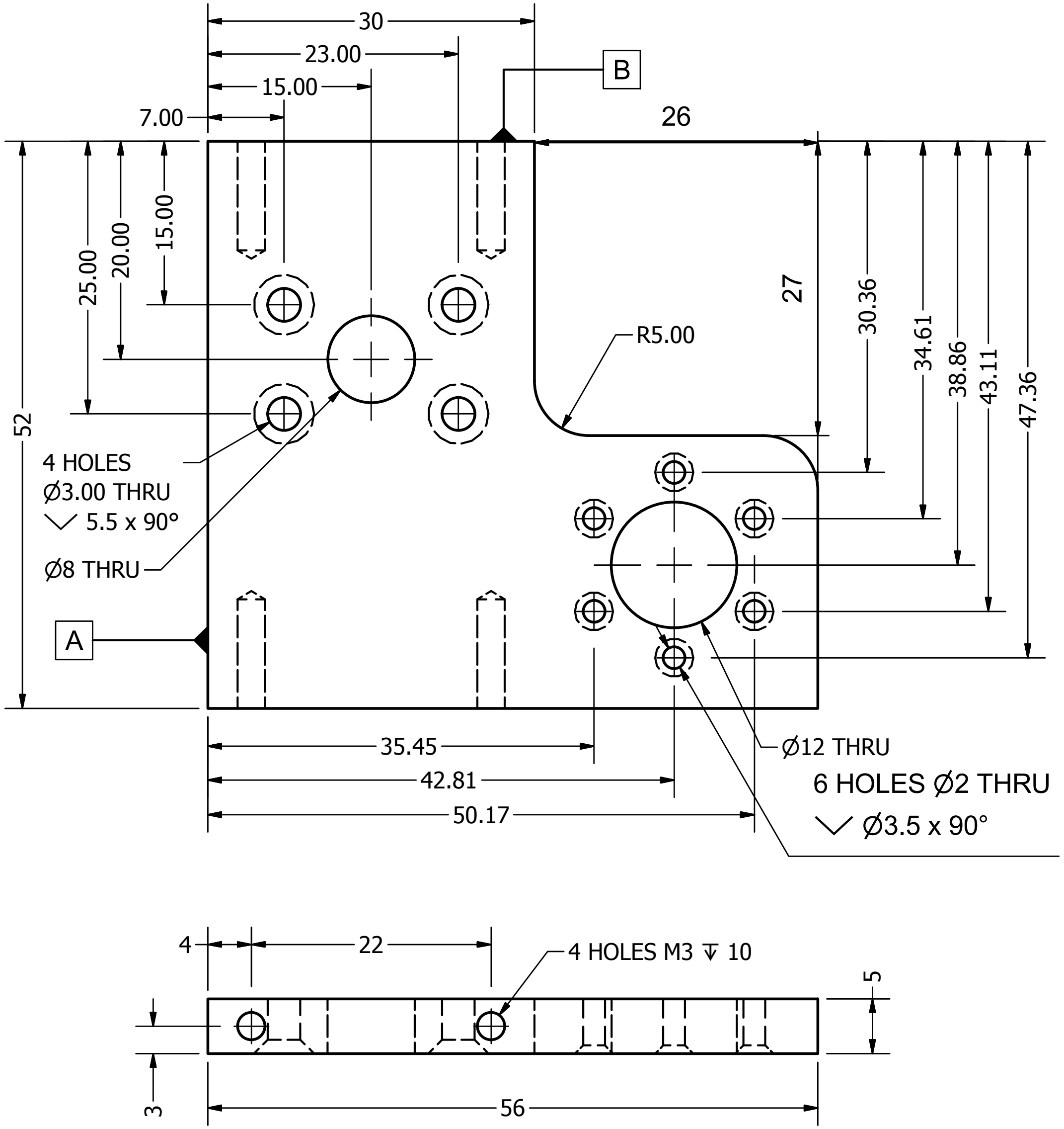
SCALE ON A3 1:1
MEASUREMENTS IN mm

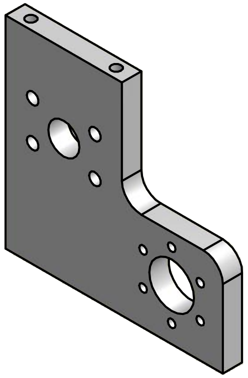
TITEL: Rear Left Drive Assembly

STUDENT No. 15979199 | **DRAWN BY: A SMIT** | **REVIEWED:** | **DATE: 25/01/11** | **SHEET No. 1 OF 1 SHEETS** | **No. 02/00/00**

NOTE: THIS PART
NEEDS TO BE LASER
CUT. DIMENSIONS
ONLY FOR ADDITIONAL
REFERENCE

NOTE: HOLES
TO BE BORED
WITH RESPECT
TO DATUM A and
B



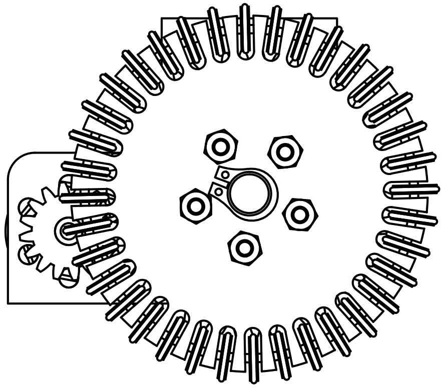


UNLESS OTHERWISE STATED	
TOLERANCES ± 0,1	
ANGLES 1˚	

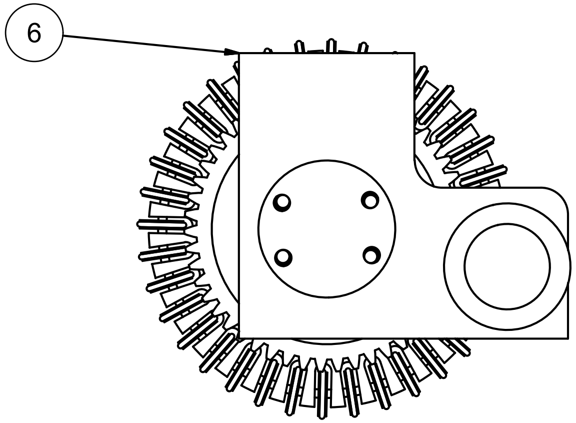
1	Rear Left Drive Mount	1	Aluminium
ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION

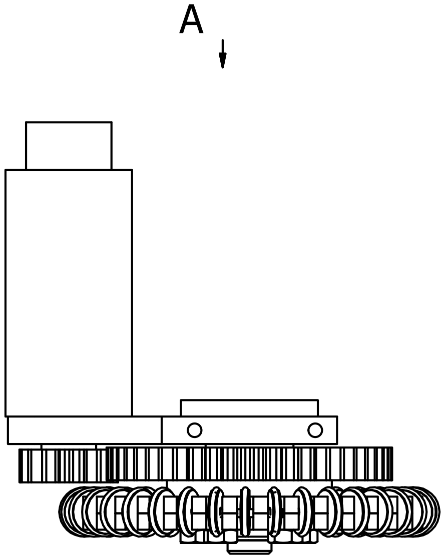
STELLENBOSCH UNIVERSITY

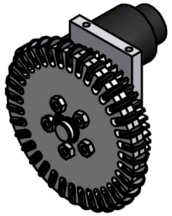
		SCALE ON A3 2:1 MEASUREMENTS IN mm	TITEL: Rear Left Drive Mount	
STUDENT No. 15979199	DRAWN BY: A SMIT	REVIEWED:	DATE: 25/01/11	SHEET No. 1 OF 1 SHEETS	No. 02/00/02



A (1 : 1)







ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION
10	ISO metric machine screws	6	AS 1427 - M2 x 8
9	ISO metric machine screws	4	AS 1427 - M3 x 8
8	Bush	1	03/00/04
7	Motor Spur Gear	1	03/00/03
6	Front Left Drive Mount	1	03/00/02
5	Circlip	1	ANSI B 27.7M - 3AMI-8
4	Washer	1	DIN 125 - A 8.4
3	Wheel Shaft	1	03/00/01
2	Omni Wheel Assembly	1	03/01/00
1	Motor	1	FAULHABER CATALOGUE 2342S012CR

UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

STELLENBOSCH UNIVERSITY

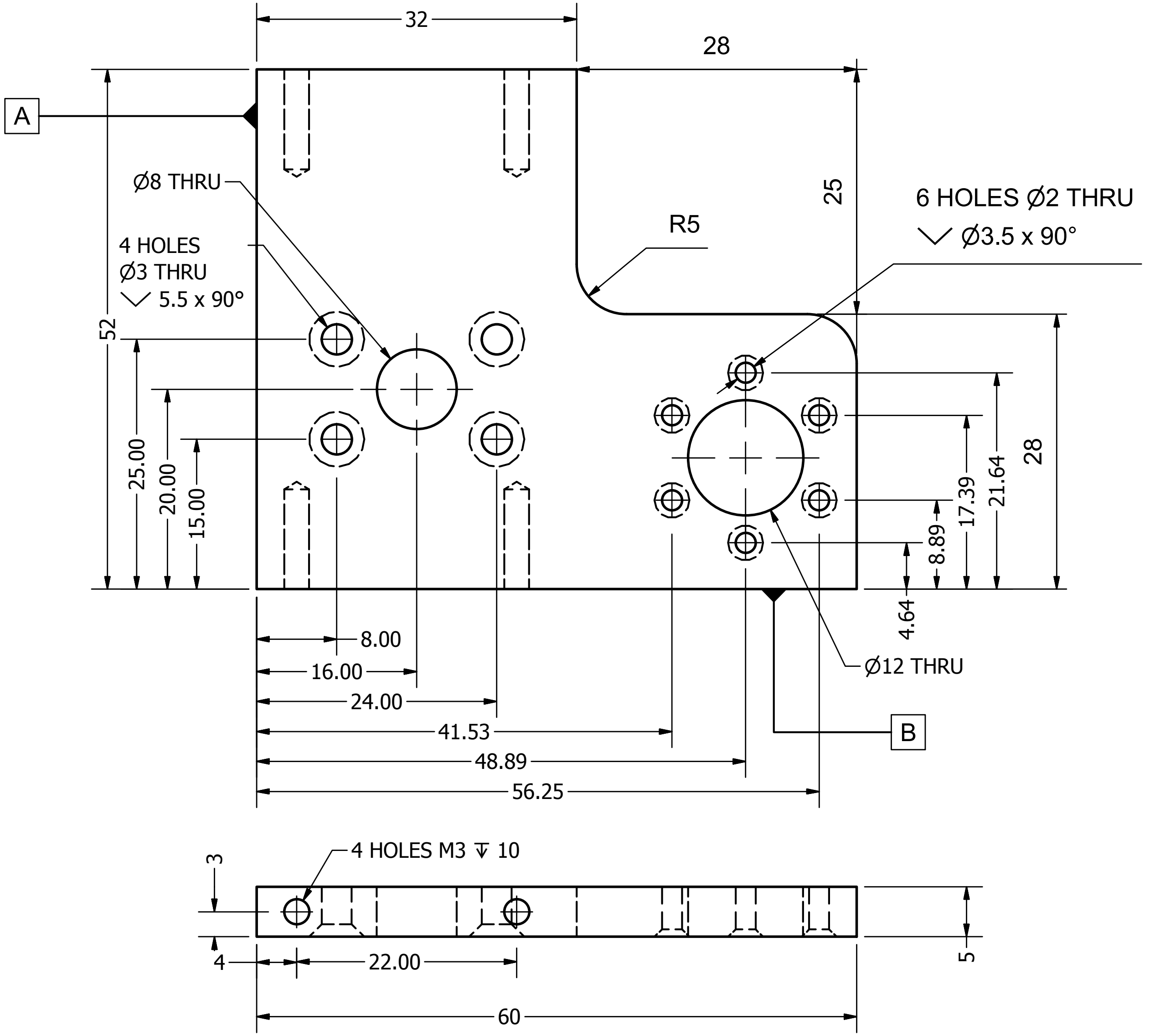
SCALE ON A3 1:1
MEASUREMENTS IN mm

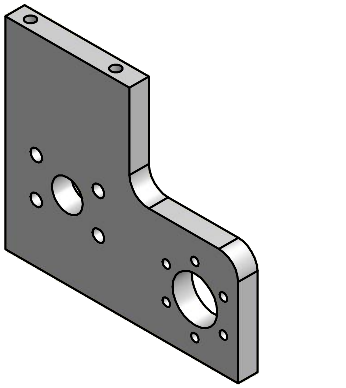
TITEL: Front Left Drive Assembly

STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	**DATE: 25/01/11**	**SHEET No. 1 OF 1 SHEETS**	**No. 03/00/00**

NOTE: THIS PART NEEDS TO BE LASER CUT. DIMENSIONS ONLY FOR ADDITIONAL REFERENCE

NOTE: HOLES TO BE DRILLED WITH RESPECT TO DATUM A AND B





UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

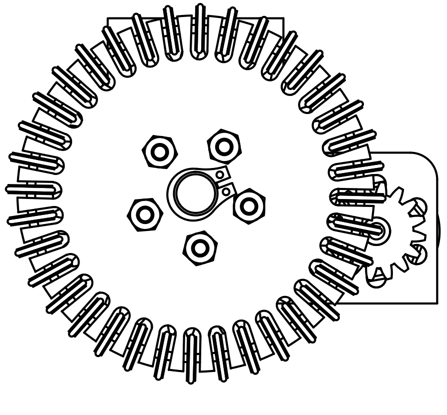
1	Front Left Drive Mount	1	Aluminium
ITEM	**DESCRIPTION**	**QTY**	**MATERIAL / SPECIFICATION**

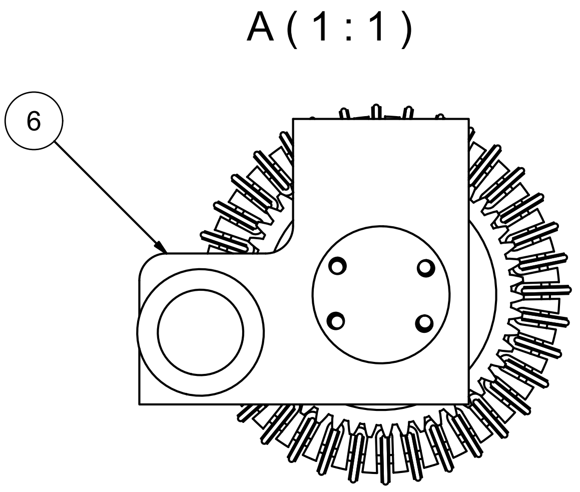
STELLENBOSCH UNIVERSITY

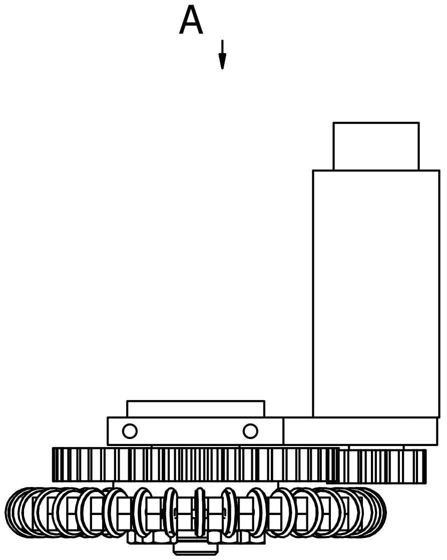
SCALE ON A3 2:1
MEASUREMENTS IN mm

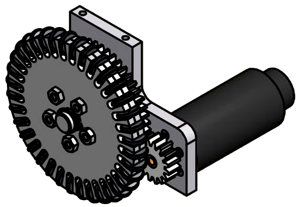
TITEL: Front Left Drive Mount

STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	**DATE: 25/01/11**	**SHEET No. 1 OF 1 SHEETS**	**No. 03/00/02**









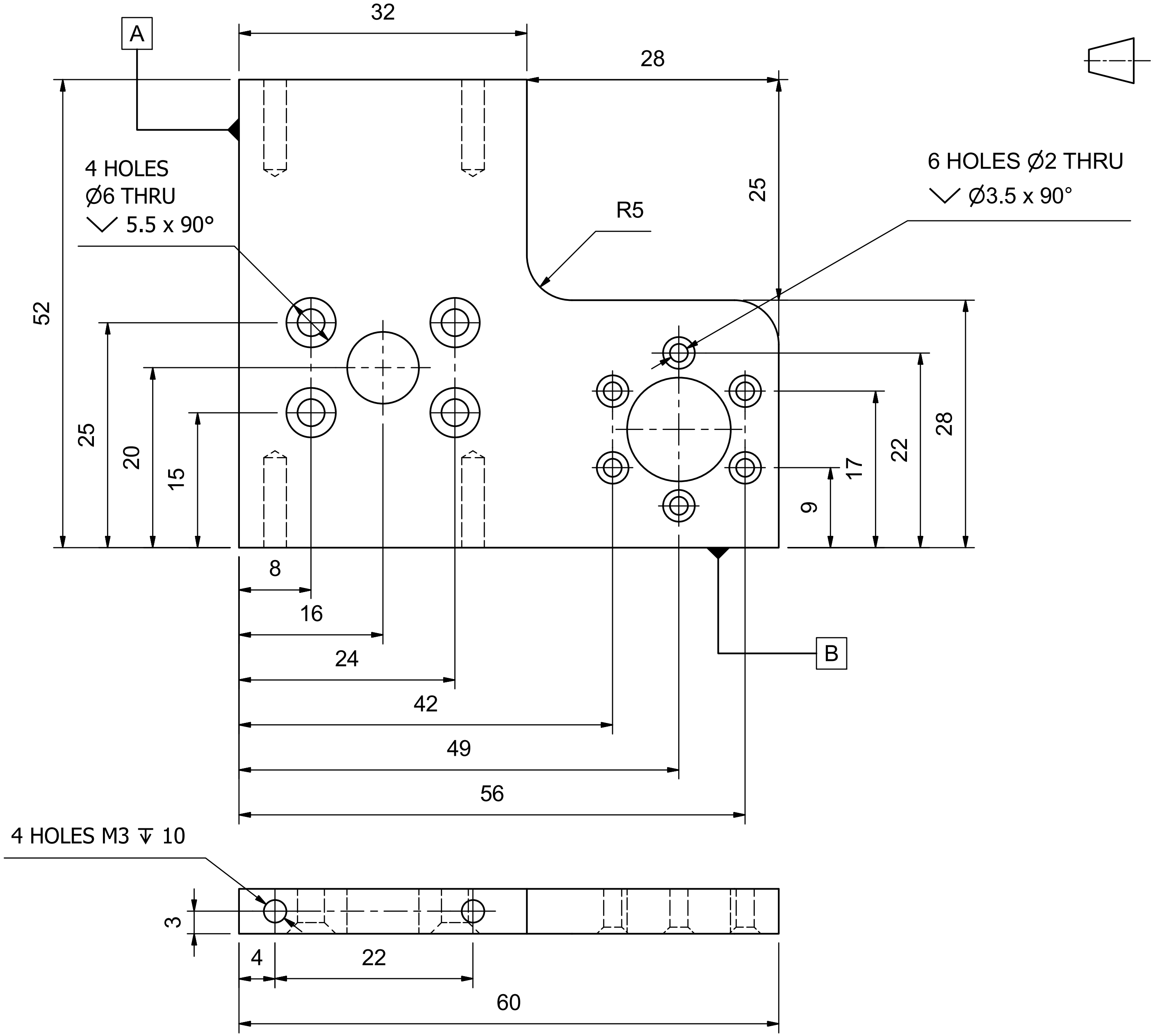
ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION
10	ISO metric machine screws	6	AS 1427 - M2 x 8
9	ISO metric machine screws	4	AS 1427 - M3 x 8
8	Bush	1	04/00/04
7	Motor Spur Gear	1	04/00/03
6	Front Right Drive Mount	1	04/00/02
5	Circlip	1	ANSI B 27.7M - 3AMI-8
4	Washer	1	DIN 125 - A 8.4
3	Wheel Shaft	1	04/00/01
2	Omni Wheel Assembly	1	04/01/00
1	Motor	1	FAULHABER CATALOGUE 2342S012CR

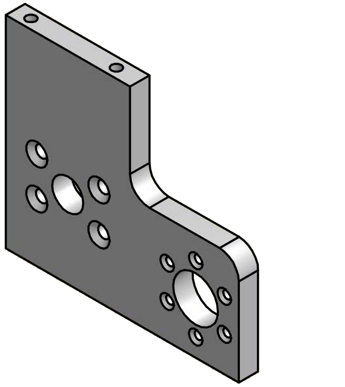
UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

STELLENBOSCH UNIVERSITY		**SCALE ON A3 1:1** **MEASUREMENTS IN mm**	**TITEL: Front Right Drive Assembly**
STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	**DATE: 25/01/11**
SHEET No. 1 OF 1 SHEETS	**No. 04/00/00**		

NOTE: THIS PART
NEEDS TO BE LASER
CUT. DIMENSIONS ONLY
FOR ADDITIONAL
REFERENCE

NOTE: HOLES TO
BE DRILLED WITH
RESPECT TO
DATUM A AND B





UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

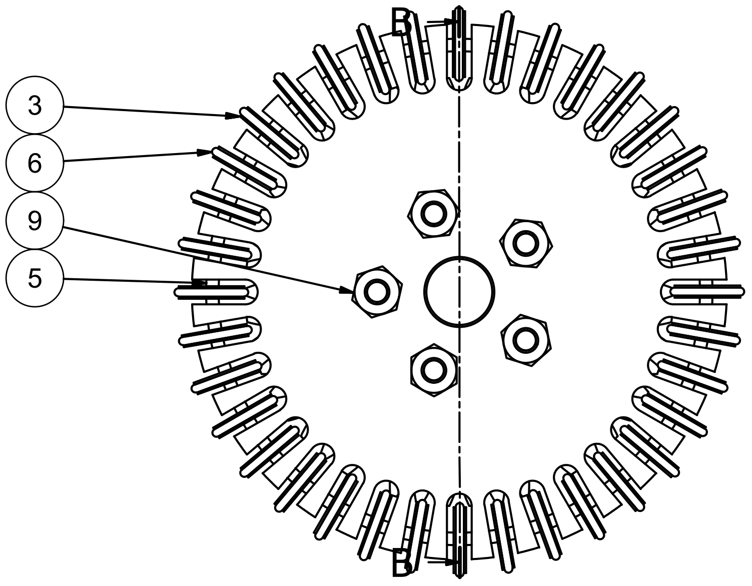
1	Front Right Drive Mount	1	Aluminium
ITEM	**DESCRIPTION**	**QTY**	**MATERIAL / SPECIFICATION**

STELLENBOSCH UNIVERSITY

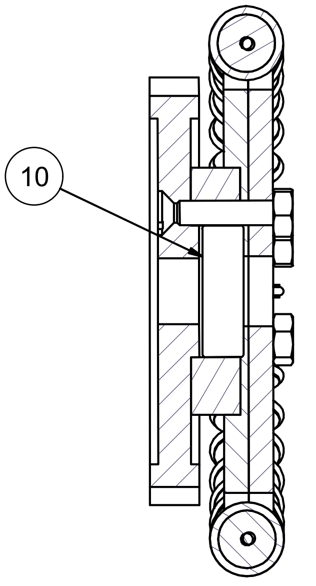
SCALE ON A3 2:1
MEASUREMENTS IN mm

TITEL: Front Right Drive Mount

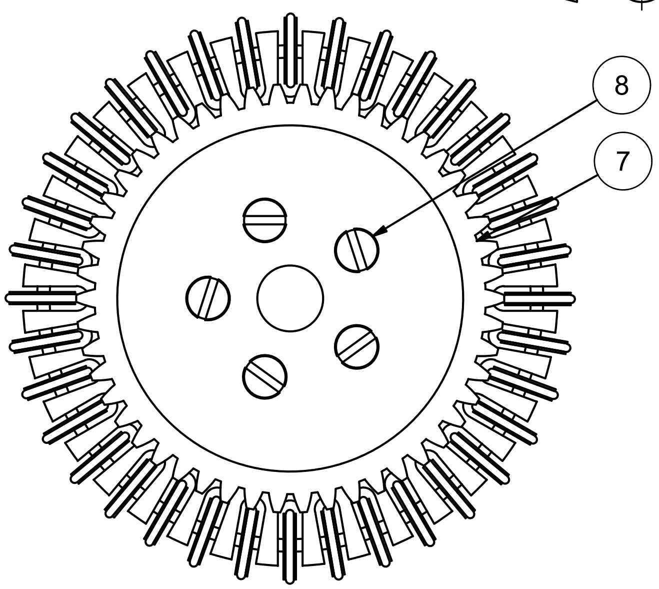
STUDENT No. 15979199 | **DRAWN BY: A SMIT** | **REVIEWED:** | **DATE: 25/01/11** | **SHEET No. 1 OF 1 SHEETS** | **No. 004/00/02**

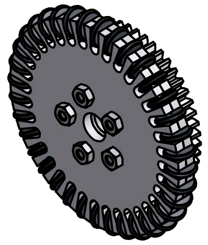


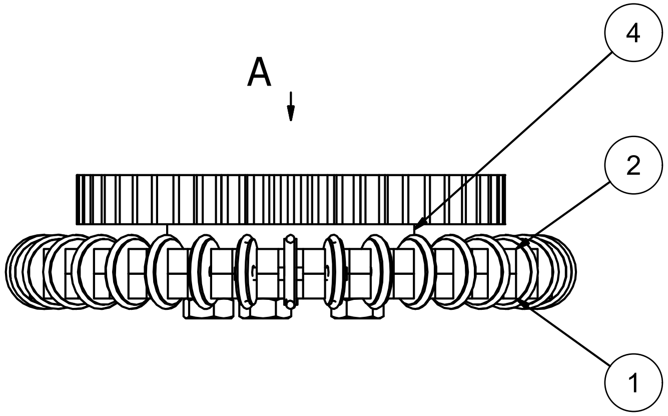
B-B (1.5 : 1)



A (1.5 : 1)





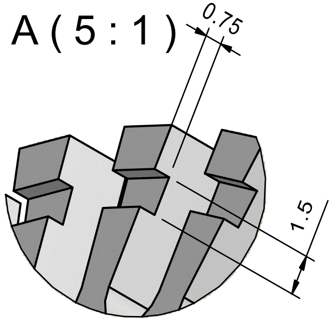


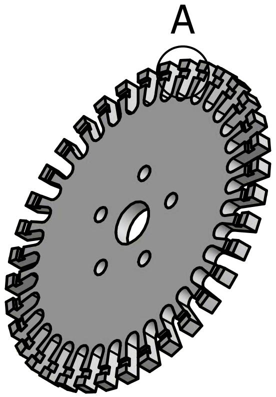
ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION
10	Single-row radial ball bearings	1	2080088 GOST 10058-90 (8 X 16 X 5)
9	ISO metric hexagon nuts	5	AS 1112 - M3 Type 5
8	ISO metric machine screws	5	AS 1427 - M3 x 16
7	Wheel Spur Gear	1	01/01/05
6	O-ring	36	Rubber ID 7 x 1
5	Wire ring	1	GALVANISED STEEL BINDING WIRE 1.25 (Length 188)
4	Hub Extension/Spacer	1	01/01/04
3	Omni Wheel Roller	36	01/01/03
2	Bottom Wheel Plate	1	01/01/02
1	Top Wheel Plate	1	01/01/01

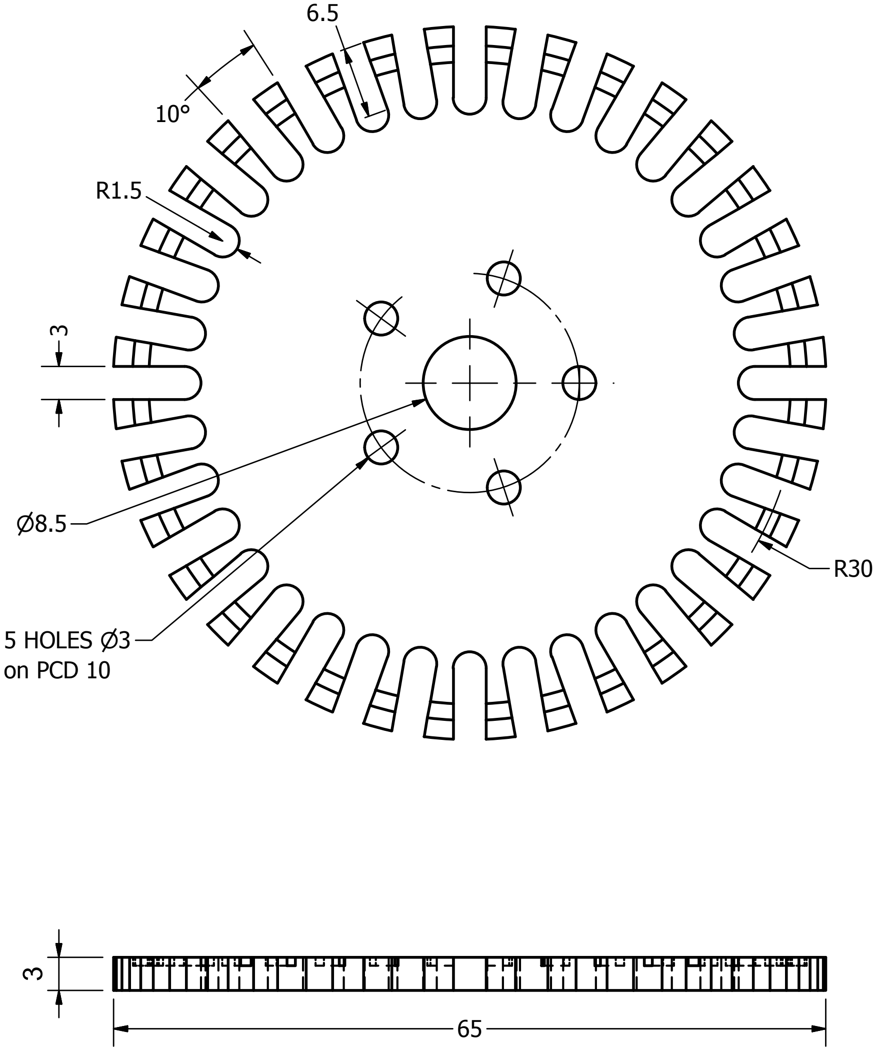
UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

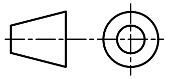
STELLENBOSCH UNIVERSITY		**SCALE ON A3 1:1.5** **MEASUREMENTS IN mm**	**TITEL: Omni Wheel Assembly**
STUDENT No. 15979199	**DRAWN BY: A SMIT**	**REVIEWED:**	**DATE: 25/01/11**
SHEET No. 1 OF 1 SHEETS	**No. 01/01/00**		

NOTE: THIS PART
NEEDS TO BE LASER
CUT. DETAIL VIEW A
TO BE MACHINED









1	Top Wheel Plate	4	Aluminium
ITEM	**DESCRIPTION**	**QTY**	**MATERIAL / SPECIFICATION**

UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

STELLENBOSCH UNIVERSITY

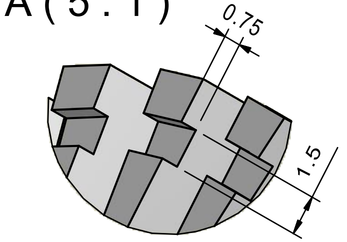
SCALE ON A3 2:1
MEASUREMENTS IN mm

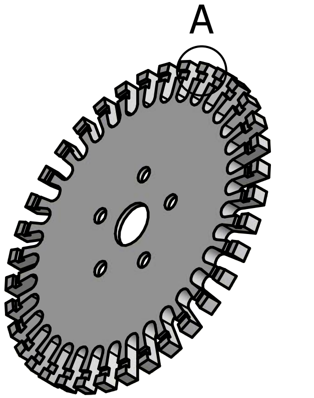
TITEL: Bottom Wheel Plate

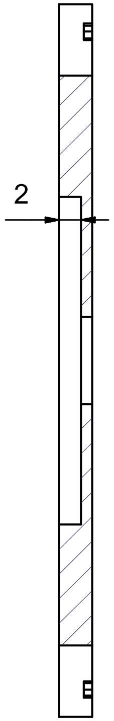
STUDENT No. 15979199 | **DRAWN BY: A SMIT** | **REVIEWED:** | **DATE: 25/01/11** | **SHEET No. 1 OF 1 SHEETS** | **No. 01/01/02**

NOTE: THIS PART NEEDS TO BE LASER CUT. DETAIL VIEW A AND SECTION VIEW B TO BE MACHINED

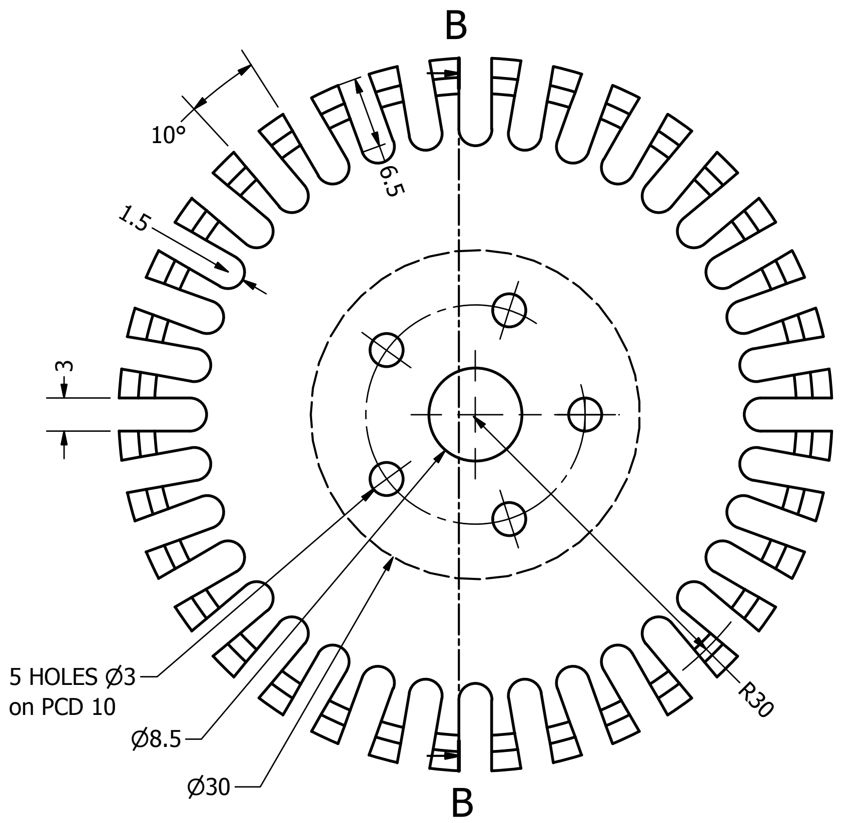
A (5 : 1)

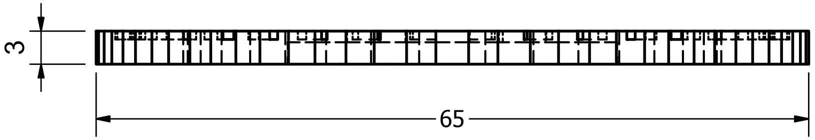






B-B (2 : 1)

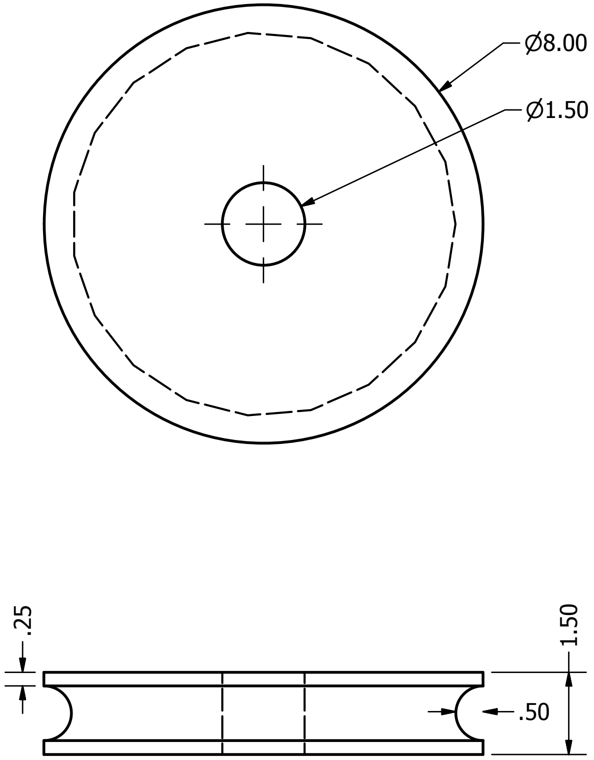


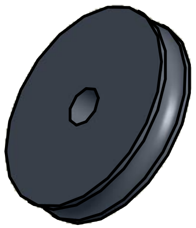


UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

1	Bottom Wheel Plate	4	Aluminium
ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION

STELLENBOSCH UNIVERSITY			SCALE ON A3 2:1 MEASUREMENTS IN mm	TITEL: Bottom Wheel Plate
STUDENT No. 15979199	DRAWN BY: A SMIT	REVIEWED:	DATE: 25/01/11	SHEET No. 1 OF 1 SHEETS	No. 01/01/02



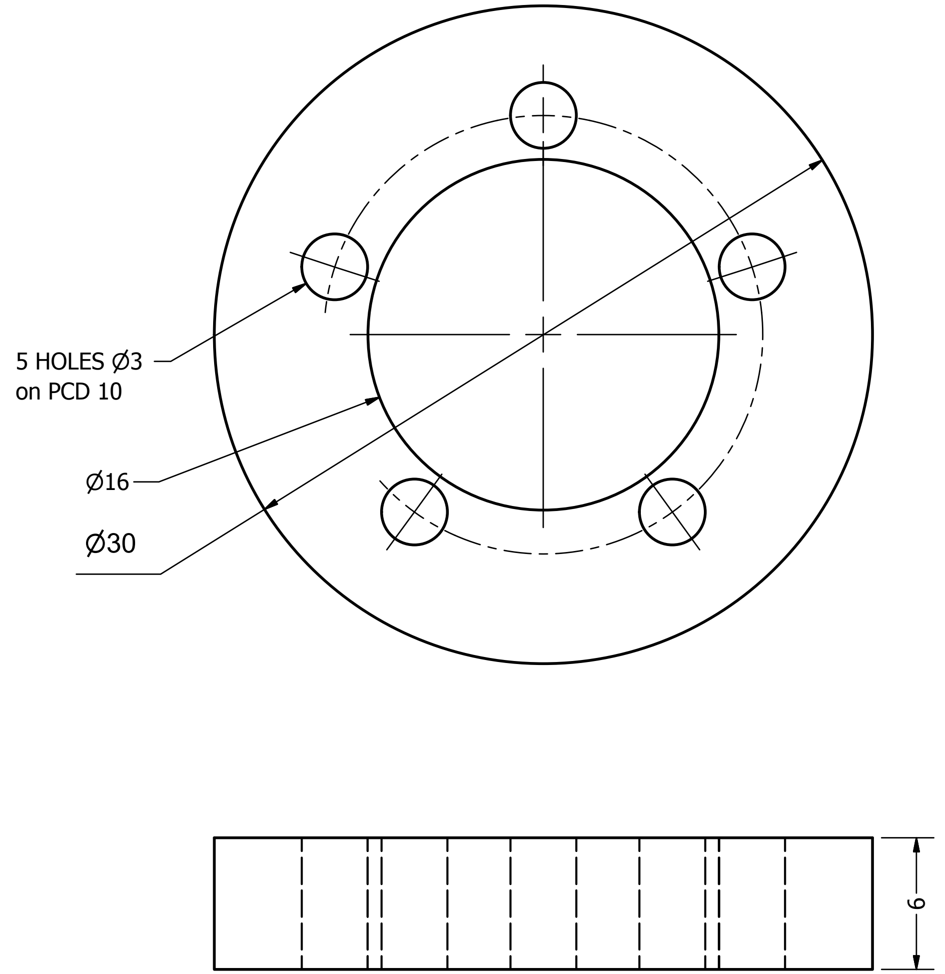


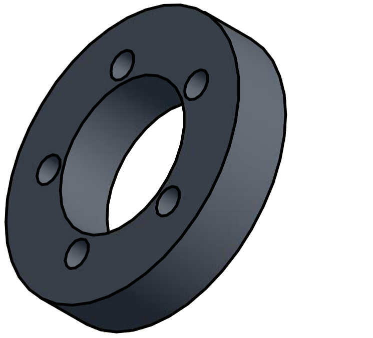
UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION
1	Omni Wheel Roller	144	PVC

STELLENBOSCH UNIVERSITY

SCALE ON A3 10:1 MEASUREMENTS IN mm	TITEL: Omni Wheel Roller		
STUDENT No. 15979199	DRAWN BY: A SMIT	REVIEWED:	DATE: 25/01/11
SHEET No. 1 OF 1 SHEETS	No. 01/01/03		





UNLESS OTHERWISE STATED

TOLERANCES ± 0,1

ANGLES 1˚

1	Hub Extension/Spacer	4	PVC
ITEM	**DESCRIPTION**	**QTY**	**MATERIAL / SPECIFICATION**

STELLENBOSCH UNIVERSITY

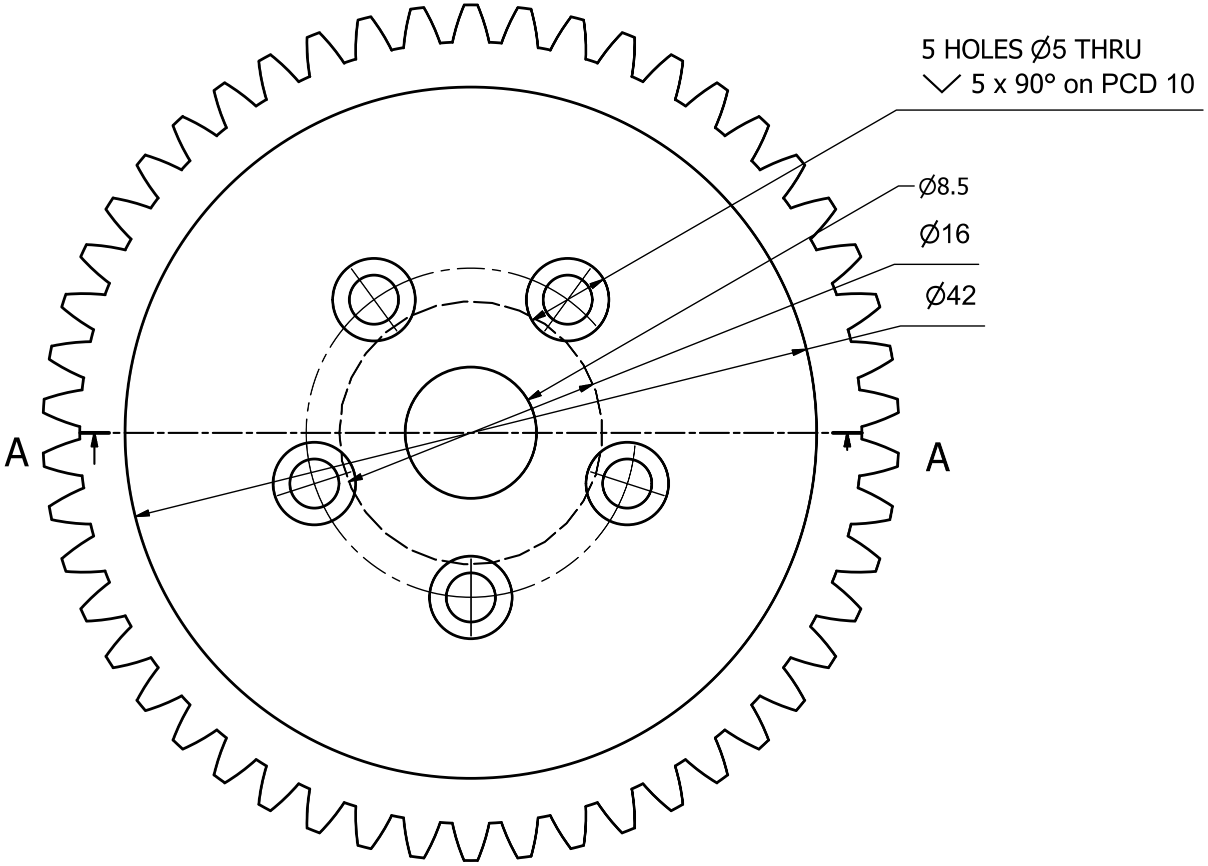
SCALE ON A3 4:1

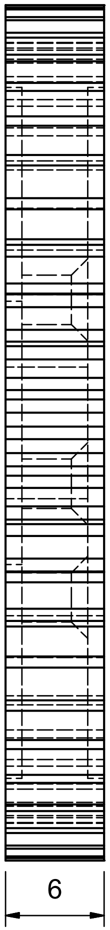
MEASUREMENTS IN mm

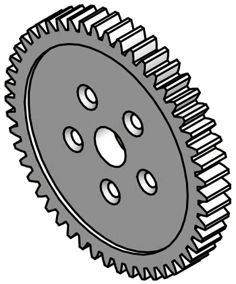
TITEL: Hub Extension/Spacer

STUDENT No. 15979199 | **DRAWN BY: A SMIT** | **REVIEWED:** | **DATE: 25/01/11** | **SHEET No. 1 OF 1 SHEETS** | **No. 01/01/04**

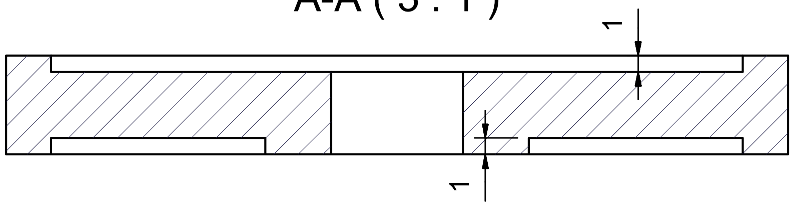
NOTE: THIS PART
NEEDS TO BE
MACHINED FROM
ORIGINAL PART
ACCORDING TO
DIMESIONS







A-A (3 : 1)

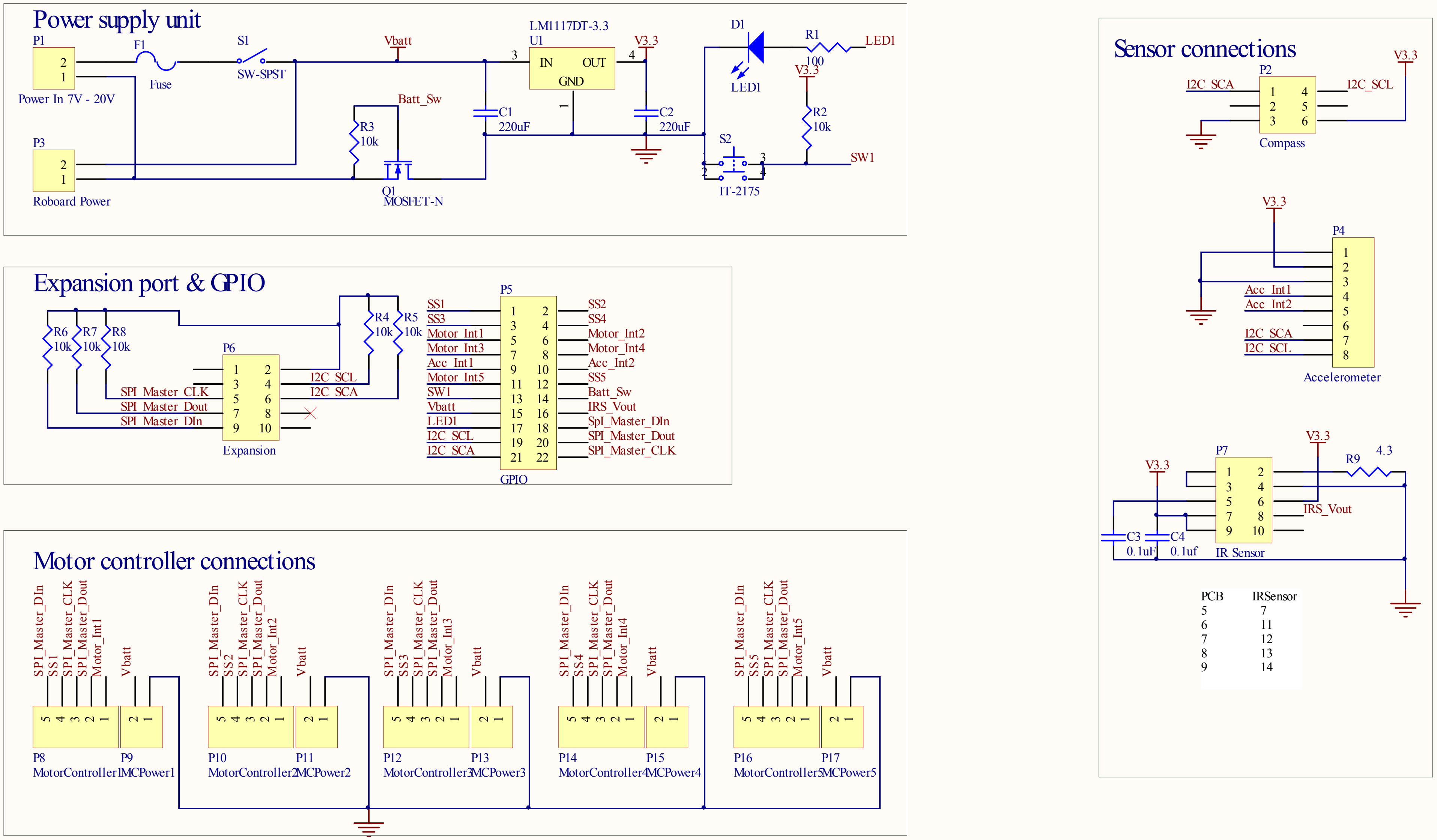


UNLESS OTHERWISE STATED
TOLERANCES ± 0,1
ANGLES 1˚

			1	Wheel Spur Gear	4	DELRIN 1 MODULE 50 TEETH
			ITEM	DESCRIPTION	QTY	MATERIAL / SPECIFICATION
STELLENBOSCH UNIVERSITY			SCALE ON A3 3:1 MEASUREMENTS IN mm	TITEL: Wheel Spur Gear		
STUDENT No. 15979199	DRAWN BY: A SMIT	REVIEWED:	DATE: 25/01/11	SHEET No. 1 OF 1 SHEETS		No. 01/01/05

Electronic Design Drawings

This appendix includes all the electronic circuit diagrams. It first shows the currently used circuits and then the recommended improved circuits that were discussed throughout Chapter 4. These improved circuits are indicated with x.1 in the version number. A bill of material is also given for the currently used circuits in Figure 1 and Figure 2.



TITLE: Hub Circuit Diagram

VERSION No. 1.0

STUDENT No: 15979199

DRAWN BY: A Smit

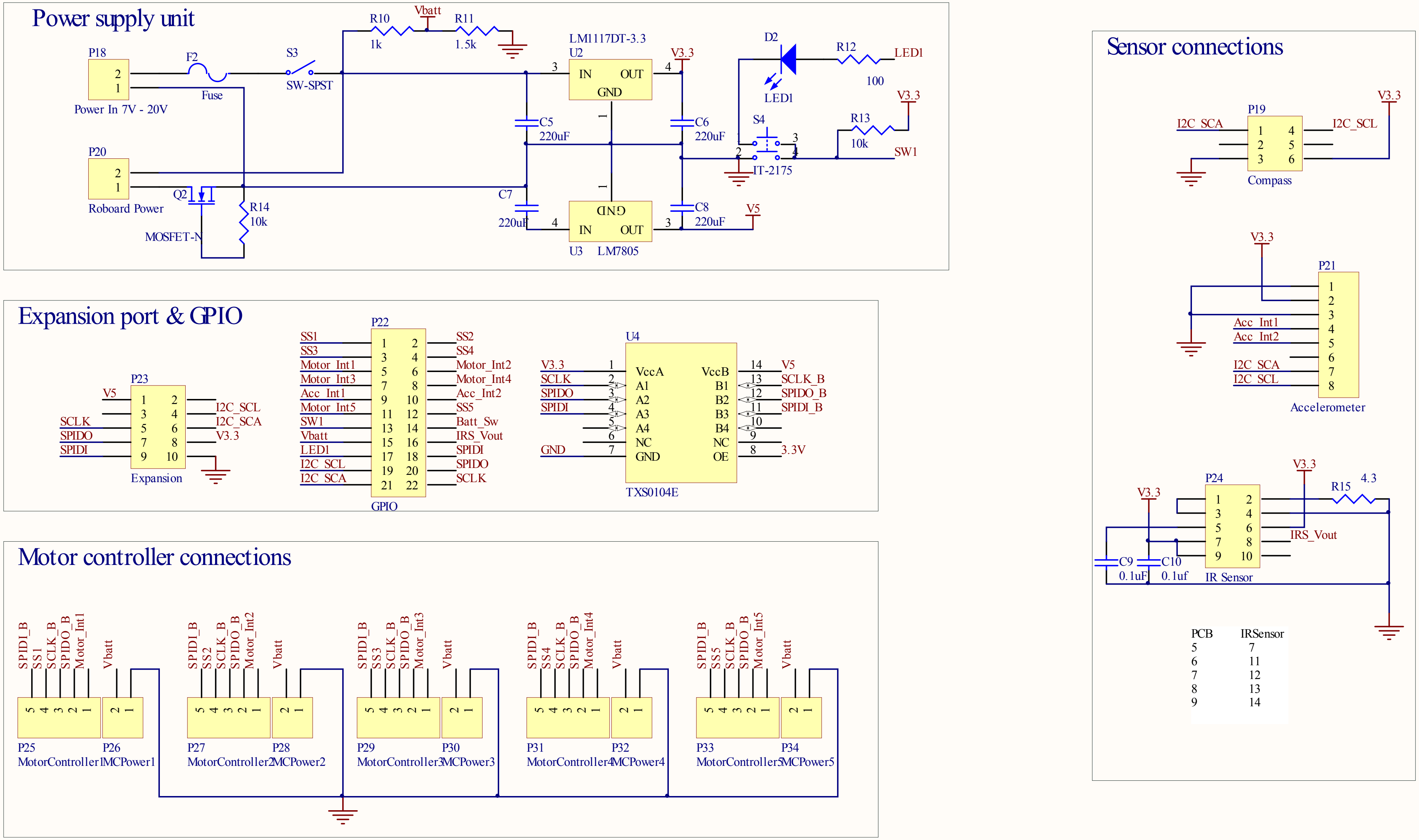
REVIEWED:

DATE: 25/01/11

SHEET 1 OF 2

EFU No. 1

STELLENBOSCH UNIVERSITY



PCB	IRSensor
5	7
6	11
7	12
8	13
9	14

TITLE: Hub Circuit Diagram

VERSION No. 1.1

STUDENT No: 15979199

DRAWN BY: A Smit

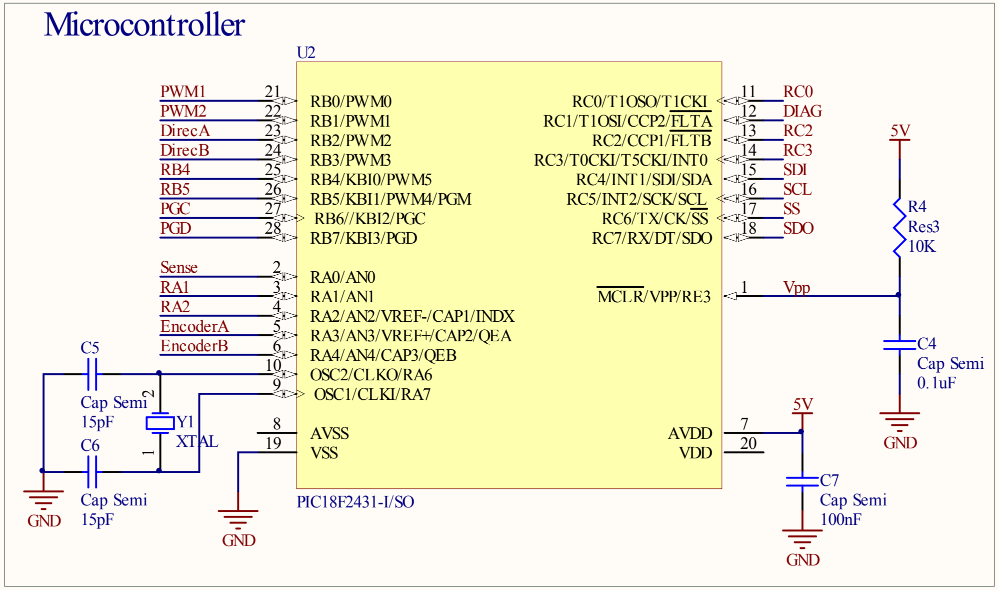
REVIEWED:

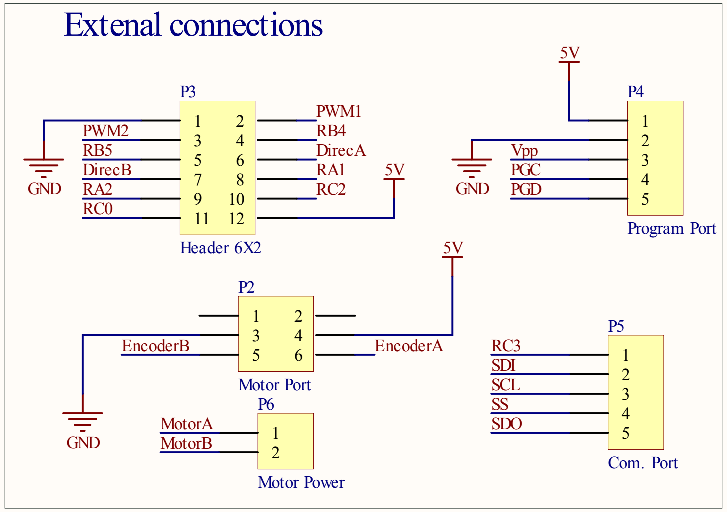
DATE: 25/01/11

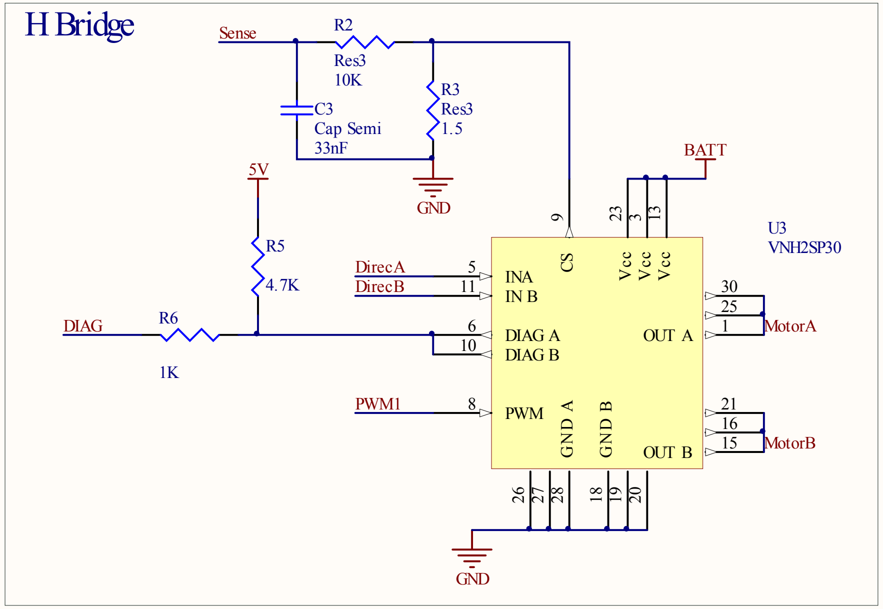
SHEET 2 OF 2

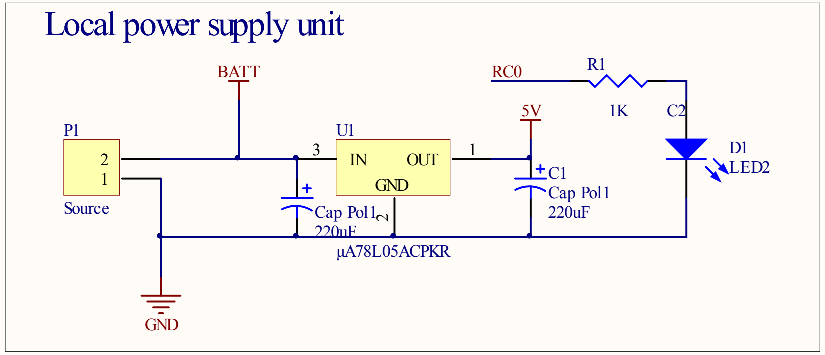
EFU No. 1

STELLENBOSCH UNIVERSITY


Microcontroller

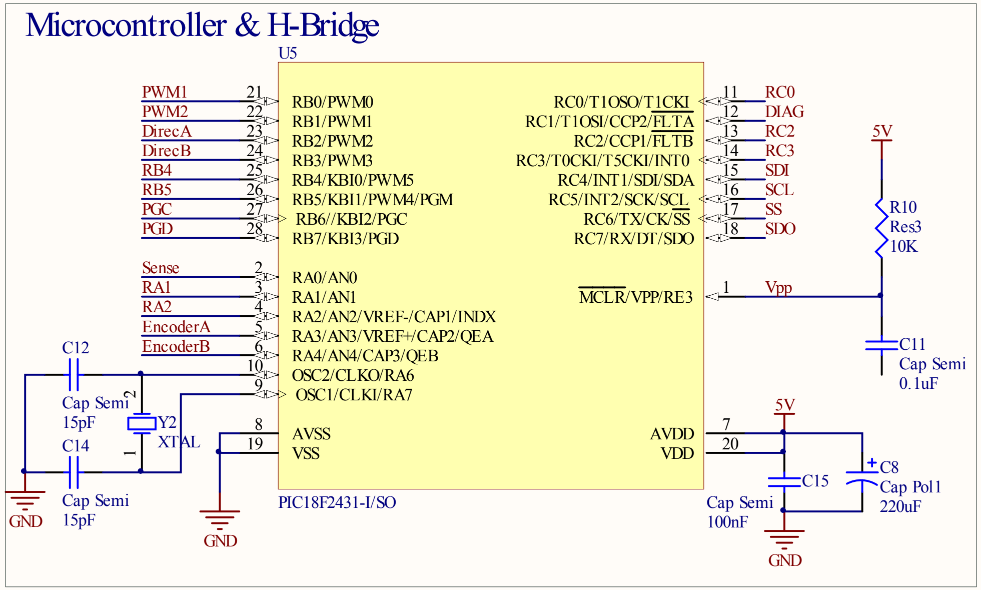

Extenal connections


H Bridge

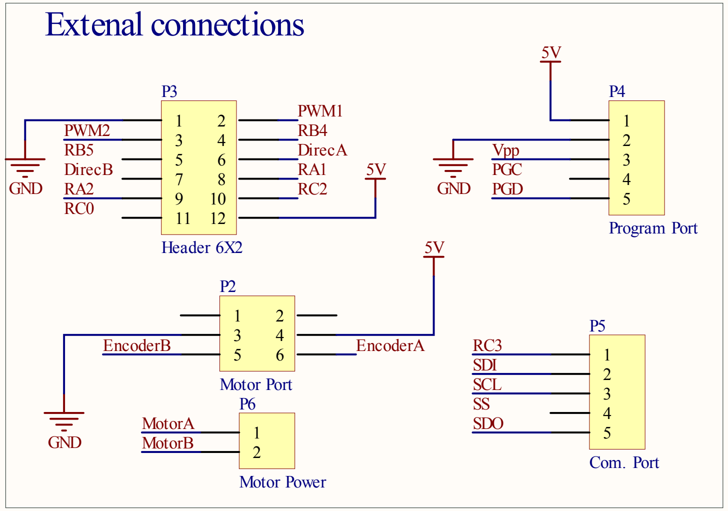

Local power supply unit

TITLE: Motor Controller Circuit Diagram			VERSION No. 2.0
STUDENT No: 15979199		DRAWN BY: A Smit	REVIEWED:
DATE: 25/01/11		SHEET 1 OF 2	EFU No. 6

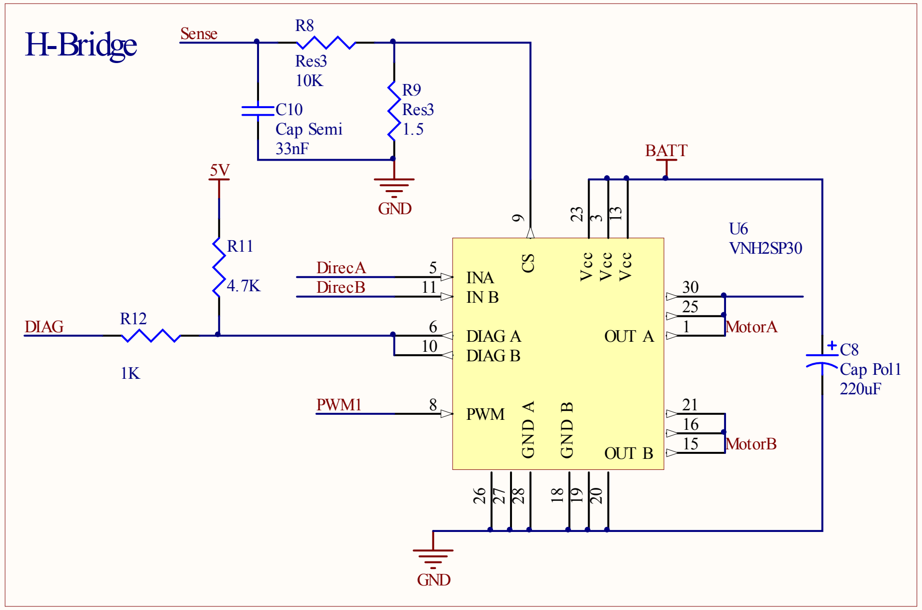
STELLENBOSCH UNIVERSITY



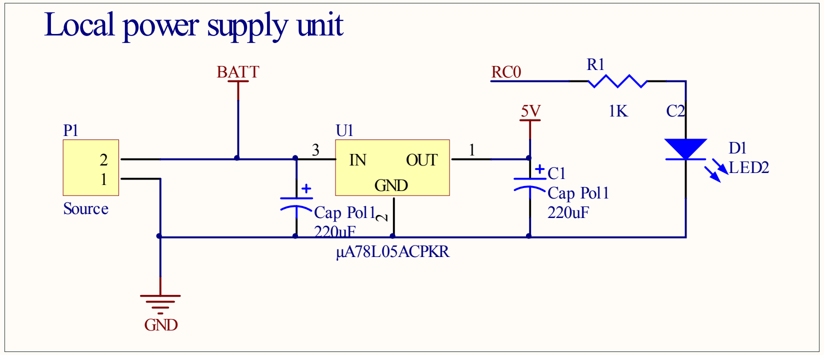
Microcontroller & H-Bridge



Extenal connections



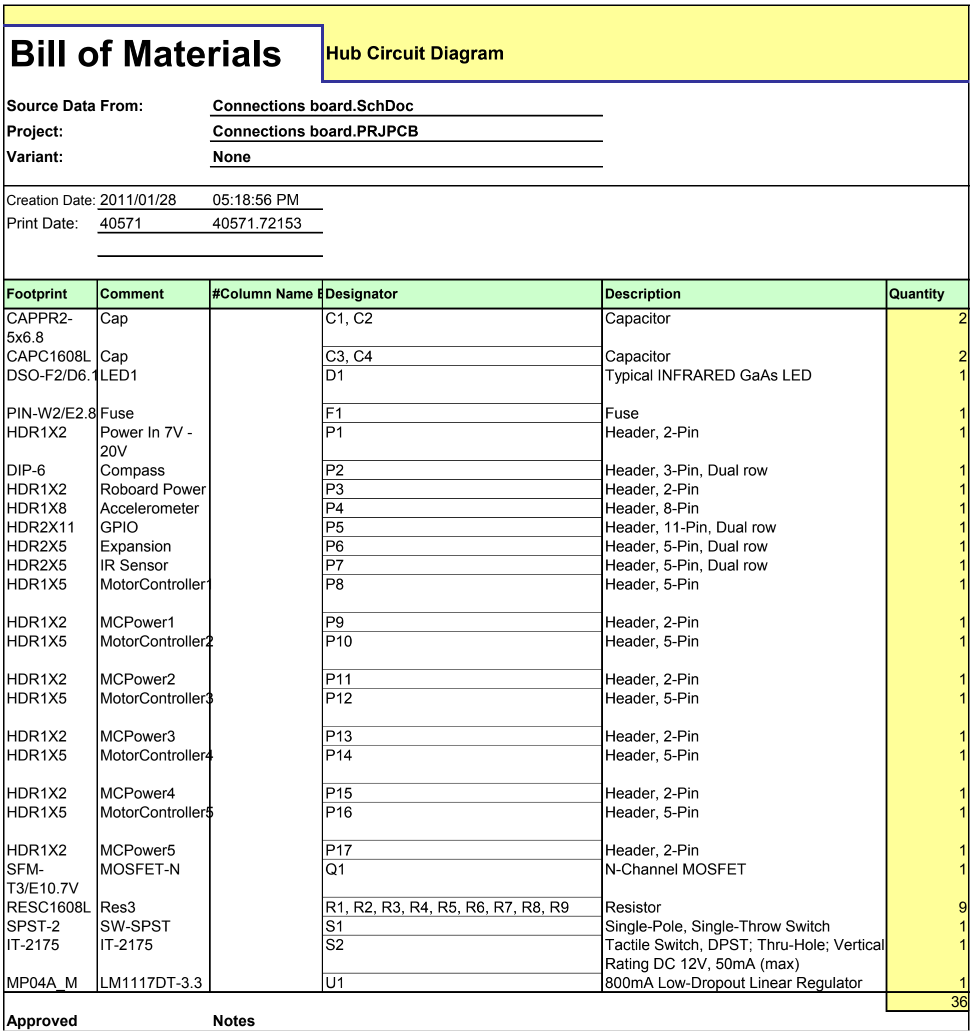
H-Bridge



Local power supply unit

STELLENBOSCH UNIVERSITY

TITLE: Motor Controller Circuit Diagram		VERSION No. 2.1
STUDENT No: 15979199	DRAWN BY: A Smit	REVIEWED:
DATE: 25/01/11	SHEET 2 OF 2	EFU No. 6



Bill of Materials **Hub Circuit Diagram**

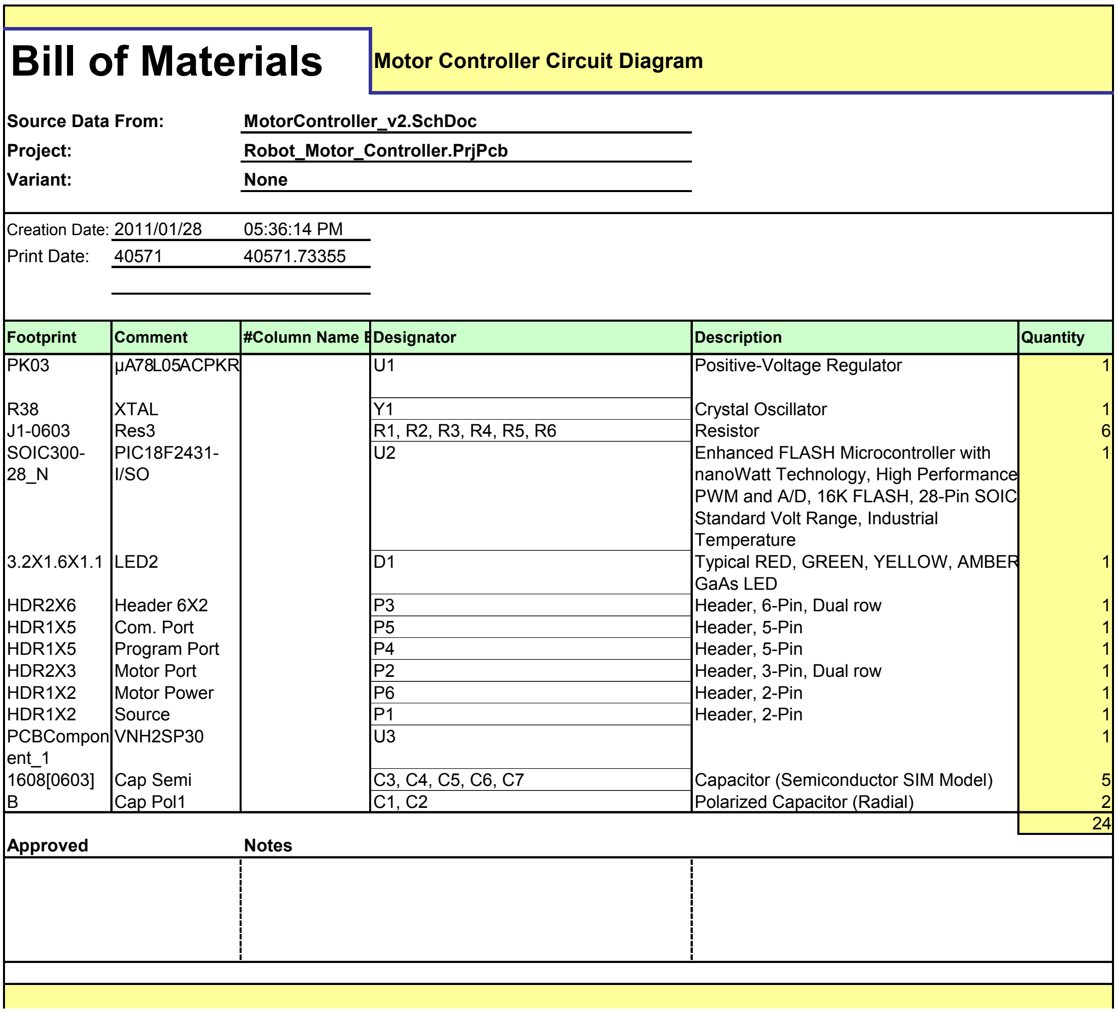
Source Data From:	**Connections board.SchDoc**
Project:	**Connections board.PRJPCB**
Variant:	**None**

Creation Date:	2011/01/28	05:18:56 PM
Print Date:	40571	40571.72153

Footprint	**Comment**	**#Column Name E**	**Designator**	**Description**	**Quantity**
CAPPR2-5x6.8	Cap		C1, C2	Capacitor	2
CAPC1608L	Cap		C3, C4	Capacitor	2
DSO-F2/D6.1	LED1		D1	Typical INFRARED GaAs LED	1
PIN-W2/E2.8	Fuse		F1	Fuse	1
HDR1X2	Power In 7V - 20V		P1	Header, 2-Pin	1
DIP-6	Compass		P2	Header, 3-Pin, Dual row	1
HDR1X2	Roboard Power		P3	Header, 2-Pin	1
HDR1X8	Accelerometer		P4	Header, 8-Pin	1
HDR2X11	GPIO		P5	Header, 11-Pin, Dual row	1
HDR2X5	Expansion		P6	Header, 5-Pin, Dual row	1
HDR2X5	IR Sensor		P7	Header, 5-Pin, Dual row	1
HDR1X5	MotorController1		P8	Header, 5-Pin	1
HDR1X2	MCPower1		P9	Header, 2-Pin	1
HDR1X5	MotorController2		P10	Header, 5-Pin	1
HDR1X2	MCPower2		P11	Header, 2-Pin	1
HDR1X5	MotorController3		P12	Header, 5-Pin	1
HDR1X2	MCPower3		P13	Header, 2-Pin	1
HDR1X5	MotorController4		P14	Header, 5-Pin	1
HDR1X2	MCPower4		P15	Header, 2-Pin	1
HDR1X5	MotorController5		P16	Header, 5-Pin	1
HDR1X2	MCPower5		P17	Header, 2-Pin	1
SFM-T3/E10.7V	MOSFET-N		Q1	N-Channel MOSFET	1
RESC1608L	Res3		R1, R2, R3, R4, R5, R6, R7, R8, R9	Resistor	9
SPST-2	SW-SPST		S1	Single-Pole, Single-Throw Switch	1
IT-2175	IT-2175		S2	Tactile Switch, DPST; Thru-Hole; Vertical Rating DC 12V, 50mA (max)	1
MP04A_M	LM1117DT-3.3		U1	800mA Low-Dropout Linear Regulator	1
					36

Approved **Notes**

Figure 1: Bill of materials for the hub - EFU 1



Bill of Materials

Motor Controller Circuit Diagram

Source Data From: **MotorController_v2.SchDoc**
Project: **Robot_Motor_Controller.PrjPcb**
Variant: **None**

Creation Date: 2011/01/28 05:36:14 PM
Print Date: 40571 40571.73355

Footprint	**Comment**	**#Column Name E**	**Designator**	**Description**	**Quantity**
PK03	µA78L05ACPKR		U1	Positive-Voltage Regulator	1
R38	XTAL		Y1	Crystal Oscillator	1
J1-0603	Res3		R1, R2, R3, R4, R5, R6	Resistor	6
SOIC300-28_N	PIC18F2431-I/SO		U2	Enhanced FLASH Microcontroller with nanoWatt Technology, High Performance PWM and A/D, 16K FLASH, 28-Pin SOIC Standard Volt Range, Industrial Temperature	1
3.2X1.6X1.1	LED2		D1	Typical RED, GREEN, YELLOW, AMBER GaAs LED	1
HDR2X6	Header 6X2		P3	Header, 6-Pin, Dual row	1
HDR1X5	Com. Port		P5	Header, 5-Pin	1
HDR1X5	Program Port		P4	Header, 5-Pin	1
HDR2X3	Motor Port		P2	Header, 3-Pin, Dual row	1
HDR1X2	Motor Power		P6	Header, 2-Pin	1
HDR1X2	Source		P1	Header, 2-Pin	1
PCBComponent_1	VNH2SP30		U3		1
1608[0603]	Cap Semi		C3, C4, C5, C6, C7	Capacitor (Semiconductor SIM Model)	5
B	Cap Pol1		C1, C2	Polarized Capacitor (Radial)	2
					24

Approved **Notes**

Figure 2: Bill of materials for the motor montroller (MC) - EFU 6

Software Diagram

The complete software system that includes all the classes is shown in Figure 3.

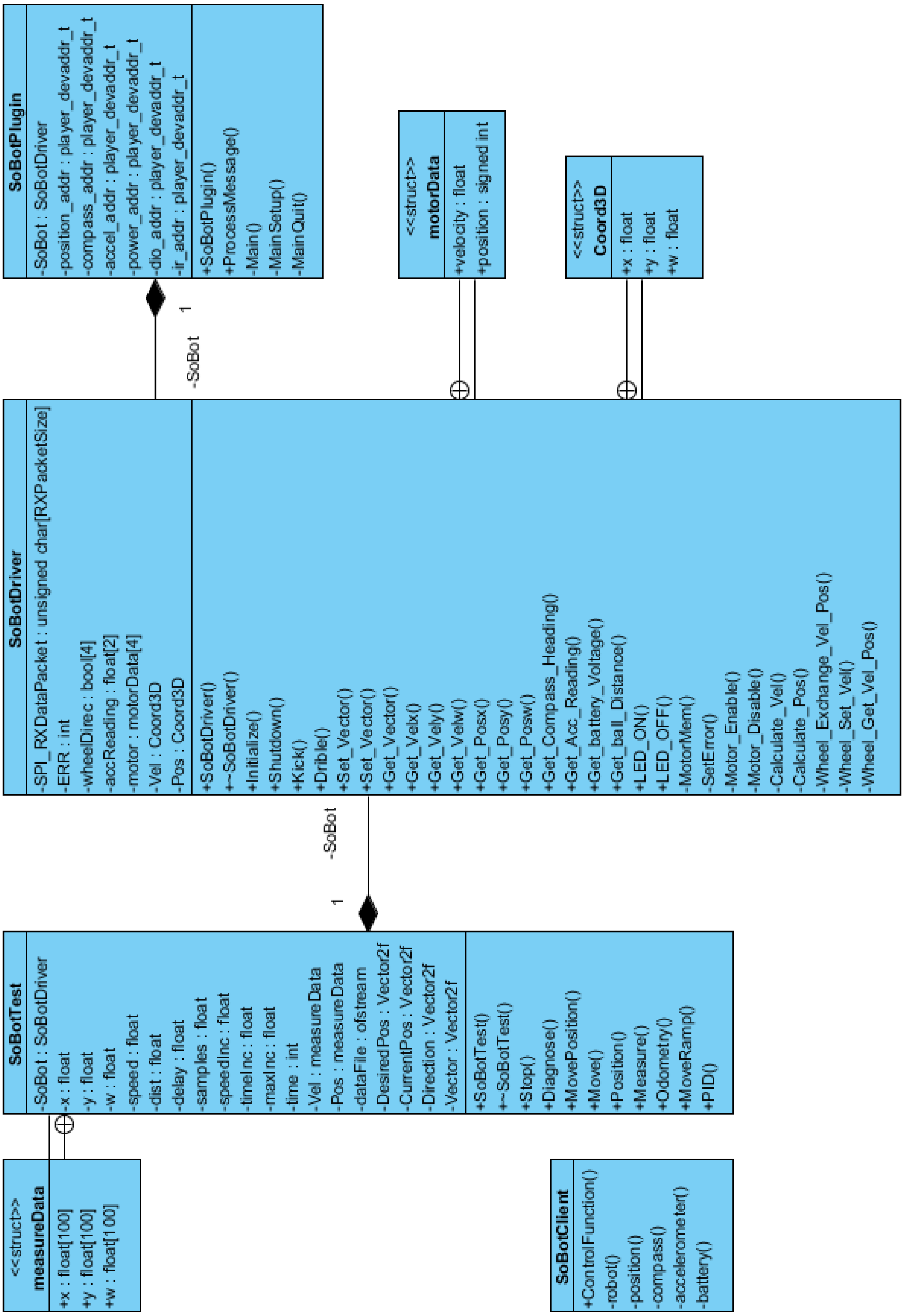


Figure 3: UML diagram of the software system

CD Contents

The attached CD contains the following items:

1. The mechanical drawings from Appendix A as well as the Inventer CAD model files.
2. The electronic circuits shown in Appendix B together with the electronic design files for Altium Designer.
3. All the software that was written for this research, including the driver class library, the Payer middleware plug-in, and the firmware for the MC as well as the documentation thereof.
4. Raw data from the performed tests and the Matlab scripts that were used to proses and plot the data.
5. Installation files for third-party software that is needed for development on the platform including the Player server, Roboard RB 100 library and Linux distributions.
6. Instructions on how to set up a completely installed image file that can be copied to the micro-SD card on the central controller (CC). The image includes all the software needed for development on the platform.
7. Datasheets, bill of materials, photos, videos and research papers.

Contact the author at: allasmit@gmail.com