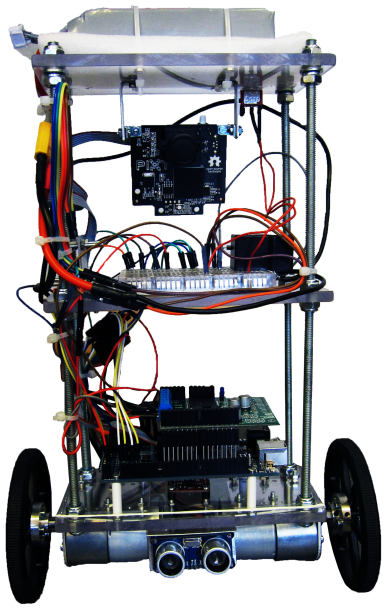
CHALMERS

BACHELOR THESIS



ONE ROBOT TO ROLL THEM ALL

Construction of a self-balancing, two-wheeled, football playing robot

Lisa Jodensvi
Victor Johansson
Charlotte Lanfelt
Samuel Löfström

Supervisor Seyedmehrdad Hosseini
Examiner Jonas Fredriksson

Department of Signals and Systems
Chalmers University of Technology

Göteborg, 2015

SSYX02-15-22

Abstract

Robots and autonomous systems are today a fast growing branch of technology within both industry and society. This paper presents an attempt to create an autonomous, self-balancing, two-wheeled robot that should be able to play football. Although several satisfying results were presented the goals were not fulfilled and the robot never got stable enough to balance by itself.

A two-wheeled robot is an unstable system and cannot balance without a stabilising controller. To make the system stable two mathematical models are derived using Newtonian mechanics. First, a full state feedback controller is designed, using both linear-quadratic regulation and pole placement, to stabilise the balancing and movement of the robot. Later, a cascaded PI-PID controller is designed to only stabilise the balance of the robot. The models and the controllers are verified using software simulations in Matlab/Simulink. The identified control parameters are then transferred to the physical model for verification.

The robot body is constructed using three shelves which are assembled with threaded rods. These shelves hold all the components necessary for the robot to function; the processor, the motors, the sensors and the battery. A gyroscope and an accelerometer are fused using a complementary filter to measure robot angle. Rotary encoders are used to measure robot velocity and position. This data is then used by the controller.

Sammandrag

Robotar och autonoma system är idag snabbt växande teknologiska forskningsområden inom både industri och samhälle. Denna rapport presenterar ett försök till att skapa en autonom, självbalanserande, tvåhjulig robot som även ska kunna spela fotboll. Trots flera goda delresultat uppfylldes inte projektmålen och roboten blev aldrig stabil nog för att balansera på egen hand.

En tvåhjulig robot är instabil och kan inte balansera utan en stabiliserande regulator. För att stabilisera systemet med reglerteknik härleds två matematiska modeller med hjälp av klassisk mekanik. Först designas en tillståndsåterkopplad regulator, med hjälp av linjär-kvadratisk reglering och polplacering, för att stabilisera robotens balans och rörelse. Senare designas en kaskadreglering med PI-PID regulator för att endast stabilisera robotens balans. Modellerna och regulatorerna verifieras med hjälp av mjukvarusimulationer i Matlab/Simulink. De identifierade regleringsparametrarna överförs sedan till det fysiska systemet för verifiering.

Robotens kropp konstrueras med tre hyllor som fästs samman med gängstänger. Hyllorna bär upp alla komponenter som är nödvändiga för robotens funktion; processorn, motorerna, sensorerna och batteriet. Ett gyroskop och en accelerometer kombineras med hjälp av ett komplementärfilter för att mäta robotens vinkel. Rotationsgivare används för att mäta robotens hastighet och position. Datan används därefter av regulatorn.

Contents

1 INTRODUCTION

This chapter is an introductory section to give the reader an understanding of the relevance of this project. Also, the purpose, where the project goals are stated, and the boundaries used is presented as well as an overview of the main problems concerning the goals.

1.1 Background

Robotic technology is a fast growing industrial market. More applications and newer technologies are developed every day. Robots have a long history in the industry where they have replaced human workers in areas which may be too dangerous for humans or where it may be more efficient to use robots. Nowadays, robots have even entered our homes as, for example, autonomous vacuum cleaners and lawn mowers. More is probably to come and this makes this area of research interesting. Therefore, the topic is highly relevant for both Chalmers as a university and for engineering students.

Lately, a sport has emerged and has increased interest in autonomous robots, the sport is called robot football. Every year, football tournaments are held for robot teams by an organisation called FIRA (Federation of International Robot-soccer Association).[1] Fully or semi-autonomous robots are required to find a ball, make proper decisions and work as a team to win the game. Last year a robot football project was made according to FIRA RoboSot rules.[2] The design was a cylinder-shaped robot with three wheels which was supposed to find a ball, catch it, transport it to the goal and score.

The project this year takes robot football in a new direction with two wheels instead of three. The FIRA rules will not be used in the design of the robot. A two-wheeled robot requires a self-balancing system to work. This makes the project an interesting control theory task, containing many of the technical engineering fields, such as mechatronics, control engineering and electronics design.

1.2 Purpose

The purpose of this project is to design and build a two-wheeled robot which is able to balance by itself, move on its own and finally locate a ball and score a goal. The project is divided into three parts: two main goals and one secondary, optional goal. These will be handled in order, to make sure that the solutions work as they should.

Main goals:

- Design and build a self balancing two-wheeled robot which can move on its own.
  - The robot will be dimensioned with only two wheels and then balanced with an autonomous control system.
- Make the robot able to identify a stationary ball and move the ball.
  - To play football the robot must know what object to identify as a football and how to manoeuvre it.

Secondary Goal:

- Make the robot able to identify a football goal and score.
  - To be able to score the robot must also know what object to identify as a goal and how to place the ball in that goal.

1.3 Problem description

To design and build the desired product the problems that needs to be solved first needs to be identified before reaching the final result. The relevant problems which the robot has to overcome are the following:

- Balance on its own.
  - The robot must be able to balance on only two wheels with the help of balancing control algorithms and different sensors that measure the angle and speed of the robot.
- Turn and move autonomously.
  - The robot should turn and move autonomously in relation to where the ball is located and later on to where the goal is located.
- Receive image data and process it.
  - The robot will get input data of the surrounding environment from a camera and from sensors. The data will then be processed by a micro-controller to make interaction possible.
- Interact with the ball.
  - When the ball is located through the image processing the robot must be able to move autonomously towards it. As the robot approaches the ball it should be able to manoeuvre the ball in a chosen direction.

1.4 Boundaries

The main boundaries of this project are to stay within the specified budget of 5000 SEK and the given time frame for the bachelor's thesis course at Chalmers University of Technology. Another boundary of Chalmers projects is to stay within the Chalmers policies.[3] To consider sustainability aspects this project will aim to use components and materials that can either be reused in future projects or recycled after use. This project will only operate within those constraints.

Since the goal is to build a two-wheeled, self-balancing robot, this project will only aim to explore a construction of that sort. The area where the robot will move must be cleared and enclosed without any interfering objects that can disturb the robot's object identification ability or the movement of the robot.

All the objects that the robot interacts with must be easy to find by the image recognition camera. Therefore, the objects need to be in bright colours that differs from each other and from the background. To simplify the robot's movements, the objects must be still when the robot sets out to look for them. There must also be sufficient light for the image recognition to work properly.

2 TECHNICAL FRAMEWORK

Before constructing a physical product there needs to be sufficient scientific material to base the design on. In this chapter the theory behind important areas of the project are introduced and described. The theory is constrained to methods, models and technology that is relevant for this project and does not cover anything outside of the project boundaries. The balancing section describes how stable system models can be created with mathematical modelling and controller design. Object identification mentions methods for locating various objects via computer vision. The path planning section describe methods of autonomous steering. Described in this chapter are also declarations and explanations of all hardware components that are of importance for this report.

2.1 Balancing

There are many different ways to mathematically model a self-balancing two-wheeled vehicle. The main differences between the existing mathematical models are what dynamic equations have been used to derive the model, as well as how many degrees of freedom have been modelled.[4]

In general, one of two different sets of dynamic equations are used when modelling the system, Euler-Lagrangian mechanics or classical (Newtonian) mechanics. Euler-Lagrange seems to better model more complex systems. It also transitions well to state-space representation, since the generalised coordinates may later be used as state variables.[5] Newtonian mechanics seem better suited to quickly handle systems with relatively small complexity and do so in a compartmentalised manner.[6]

When it comes to degrees of freedom, the tilt angle of the robot as well as the longitudinal displacement are often modelled, since the motion of the robot consists of balancing and moving in a straight line. For robots which are meant to be able to turn around the z-axis with high accuracy, the yaw angle is included as a degree of freedom.

2.1.1 State-space representation

When describing a dynamic system it is often helpful to use state-space representation. The idea is that there at all times exists a vector, the state vector, which fully represents the dynamic state of the system.[7] A linear state-space representation is generally written as:

$$\dot{x} = Ax + Bu \tag{1}$$

$$y = Cx + Du \tag{2}$$

where x is the state vector containing the state variables, u and y are the system input and output, respectively, and the matrices A, B, C and D define the state dynamics of the system.[8] The state variables of the state vector are usually chosen to be system variables of interest for desired control and observation of the system. State space models are especially useful when describing systems with multiple inputs and outputs, since the order of the vectors and matrices only needs to be adjusted accordingly.

A benefit with a system that is represented by a state-space model is that it is possible to evaluate the system's stability by checking the eigenvalues of the A matrix. If an eigenvalue is not strictly negative and real it represents an unstable pole. Another benefit is that the controllability is easily checked by evaluating the controllability matrix, $C = \begin{bmatrix} B & AB & A^2B & ... & A^{n-1}B \end{bmatrix}$. The system is controllable if C has full rank.[8]

2.1.2 State feedback control

State feedback control is a well documented way of controlling a dynamic system on state-space form, see figure 6 in Method.[9] Generally when controlling a system, the goal is to minimise the error value, $e = r - y$. Note that the reference signal, r, may be zero for systems which do not track reference values. With state feedback control, the states are used as inputs to the controller which through a feedback loop determines the correct control signal to apply to the system, thereby setting the control signal to $u = r - Kx$.

The weighting matrix, K, which is applied to the feedback-loop of the system, alters the poles of the system to stabilise and control it. The weighting put on the different state variables is determined considering the desired behaviour of the system. The higher the weighting on a specific state variable, the more impact the changes in that variable have on the system's behaviour.

If there is a need for the system to track specific changes in variables, reference signals may be added. N is a pre-compensating coefficient which makes sure that the error of the controlled system is still zero by compensating for the fact that the reference signal is not compared to the output signal of the system but rather the weighted, looped states. Adding N changes the control signal to $u = Nr - Kx$.

2.1.3 Linear-quadratic regulation

Linear-quadratic regulation (LQR) is an optimisation algorithm used to control a system by minimising the system's deviations from desired values.[7] It is of great help when dealing with systems which are multiple-input multiple-output (MIMO). The goal is to minimise the cost function $J = \int_0^\infty (x^T Q x + u^T R u) dt$ of the linear system $\dot{x} = Ax + Bu$ and produce the weighting matrix K for the feedback control $u = -Kx$. Q and R are weighting coefficient matrices which represent the relative importance of the control signal and error value, respectively, for J. K is found to be $K = R^{-1}B^T X$ where X is found by solving the Riccati equation $A^T X + XA - XBR^{-1}B^T X + Q = 0$.[7] This procedure can be swiftly carried out with the help of software tools like Matlab. In practice, LQR comes down to finding the proper state feedback controller, K, by inputting values for Q and R which correspond to the desired behaviour of the system.

2.1.4 Pole placement

The purpose of pole placement is, just as with LQR, to find the weighting coefficient matrix K for a system. However, instead of using the Q and R matrices that are created in LQR the poles of the system are placed at predetermined locations in the s-plane. This is a convenient method since the placed poles correlate directly to the system's eigenvalues and therefore the stability of the system.

The principle can only be used on a controllable system and is done by evaluating the existing characteristic equation for the full state feedback system:

$$det[sI - (A - BK)] \tag{3}$$

The K matrix is then picked by placing the poles as desired and adapting the K matrix to what it needs to be for the poles to be in the desired places.[7]

2.1.5 PID-control

PID is short for proportional-integral-derivative controller. The function of the controller, just like the state feedback controller, is to minimise the error value $e = r - y$ by determining a fitting control signal u, see equation 4. All of the three different parts of the PID-controller put their

focus on reference and error with different relations to time. With a P-controller the present error is reduced with a proportional change. The I-controller integrates over time and can reduce the error based on past errors. The D-controller derives and measures the present change of the error and uses this to predict future errors. When all three controllers are used together it is called a PID-controller. But just as often it is sufficient to use just one or two of them which are then called P-, I-, PI- or PD-controllers. It all depends on the system at hand and how it needs to be controlled.

The significance of the three parts of the controller on the control effort is determined by their respective weighting coefficient; K_p, K_i and K_d. Through mathematical models and methods for variable estimation it is possible to get reasonable values for the K-variables.[7] This is however very complex and time demanding, since the model and the real system usually differ appreciably and the model therefore needs to be heavily adjusted before presenting any useful results. One way to avoid this is to search for an existing model for a system similar to the one at hand, then start with that one and go straight on to the final calibrations.

$$u(t) = K_p e(t) + K_i \int (e(t))dt + K_d \frac{de}{dt} \tag{4}$$

Cascade control
Sometimes there are two variables interfering with one another that both need to be controlled. This can be done simply by controlling them within the same control loop. Too big control loops can however become a slow control loop together. If one of the two variables is in need of a fast control loop it is useful to build a cascade controller from the two controllers, see figure 8 in Method. The outer loop ($loop_1$) gets reference signal (r_1) as an input. After an error (e_1) is calculated it gives a control signal (u_1) which is also the reference (r_2) for the inner loop ($loop_2$). The inner loop also calculates an error (e_2) and sends forward a control signal (u_2) that is adjusted in the controller and transformed into an output (y_2) which is measured and sent back for error (e_2) calculation. The output (y_2) continues in the outer loop through the controller and becomes the output (y_1) which is sent back to error (e_1) calculation for the outer loop. In that way the inner loop does not have to wait as long as the outer one for feedback.[9]

2.2 Object identification

A computer or a small processor can, with the help of sensors, identify and track an object in many different ways. Lately the technology of computer vision have been much improved and is now one of the best methods for object identification.

Computer vision consists of two parts, image processing and image recognition, and there are many different ways to achieve this. Usually the word image is referred to instead of picture, since computers store numerical images of a picture or a scene. The processing part is how the data is collected, changed and stored. Usually the image needs to be deconstructed so that only a few details remain that are easily distinguished. Enhancement of an image usually refer to the restoration of the image and it is seldom needed for any picture recognition. Image recognition is the programed understanding for the computer, how it compares and connects the image to known objects.[10]

2.3 Path planning

A common problem with control engineering is how to make a controlled object move to a specified point in space. It is necessary for an autonomous robot to make proper movement decisions and this can be achieved with path planning algorithms.

A simple algorithm that can move a two-wheeled robot to a desired location is for example one where, firstly, the difference in the angle between the final location and the robot's moving direction will decrease by turning the robot. This can be achieved, for example, with a P-controller to control

the turning as can be seen in [11]. By letting the motors run with the same speed but in different directions, a two-wheeled robot would turn on the spot. The sign of the angular difference value determines in what direction the robot should turn. When the robot is aligned with the final location, the robot only needs to go forward to get there.

Another basic method has been investigated in [11]. This approach is almost the same as the one mentioned above but the robot changes direction as it moves towards the destination rather than first turning and then moving forward. The movement of the robot towards a destination, in this case a ball, can be illustrated as seen in figure 1. The wheel velocities determine the turning of the robot, as they can be different from each other. In figure 1, the turning is achieved by letting v_l and v_r be different from each other:

$$v_r = v + \omega\frac{B}{2} \tag{5}$$

$$v_l = v - \omega\frac{B}{2} \tag{6}$$

where v is the velocity of the robot, ω is the angular velocity around the robot's centre and B is the width of the robot body.[11] By controlling both wheels the robot can either go straight ($v_r = v_l$, ω=0) or turn ($v_r \neq v_l$, $\omega \neq 0$), either on the spot or while driving forward. If v_r>v_l, the robot will turn counterclockwise. In the same way, it will turn clockwise if v_l>v_r. This enables the use of a velocity difference variable to be added to one of the wheels' velocity while the same difference is subtracted from the other wheel's velocity for making turns while, for example, going towards a destination. The velocity difference can be seen in the formula's above (for v_r and v_l) as $\omega\frac{B}{2}$, that is, the transformed angular velocity ω.

Thus, determining a destination by finding the distance and angular difference with respect to the destination, the robot shall control the speed of each wheel to minimise the angular difference as well as the distance between itself and the final destination.

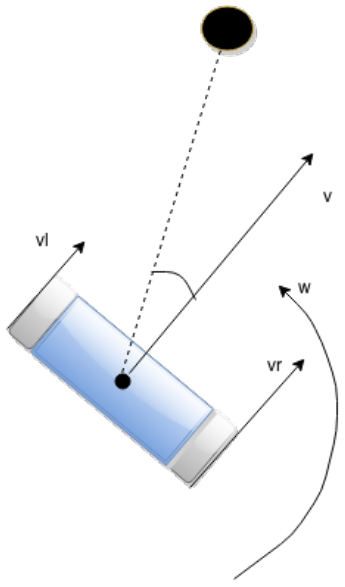


Figure 1: Illustration of a turning two-wheeled robot. Based on [11] and [12].

2.4 Micro-controllers

There exist several different kinds of control units, so called micro-controllers. They can be used in many different ways and are more or less advanced with different ports and compatible parts. They can be compared to a small computer containing a processor, memory storage and connector-pins for input and output.

2.4.1 Arduino

Arduino is an open source micro-controller which uses a simple interface for programming in ports and out ports. By controlling these it is easy to control the components of choice that have been plugged into the Arduino. It comes in different micro-controller designs with different inputs and outputs. One version is the Arduino Mega ADK which consists of 54 digital inputs or outputs, 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection and an ICSP(In-Circuit Serial Programming) header.[13]

2.5 Brushed DC motor

Brushed DC motors are motors designed to run on direct current which can change their speed by changing the operating voltage. Basically, they consist of a coil that generates a magnetic field around the armature when powered. The magnetic field then causes the armature to rotate since the left side is pushed away from the left side of the magnetic field and drawn to the right side and vice versa for the right side, see figure 2. When the armature is horizontally aligned the current is reversed by the commutator which causes reversed polarity of the magnetic field. That keeps the armature rotating since the poles of the magnetic field changes. In this way they can make full revolutions and run both forwards and backwards.[14]

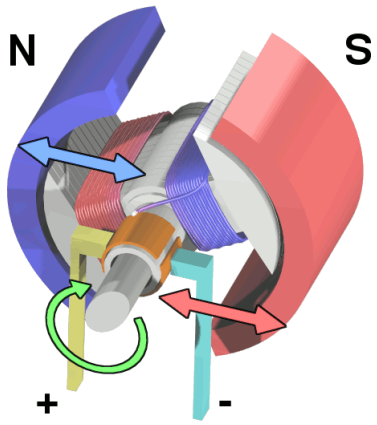


Figure 2: The inside of a brushed DC motor. Used under CC BY-SA.[15]

2.6 Sensors

When attempting to control a robot it is important to gather the essential information from the surroundings and compare it with the references of the system. In a real-time system, the sensors must be able to give accurate and real-time feedback with small intervals.

2.6.1 Motor encoders

Encoders can be used together with motors to determine parameters regarding the rotational motion of the motor shaft, like speed and position. With more than one encoder, the direction of the rotation can be determined. When this is done on more than one motor, the direction of the moving object can be determined as well as the turning of the object.

The encoders are mounted on the electrical motors and turn the mechanical motion of the motor shaft into electrical signals by using different methods depending on the type of the encoder. There exist for example magnetic encoders, optical encoders, absolute rotary encoders and many more. Magnetic encoders are one of the most common and it works by means of converting the mechanical motion of the motor into electrical signals by utilising the Hall effect.[16][17] The output is a pulsed voltage (high or low) determined by the strength of the magnetic field from a magnetised rotating disk.

There are also quadrature encoders which are encoders with two channels (two outputs). One of the channels is 90 degrees phase shifted relative to the other so that it is possible to determine direction and to achieve a higher resolution.[18]

2.6.2 Accelerometer and gyroscope

To measure the angle of an object an accelerometer or a gyroscope can be used. They both use different techniques to determine the angle. The accelerometer measures the object's acceleration in three axes by translating the relative movement of small masses within the sensor into an acceleration relative to the gravitational force. Using trigonometry, it is possible to calculate the angle of the resulting vector of the vertical and horizontal axes of the accelerometer. Since the accelerometer is very sensitive to applied forces, and since the movement of the measured object affects the measured value, the result becomes noisy. Therefore, it is not possible to use that value when trying to determine a stable angle.[19].

The gyroscope measures the angular acceleration with the help from vibrating elements within the sensor that senses force and rotation. To get the angle the angular acceleration is integrated which gives a good stable value at first. Eventually, small errors will accumulate due to the integration of the angular velocity and it will gradually drift towards a larger and larger error.

By fusing together the fast accuracy of the gyroscope with the slow accuracy of the accelerometer, a stable and accurate angle measurement can be achieved. This can be done with the help of a complementary filter, see figure 3. The complementary filter combines the angle given by the two sensors by applying a high-pass filter to the gyroscope readings and a low-pass filter to the accelerometer readings and adding them together [20]. The discrete equation for the complementary filter is shown in equation 7.

$$\theta = \alpha \cdot (\theta + gyroAngle * sampleTime) + (1 - \alpha) \cdot accelerometerAngle \tag{7}$$

where α is the filter coefficient between 0-1 which determines to which degree the final angle measurement will depend on each of the two sensors.

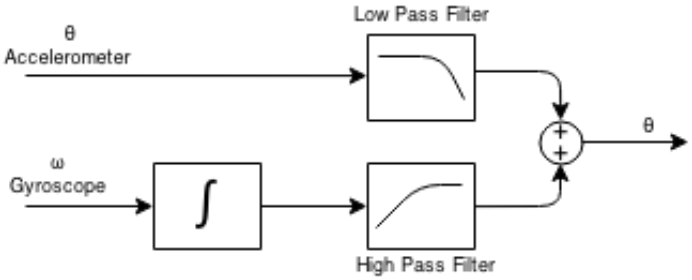


Figure 3: Illustration of a complementary filter.

2.6.3 Pixy camera

The CMUcam5 Pixy is an image sensor board with an integrated processor. It provides complete computer vision with both image processing and image recognition. It collects data, like x- and y position as well as height and width of a chosen object in its field of view, and delivers it to a chosen controller. The image recognition uses a hue-based colour-filtering algorithm that calculates the colour and saturation of each RGB pixel in its field of view. The only demand of the object is that it needs to have a distinct colour for it be easily distinguished from its surroundings. The Pixy can recognise up to 7 different colours and track up to a hundred objects at the same time. It also processes the image at 50 frames per second which makes it capable of tracking an object moving as fast as, for example, a falling ball.[21][22]

2.6.4 Ultrasonic sensors

Ultrasonic sensors are based on the fact that the distance to an object can be calculated with help of the speed of sound (340 m/s). The distance can be collected when the object is in a known medium and by measuring the time it takes for a sonic pulse to be reflected by the object. By multiplying the time with the speed of sound, the distance can be calculated.[23]

3 METHOD AND IMPLEMENTATION

The scientific way to solve a problem is to make sure that the right methods are being used. This chapter describes how the project has gone from planning to modelling, simulation and choice of components and thereafter building and programming of the robot. The work has been done in parallel with a constantly iterative process to find the best solutions. All the work has been evaluated along the way and much have been redone after test and implementation on the product to reach solid solutions and to reduce impact on the environment.

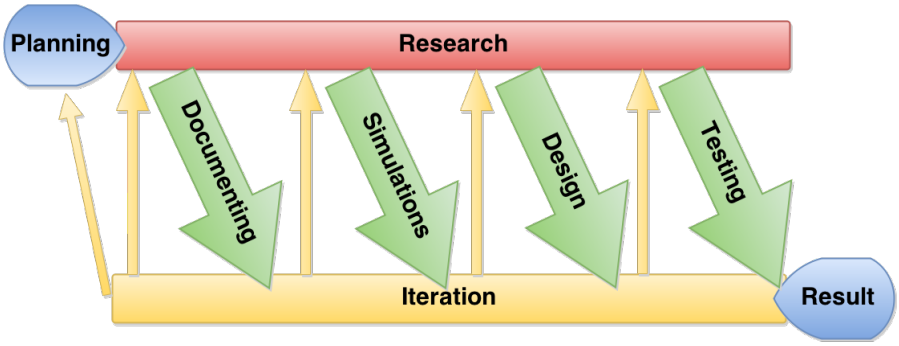


Figure 4: The flowchart and iteration process of the project.

3.1 Modelling and Simulation

The task of finding a suitable mathematical model has been an arduous one. The trick has been to balance between simplifying the model using relevant assumptions while maintaining the accuracy of the model. Difficulties with implementation of the initial model justified the transition to a more basic model of system and control regulator at a late point in the project time-line. Both models and control strategies are described in this chapter.

3.1.1 First model

The first mathematical model chosen for this project was derived using Newtonian mechanics with the following assumptions:[6]

- Two point masses represent the robot's ideal rigid body and wheel.
- The inner friction of the DC motor is neglected.
- No-slip condition between wheels and ground.
- When balancing, the angular velocities of the wheels are close to zero.
- The mechanical system is much slower than the electrical system.
- No yaw around the z-axis. This assumption seems an odd choice when designing a robot which needs to be able to turn. For the intended implementation, however, it was deemed to be good enough for success while also lowering the complexity of the model.

The complete model of the system was derived by combining three models; the DC motors, the wheels and the robot body, see figure 5.

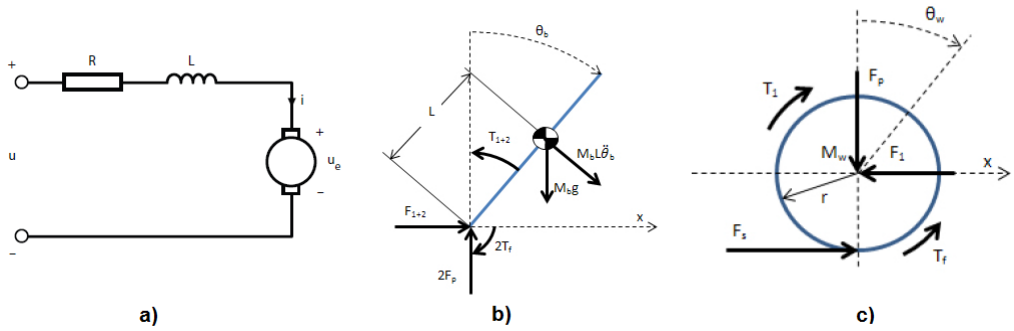


Figure 5: The models of a) the DC motors, b) the robot body and c) the wheels. Combined images from Bonafilia et al.[6]

The resulting model was (the parameters can be identified in table 1):

$$M_b\ddot{x}_b + M_bL\ddot{\theta}_bcos(\theta_b) = -2(M_w + \frac{J_w}{r^2})\ddot{x}_w - \frac{2K_eK_m}{r^2R}\dot{x}_w + \frac{2K_eK_m}{rR}\dot{\theta}_b + \frac{2K_m}{rR}u - \frac{2T_f}{r} \tag{8}$$

$$(J_b + M_bL^2)\ddot{\theta}_b = M_bgLsin(\theta) - M_b\ddot{x}_bLcos(\theta_b) + 2T_f - (T_1 + T_2) \tag{9}$$

The system dynamics could be represented by a nonlinear state-space model, $\dot{x} = f(x, u)$, where u is the voltage applied to the motors and x is the state vector of the system. Choosing the state variables in the state vector to be position, x, velocity, v, pitch angle, θ and angular velocity, $\dot{\theta}$, the state vector became:

$$\dot{x} = \begin{bmatrix} x & v & \theta & \dot{\theta} \end{bmatrix}^T \tag{10}$$

To be able to control the system by implementing basic controllers, the model was linearised.[9] Taking the jacobians of the A and B matrices and evaluating them in a position when the robot is balanced upright ($x = v = \theta = \dot{\theta} = 0$), the linearised state model was obtained as $\dot{x} = Ax + Bu$ and $y = Cx + Du$, where:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \alpha & \beta & -r\alpha \\ 0 & 0 & 0 & 1 \\ 0 & \gamma & \delta & -r\gamma \end{bmatrix} \tag{11}$$

$$B = \begin{bmatrix} 0 & \alpha\epsilon & 0 & \gamma\epsilon \end{bmatrix}^T \tag{12}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{13}$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{14}$$

$$\alpha = \frac{2(R_b - K_eK_m)(M_bL^2 + M_brL + J_b)}{R(2(J_bJ_w + J_wL^2M_b + J_bM_wr^2 + L^2M_bM_wr^2) + J_bM_br^2)} \tag{15}$$

$$\beta = \frac{-L^2M_b^2gr^2}{J_b(2J_w + M_br^2 + 2M_wr^2) + 2J_wL^2M_b + 2L^2M_bM_wr^2} \tag{16}$$

$$\gamma = \frac{-2(R_b - K_eK_m)(2J_w + M_br^2 + 2M_wr^2 + LM_br)}{Rr(2(J_bJ_w + J_wL^2M_b + J_bM_wr^2 + L^2M_bM_wr^2) + J_bM_br^2)} \tag{17}$$

$$\delta = \frac{LM_b g(2J_w + M_b r^2 + 2M_w r^2)}{2J_b J_w + 2J_w L^2 M_b + J_b M_b r^2 + 2J_b M_w r^2 + 2L^2 M_b M_w r^2} \tag{18}$$

$$\epsilon = \frac{K_m r}{R_b - K_e K_m} \tag{19}$$

Table 1: List of first model parameters.

Parameter	Value	Description
M_b	1.5	Mass of the robot $[kg]$
M_w	0.0227	Mass of the wheels $[kg]$
J_b	0.0032	Moment of inertia about center of mass $[kgm^2]$
r	0.045	Radius of wheels $[m]$
J_w	4.593e-05	Moment of inertia for the wheels $[kgm^2]$
L_m	0.08	Distance from wheel to center of mass $[m]$
K_e	0.573	EMF constant $[\frac{Vs}{rad}]$
K_m	0.240	Motor constant $[\frac{Nm}{A}]$
R	3.3	Motor resistance $[\Omega]$
b	0.002	Viscous friction constant $[\frac{Nms}{rad}]$
g	9.82	Gravitational constant $[\frac{m}{s^2}]$

The parameter values in table 1 were measured off of the constructed robot where possible, or approximated with calculations where measurements were difficult to perform. The inertia values were calculated by approximating the wheels to be solid cylinders rotating around their centres and the robot body to be a solid block rotating around the short side. The motor constant, K_m was approximated by taking the stall torque of the motors divided with the stall current of the motors and the EMF constant, K_e, was approximated by dividing the input voltage of the motors with the maximum speed of the motors. The viscous friction constant was assumed to be equal to the one in the model. The rest of the values were measured.

Control strategy
When all the parameter values had been the determined, they were used to calculate the linear system matrices. Without any control regulation the system was unstable, as any inverted pendulum would be. This was affirmed by examining the eigenvalues of the A matrix and observing that the system had unstable poles. The controllability of the system was also examined, to make sure that the system could be controlled. The resulting controllability matrix had the dimensions 4x4 and the rank 4 which inferred that the system was controllable.

Since the system was planned to be MIMO and since the robot would be able to measure all state variables in real time, a full state feedback controller was applied to control the system, see figure 6. This entailed using LQR to find the correct weighting coefficient, K, as well as the pre-compensation coefficient, N.

The controller stabilised the system and allowed for reference signals to be applied for the system to track. Position and angle were chosen as reference signals. Position was needed as a reference to control the movement of the robot and the angle reference was constantly set to 0 so that the robot would maintain its upright angle.

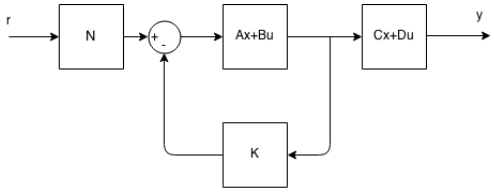


Figure 6: Conceptual block diagram of state feedback control.

A simulation of the system was then constructed to determine whether the model and the results of the control calculations were sufficiently accurate. The simulation model was constructed in Matlab/Simulink, see figure 7. It allowed for a rapid test-iteration where different reference signals and disturbance signals could be applied to the model.

A lot of effort was put into the iterative process of finding suitable values for the Q and R matrices of the LQR. The goal was for the robot to balance upright while maintaining its longitudinal position. The trick was to find the correct balance between the weighting on the x variable and the θ variable so that the specified requirements for the final robot could be achieved.

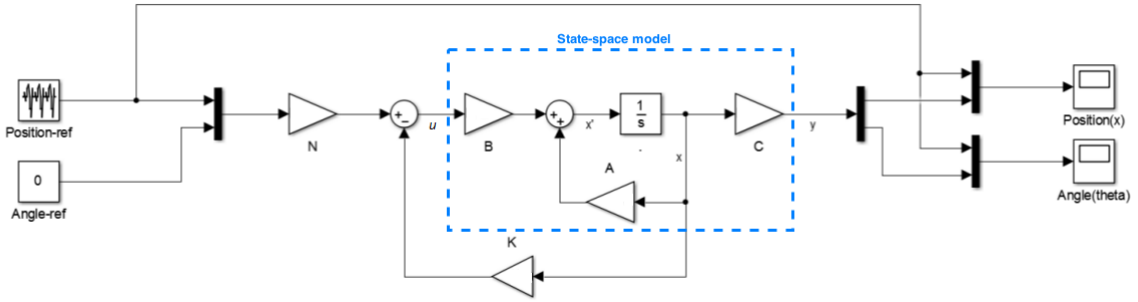


Figure 7: Simulation block diagram made in Matlab/Simulink.

Pole placement
The LQR problem comes down to choosing the correct Q and R matrices, the math takes care of the rest. While the workload may be reduced by having the Matlab command find the proper weighting parameters, the engineer's understanding of the controller is equally reduced. That is why, when the correct parameters for successful control could not be found using LQR, pole placement was used instead. The main issue when designing the controller using pole placement was to place the poles far enough to the left in the left hand plane of the complex plane to make the system stable and robust while making sure that the control signal did not exceed the maximum voltage output of the physical system.

3.1.2 Second model

The need for an optional mathematical model arose when the first model proved difficult to implement in practice using the constructed robot. The second model was designed using an existing robot whose body more closely resembled the body of the constructed robot. The idea was that implementation of this model would not be as sensitive to model errors as the first one. The second mathematical model chosen for this project was also derived using Newtonian mechanics [24] with the additional, simplifying assumptions:

- The dynamics of the DC motor is fast, which means that the inductance of the motors can be neglected.
- The differential speed of the wheels is 0 at all times.

The resulting nonlinear model was:

$$(m+M)\dot{v} - mdcos(\theta)\ddot{\theta} + mdsin(\theta)\dot{\theta}^2 = \frac{K_m}{rR_a}u \tag{20}$$

$$cos(\theta)\dot{v} - d\ddot{\theta} + gsin(\theta) = 0 \tag{21}$$

Table 2: List of second model parameters.

Parameter	Value	Description
m	0.15	Mass of the battery + shelf $[kg]$
M	1.35	Mass of the rest of the robot $[kg]$
d	0.18	Distance from rotational axis to battery $[m]$
r	0.045	Radius of wheels $[m]$
R_a	3.3	Motor resistance $[\Omega]$
K_e	0.573	EMF constant $[\frac{Vs}{rad}]$
K_m	0.240	Motor constant $[\frac{Nm}{A}]$

which could be represented as a state-space model, linearised around the upright position of the pendulum:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{g}{d}(1+\frac{m}{M}) & 0 & 0 \\ \frac{gm}{M} & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{K_m}{rR_aMd} \\ \frac{K_m}{rR_aM} \end{bmatrix} u \tag{22}$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ v \end{bmatrix} \tag{23}$$

where the state variables are pitch angle, θ, angular velocity, $\dot{\theta}$ and velocity, v.

Control strategy
To simplify the control problem further, the controller for the second model was chosen to be a PID controller which only controlled the angle of the robot, thereby reducing the system to SISO. Since this would mean that the robot theoretically could balance upright while travelling along the x-axis, an optional PI controller was added to control the robot velocity. This would be implemented in the manner of a cascade controller with the angle controller as the faster, inner feedback-loop and the velocity controller as the outer loop, see figure 8.

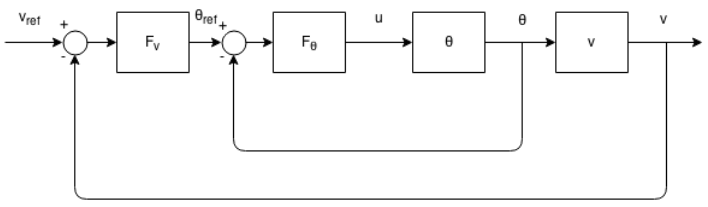


Figure 8: Block diagram over cascade controller.

The reasoning behind the choice of controllers was that the angle control needs to be fast and accurate and the velocity control can be much more slow and steady. To that end, a PID controller for the angle was necessary because of the need for fast, but smooth, response times and the need for eliminating the steady-state error.[24] For controlling the velocity, a PI controller would be sufficient since there is no need for the dampening effect of the D term on such a slow control effort. Similar to the previous control strategies, most of the time put into designing a PID

controller is used iterating coefficients trying to tune the controller into behaving in a suitable way. Firstly, the inner loop of the controller was tuned individually, before trying to tune the coupled cascade controller. The controller was implemented and tuned on the constructed robot without any previous simulations.

3.2 Robot construction

The robot had to be designed in a user friendly manner, in order to save time and avoid additional work in the future. A shelf design was chosen as a base for the rest of the construction, where components could easily be accessed and adjusted, see figure 9 and appendix A.3. A shelf system is also a good design for an inverted pendulum, due to the symmetry and the possibility to place a load (in this case the battery) on top of it. All choices of design and components have been made focusing on their ability to solve the problems and how user friendly they are. This has been a highly iterative process where many of the components depend on each other.

Figure 9: A CAD (computed aided design) picture of the robot model.

3.2.1 Arduino/Controller

The communication between the motors, sensors and Pixy is handled by an Arduino microcontroller. The Arduino model Mega ADK was chosen because of its memory capacity, that it had enough input/output ports and the fact that there was already one available from a previous project. Arduino controllers are easy to work with and there are a lot of Arduino compatible electrical components, such as different kinds of motor shields.

3.2.2 Pololu motors and motor shield

The stall torque and the no-load speed were two main factors regarding the motor that had to be taken into account in order to balance the robot. As research, a Balanduino robot, a fully functional two-wheeled robot very similar to the robot of this project, was examined.[25] This robot has a motor with a stall torque of close to 0.8 Nm and a no-load speed of 350 rpm. Since this proved to be sufficient stall torque to balance the Balanduino robot and more than enough speed, the minimum requirements for our robot were set to a stall torque of 0.8 Nm and a no-load speed of 150 rpm.

A geared DC motor (gear ratio 50:1) from Pololu fulfilled all the mentioned motor requirements.[26] The motor has a 1.2 Nm stall torque and a no-load speed of 200 rpm. It also came with Hall effect-

based quadrature encoders (magnetic encoders) which have a maximum resolution of 3200 counts per revolution of the gearbox's output shaft.

Together with the motors a dual motor driver shield was used to connect the motors to the Arduino.[27] The shield made it possible to control the motors with the Arduino while supplying them with power from the battery. This was necessary since the Arduino itself cannot supply enough power for the motors.

3.2.3 Sensors and camera

Sensors are needed to collect the required data for the robot such as tilt angle, the angular velocity of the tilt motion, the velocity and position of the robot's body as well as the position of the ball. The sensors used were motor encoders, an accelerometer, a gyroscope, the Pixy sensor (image processing camera) and an ultrasonic sensor.

The angle was determined by using the TinkerKit accelerometer and gyroscope, filtered through a complementary filter. TinkerKit sensors connect and communicate well with Arduino and there are also open source code libraries available for these components.

The position and velocity was determined with the motor encoders. By counting pulses from the encoder output, measuring the time and knowing the distance travelled between pulses, both the position and the velocity of the robot could be calculated.

To determine where the ball is located the robot uses the Pixy camera. It recognises different colour codes, so by using a predetermined ball the Pixy can easily recognise it and send back the coordinates of where it is located in the camera view. This information is used to align the robot towards the ball.

The ultrasonic sensors are used to determine the distance of the ball. By measuring the time for the echo to return and then divide the time by two (only one way) the distance can be calculated by multiplying this with the speed of sound (340 m/s). The chosen sensor has a detection range between 20 and 4000 mm which is good enough.[23]

3.2.4 LIPO Battery

The robot's power source had to be attached to the robot and work well with the symmetry and design of the robot, so that it would not interfere with the balancing. It also had to be sufficiently strong to run the two motors with 12 V for a long time, through active usage and many test runs. The battery chosen for this was a LiPo (lithium-polymer) battery pack.[28] It provides approximately 2.2 Ah and has a voltage of 11.1 V which is close enough to the recommended voltage for the motors. The Arduino board is recommended to be used with 7-12 V so an 11.1 V battery is sufficient.

A switch circuit was developed to be able to turn the power on and off without having to connect and disconnect cables. The circuit is comprised of a switch, a relay and a diode. The circuit can be seen in figure 10. The switch turns the relay on and off, which, if on, lead power to the motor shield. The relay is able to handle higher power than the switch and that is why the switch controls the power flow via the relay. The diode is put in the circuit as an extra precaution if high currents were to flow from the motors to the relay.

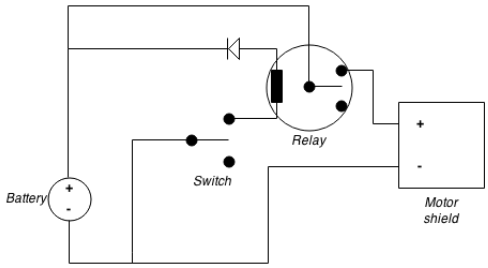


Figure 10: The switch circuit.

3.2.5 Construction of the robot body

The body of the robot was made from transparent thermoplastic that was sawn into three identical rectangular shapes. All the components were then mounted on the thermoplastic shelves with screws in holes that were drilled in the plastic, or in aluminium L-brackets that were mounted on the shelves. The shelves were assembled with threaded rods and nuts and the wheels were mounted with universal hubs and appurtenant brackets. Figure 11 show a conceptual schematic over the component connections.

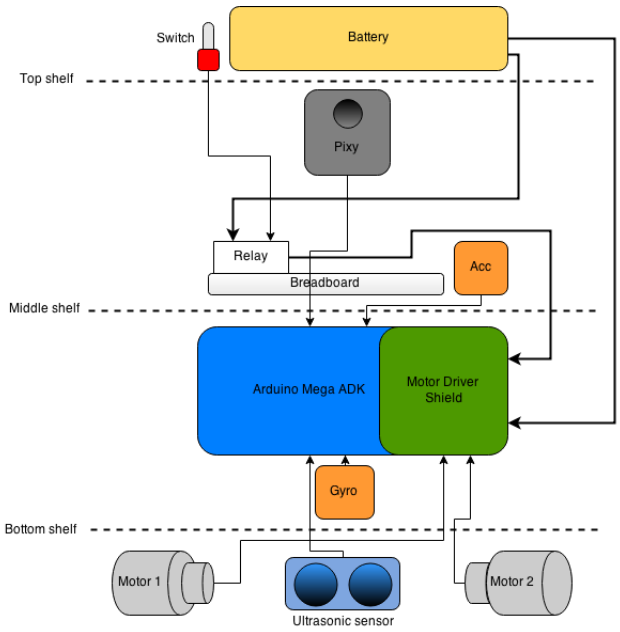


Figure 11: A conceptual schematic over component connections.

3.3 Software

The software is built on two main problems: the balancing problem and the path planning problem which includes object identification.

3.3.1 Balancing

The balancing software is based on the simple algorithm shown in figure 12. The program starts with reading all the values of the state variables (position, velocity, tilt angle and angular tilt velocity of the robot). These are needed to determine the control signal. The state variables are measured with the different sensors on the robot as stated above. The method of obtaining the tilt

angle and tilt angular velocity has already been mentioned. With the use of interrupt functions the pulses from the encoders can be counted and thus both the position and velocity of the robot is determined.

If the state variables are all zero, then the control signal will become zero as long as the reference signals (position and tilt angle) are zero, because there is no need for driving the motors if the robot is already balanced. However, if any of the state variables become non-zero there is a need for driving the motors to stabilise and to balance the robot. This makes the next step in the algorithm important, the calculation of the control signal. The control signal is calculated using a specified control method. Above, three different control strategies was chosen; LQR, pole placement and PI-PID cascade control. When the signal is determined it is applied to both motors in order to make the wheels turn equally and make the robot balance. This process is iterated as long as the robot should be balancing.

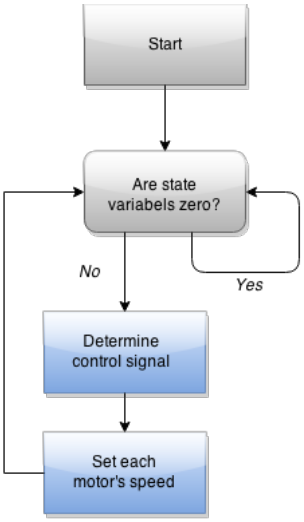


Figure 12: The conceptual design of the balancing software.

3.3.2 Object identification and path planning

To make a robot move autonomously there must be a pre-planned path for the robot to follow, a path that will change according to the tasks at hand. Planning a path for a robot to move towards an object, which in this case is a ball, will demand information about where the robot and the ball is located in relation to one another. The optional goal, to detect a football goal was never initiated since the project never got passed the first two main goals. All the information about the object was chosen to be gathered with the Pixy CMUcam5, which does all the image processing by itself, and the ultrasonic sensor. With the connection between the Arduino and the Pixy, the necessary information about an object, such as the x and y coordinates in the camera's two-dimensional view, can be retrieved and used. The ultrasonic sensor was connected to the Arduino to provide information about the distance of the ball. This information is crucial for path planning in order to make the robot approach the ball.

The conceptual design of the object identification and path-planning algorithm can be seen in figure 13. The algorithm starts with checking whether or not the ball is detected by the Pixy. If it is not, the robot should give the same speed but let the motors turn in different directions. This will cause a circular motion around its central axis and the robot will turn on the spot. It is assumed that the physical system is slow enough that the balancing during the turning motion will not be affected. The robot will only turn one complete revolution during which it will stop if the Pixy detects the ball and it will then align itself towards the ball. If, however, the robot completes the revolution around its axis and the ball is not detected, the robot will stop.

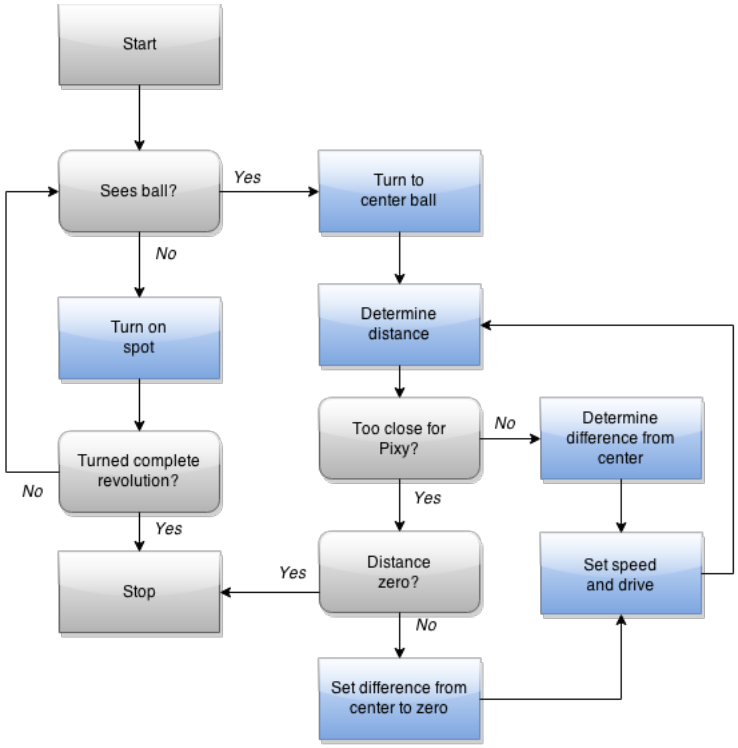


Figure 13: The conceptual design of the object identification and path planning software.

When a ball is detected, the program will continue with determining if the ball is in the middle of the camera view or not. The camera is placed parallel to the robot's wheel axis, so when the ball is placed in the middle of the camera's field-of-view the ball is also right in front of the robot. The middle of the camera view is the middle of the x-axis and the same as coordinate (160,y), see figure 14.

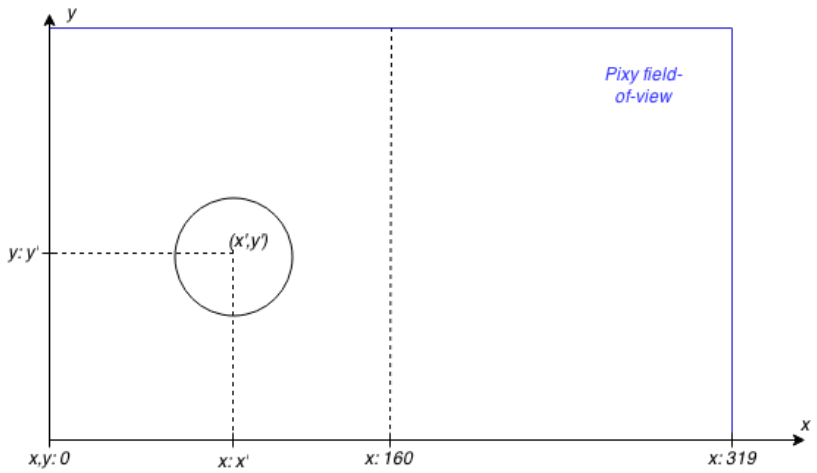


Figure 14: A graphical view of how the coordinates for a ball are in the field-of-view of the Pixy camera.

The coordinate can be retrieved by calling a function from the Arduino to the Pixy. The offset from the centre is determined by subtracting 160 from the x-coordinate of the ball. If the ball is not centred the function will first centre the ball once, after this initial turn the only alignment of the ball to the centre of the view will be handled by a P-controller which controls the turn during motion depending on the offset from the centre:

$$\omega = K_p(x - 160)/(320) \tag{24}$$

where ω is the angular velocity of the turn, K_p is the P-controller's gain (maximum angular velocity for turning) and the maximum difference is the maximum value of the points on the Pixy camera's x-axis, 320. K_p can be set by performing experiments to see how fast the robot could turn while still be able to balance.

The first alignment is crucial due to the fact that the ultrasonic sensor has to be able to determine the distance. The ultrasonic sensor has a limited "field-of-view" which makes it important to align the robot at the beginning so that correct values for the distance can be received. After that first alignment it is assumed that the ball will never change a lot from the centre position of the x-axis of the camera's view, which makes the P-controller on the turning a good enough alternative. The turning is, after the initial alignment, implemented using the formula's for v_l and v_r shown in section 2.3 by letting ω in the expressions for v_l and v_r be calculated using the P-controller above.

Thereafter, the distance between the ball and the robot is gathered from the ultrasonic sensor. If the distance is too small so that the Pixy can not see the ball anymore the code then checks whether or not the distance is zero. If the distance is zero the robot will stop. If, however, the distance is not zero, the difference between the x-coordinate of the ball and the centre of the camera view will be set to zero because it is assumed that the ball is so close that it does not matter anymore whether or not the robot changes its orientation a bit. The robot will then move towards the ball in a straight path. However, if the ball is not determined to be too close for the Pixy to see it, the difference between the x-coordinate and the centre of the camera view will be calculated and then the correct control signals for the motors will be set so that the robot moves towards the ball.

3.4 Sustainability aspects of the project

The project has aimed to be as environmentally friendly as possible and when there has been many possible solutions to a problem the chosen solution has always been the more sustainable one. The choice of materials for construction of the robot body have for example been steel and thermoplastics. They are both easy to work with (the less mistakes made, the less material is wasted) and easy to recycle.

Many of the parts, both mechanical and electrical, have been re-used from previous projects instead of bought new which has both saved money and reduced the impact on the environment. The assembled robot is easy to take apart. It is therefore easy to re-use the different parts and even recycle them when exhausted.

When buying the necessary products for the robot the environmental factor has always been taken into account. The battery of the robot is, for example, rechargeable and is such that if taken good care of it can be used for a number of future projects. The cables bought are such that there is no need for soldering. Instead they can be used over and over again so that the usage of both components and cables does not consume any new and unnecessary materials.

3.5 Verification

The amount of results that had to be verified through tests where limited since the number of definable results where quite few. First the simulation was verified through plots of relevant system responses that were analysed and interpreted. The motors capacity in terms of torque and no load speed could be read in the data sheet for the specified motors, and there was never a reason

to question that capacity. For the dimension and weight of the robot the verification was done with the help from scale and ruler. The lifespan of the robot and its ability to withstand falls also needed to be verified and that was done through testing at several occasions and an evaluation of the robot's condition at the end of the project. The verification of the budget was done last in a sum up of all the different expenses and by comparison to the given budget. For a complete list of all verification tests, both used and unused, see appendix A.2.

4 RESULTS

This chapter answers to goals of the project set in the purpose section. It covers all the results of the project, from the early simulation results to the construction, programming and final functions of the robot. It also covers the fulfilment of the requirements for the robot and a section about budget and costs.

4.1 First model

The parameters of the project robot was put into the first state-space model which yielded the state-space matrices:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -119.6 & -23.7 & 5.4 \\ 0 & 0 & 0 & 1 \\ 0 & 1321.8 & 314.5 & -59.5 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 9.8685 \\ 0 \\ -109.0467 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This model became the basis for the Matlab/Simulink simulation that was performed.

4.1.1 Simulation results of the first model

The simulation was a process of iteration, in terms of varying controller parameters, and resulted in a suggestion for the weighting coefficients:

$$K = \begin{bmatrix} -20.00 & -41.48 & -87.39 & -4.41 \end{bmatrix} \text{ and } N = \begin{bmatrix} -20.00 \end{bmatrix}$$

where the weighting on the angle variable is clearly larger than the weighting on the other variables. This follows from the LQR method where the appropriate Q and R matrices were found to be:

$$Q = \begin{bmatrix} 400 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 6000 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } R = \begin{bmatrix} 1 \end{bmatrix}$$

The values were used for the final verification of the simulation. First, a step function was applied as the system's position reference signal. Figure 15 shows the step response of the robot position and angle. The position step response is a well controlled, smooth motion without any considerable overshoot. The angle step response shows a balanced correction for the small deviation that occurs when the simulation changes position.

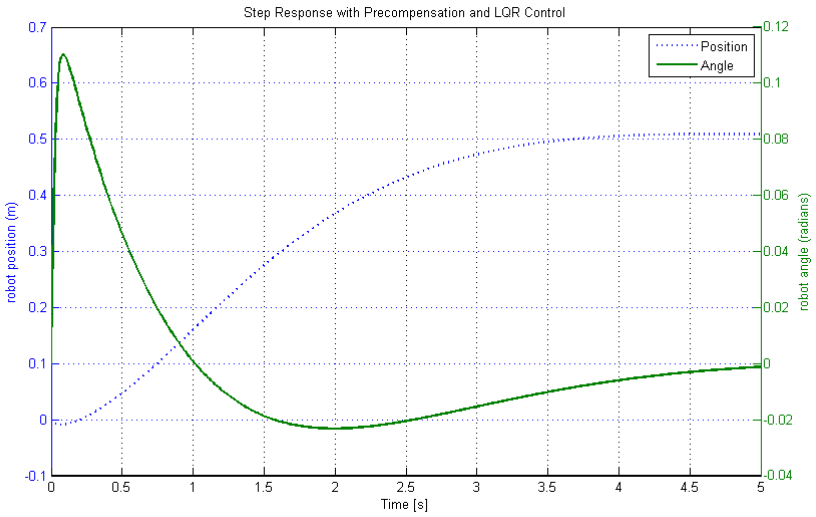


Figure 15: The system response for position and angle when subjected to a step function on the position.

Next, a variable position reference signal was applied to the system and the response was plotted. Figures 16 and 17 show the system's position and angle responses to the dynamic position reference. The position tracking was good and the pitch angle deviation from 0 was at most around 11 degrees which fell within the range of the defined requirements. Figure 18 shows how the control signal varies when tracking the position reference. Large movements of the robot result in control signals larger than the physical system will be able to supply. However, this only occurs when the simulated movement is much larger and faster than the physical system will need to be.

The results of the simulations suggested that the controlled model was accurate enough to move forward with testing on the physical system.

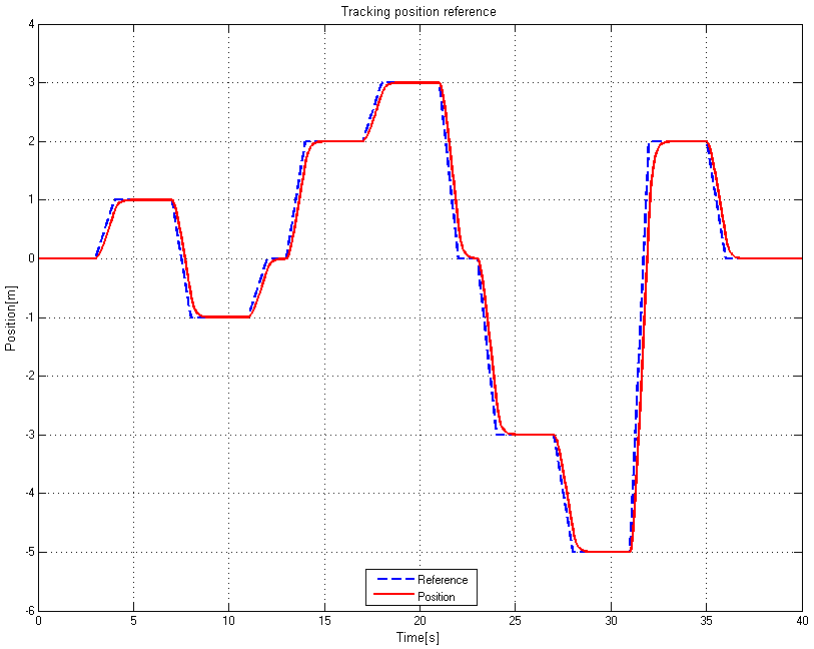


Figure 16: The system is tracking a reference position.

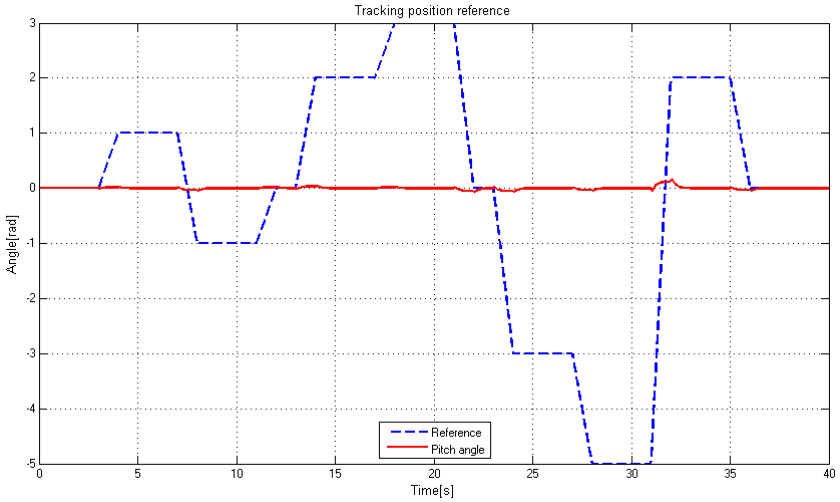


Figure 17: The system pitch angle response. (position reference)

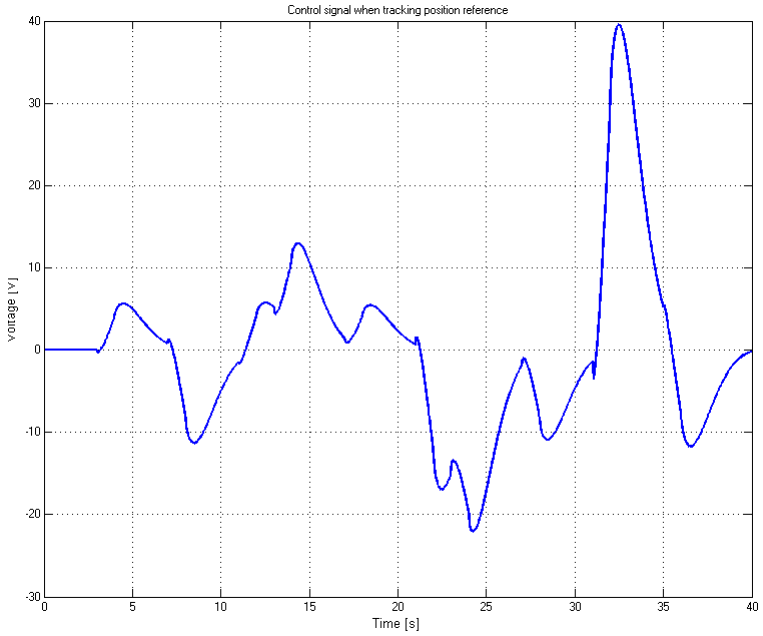


Figure 18: The system control signal. (position reference)

The practical implementation of this controller was not successful. The controller was not able to stabilise the robot for an extended amount of time. At best, the robot balanced back and forth a few times before overshooting and failing to compensate. This result was concluded after a large amount of calibration effort.

4.1.2 Pole placement

Using pole placement, a controller was found with the weighting coefficients:

$$K = \begin{bmatrix} -0.78 & -12.82 & -4.07 & -0.32 \end{bmatrix} \text{ and } N = \begin{bmatrix} -0.78 \end{bmatrix}$$

These were found using the poles $P_1 = -6 + i2, P_2 = -6 - i2, P_3 = -3 + i, P_4 = -3 + i$, see figure 19, which were determined through trial-and-error in Matlab. A step function was applied to the

controlled system and the response is plotted in figure 20. It was determined to be faster than the previous controller while sacrificing a little of the error margin on the angle variable. The response was still within the specifications.

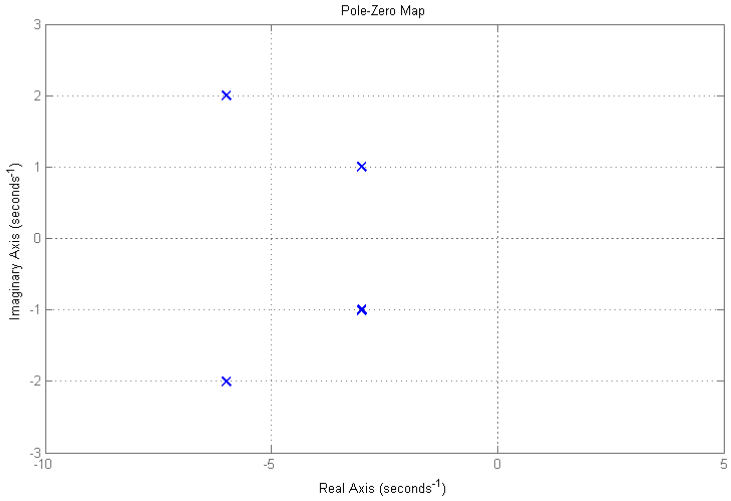


Figure 19: Pole placement of the controller.

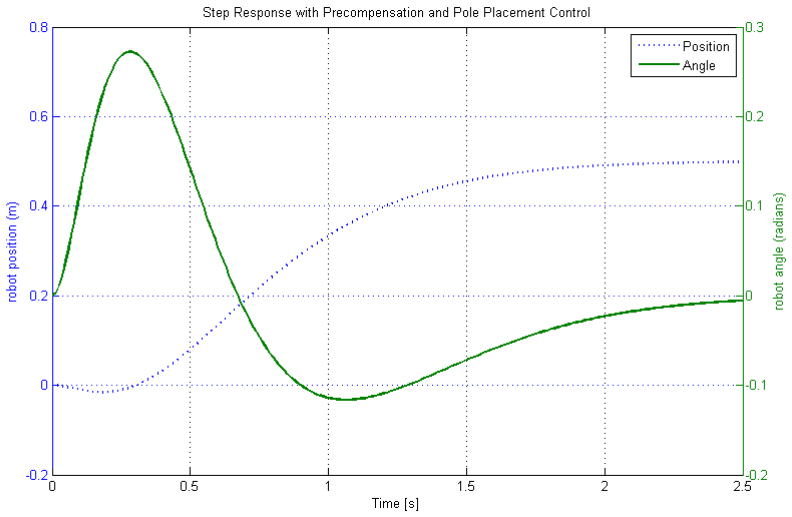


Figure 20: The system step response using pole placement.

The implementation of the controller using pole placement was not successful. The controller was not able to stabilise the entire system despite calibration efforts. However, the controller was able to stabilise the robot angle when the robot was firmly held by the wheels, which suggested that the control problem might be solved if the complexity of the model was reduced.

4.2 Second model

The parameters of the project robot was put into the second state-space model which yielded the state-space matrices:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 60.56 & 0 & 0 \\ 1.09 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 6.65 \\ 1.20 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

This model was verified only by evaluating its step response in Matlab. Figure 21 shows the response when applying a step to the robot angle. The settling time is faster than that of the previous model, the maximum error value is lower and the control effort is smoother. The PID parameters used were; $K_p = 60, K_i = 200, K_d = 4.5, T_f = 0.005$.

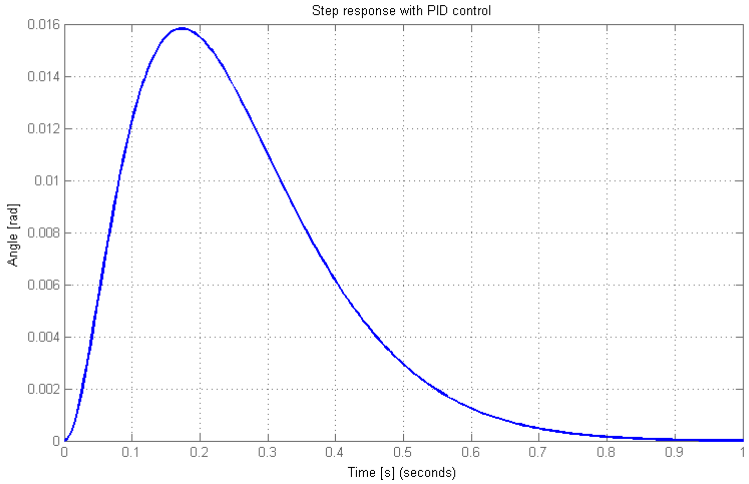


Figure 21: Step response with PID controller.

The implementation of the controller using PID control was not successful. The controller was not able to fully stabilise the system. There was, however, a significant improvement when compared to the previous model.

4.3 Construction and components design

The robot is built up of three shelves made out of transparent thermoplastics, see figure 22. These are assembled using four threaded rods with screw nuts on the upside and downside of each shelf to keep the construction stable and to keep the three shelves in the correct position. The distance between the shelves were decided according to which components were placed on each shelf and the height of those components, but in both cases the distance is about 120 mm. Space between the components is needed to avoid unnecessary heat development.

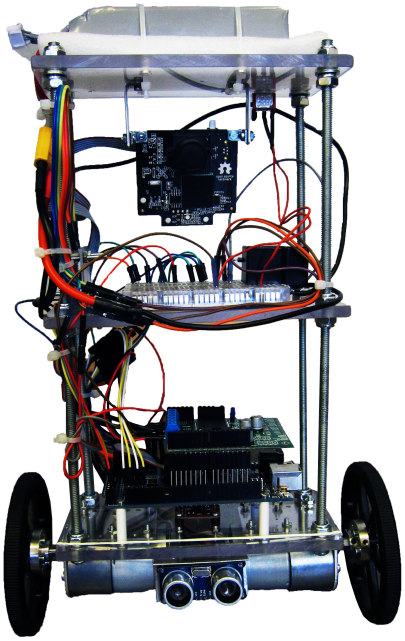


Figure 22: The full construction of the robot.

The location of the components are as follows:

Top shelf:
The power supplied to the robot comes from the battery mounted on the top shelf. A switch to turn the power on and off is also placed here. The Pixy camera for the image recognition is mounted under the shelf in a custom made gantry. The Pixy is mounted in a 30 degree angle relative to the z-axis of the robot which gives it a sufficient view of what is in front of the robot. For pictures of the top shelf see figures 23 and 26a.

Middle shelf:
The accelerometer which provides one part of the angle measurement is placed on the shelf with a handmade bracket. The bread board which is used for some of the couplings between the components is attached with the self-adhesive tape that comes with it. For picture of the middle shelf see figure 24.

Bottom shelf:
The gyroscope is placed on the shelf with screws and provides the other part of the angle measurement. The Arduino with the motor shield on top of it is placed with screws and distances in order to grant enough clearance above the gyroscope. All communication between all different components is done from here by the Arduino. On the underside of the shelf the motors are mounted with L-brackets. On the motor drive shafts, the wheels are connected with intermediary hubs. The encoders are mounted on the other side of the drive shafts. Between the motors, an ultrasonic sensor is mounted facing the robot movement direction. For pictures of the bottom shelf see figures 25 and 26b.

No waste material except the lost material from drilling was created during construction. The total weight of the robot is around 1.5 kg. That includes the framework of the robot and all the components mounted on the shelf's.

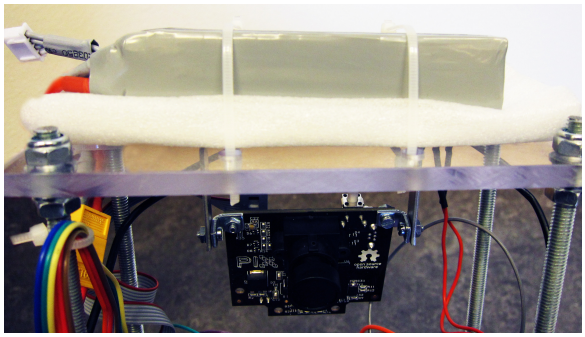


Figure 23: The top shelf of the robot.

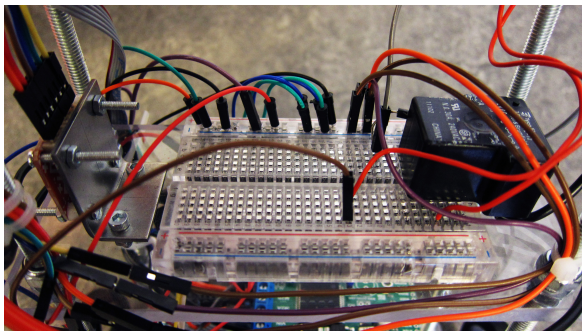


Figure 24: The middle shelf of the robot.

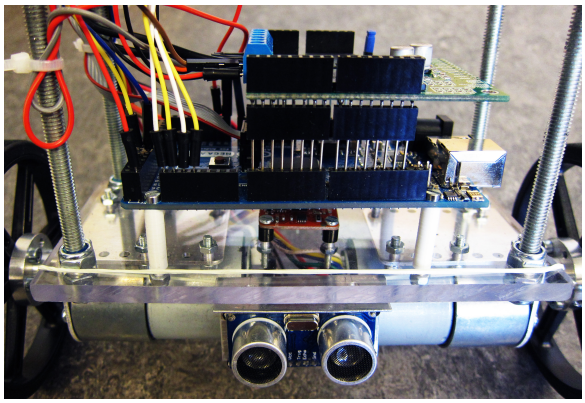
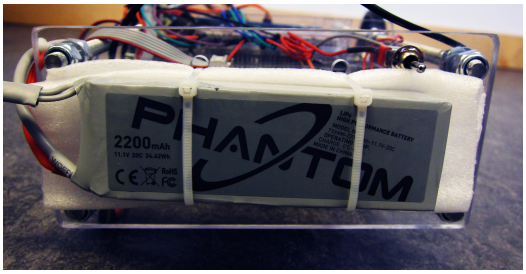
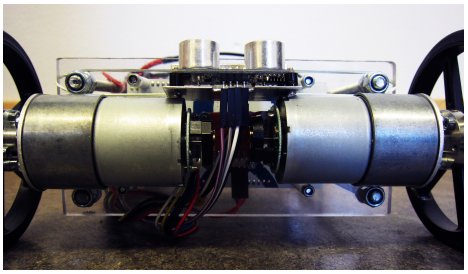


Figure 25: The bottom shelf of the robot.



(a) Top shelf from above.



(b) Bottom shelf from underneath.

Figure 26: The robot from above and under.

4.4 Programming design

The software developed for the robot is found in appendix A.4. The program is complete, with balancing and path planning algorithms as described in section 3.3 but not fully tested as the robot could not balance properly. Therefore, there is no confirmation of the path planning code. However, the balancing was tested, the program worked but it still did not balance properly.

4.5 Function of the robot

While watching the robot in action it is obvious for any observer that the robot cannot play football. All minor functions of the robot need to be taken into account to be able to understand what and where things might have gone wrong.

One of the most crucial and basic functions for the robot is the communication between components. All components managed to communicate with the Arduino, to both send and receive information. Accelerometer and gyroscope provided data that resulted in an accurate tilt angle for the robot and the motors responded by accelerating and correcting any deviation from the upright position. The ball was located by the camera and the robot responded to its location offset by making the movement for turning. But when the robot is standing by itself it cannot provide required stability and balance, and because of that it cannot be fully functional in any other area either.

4.6 Fulfilment of goals

None of the three goals for the project were fully accomplished. The first and most important was for the robot to balance on its own. As mentioned already there were satisfying results in the simulations. But the robot itself was never able to fully balance and therefore did not fulfil that goal. The goal of interacting with a ball did get as far as to track the ball and its position as well as act as intended with that information. Though it could not interact with the ball because of the balancing problem. The last and optional goal was never initiated because of the two unfulfilled main goals.

4.7 Fulfilment of requirements

The requirements for the robot, see appendix A.1, was set so that it would be able to clear all the problems set in the problem description, section 1.3. Many of the requirements also depend on each other which made it hard to fulfil many of them when the first ones were still pending. For the exact defined tests and results see appendix A.2.

The budget for all the components closed up at around 4700 SEK. This is below the limit of 5000 SEK and fulfils the budget requirement. The construction and design of the robot body passed all the set requirements, the weight of the robot did not exceed 2 kg and the outer dimensions did not exceed the set maximum dimensions of 200x300x400 mm [LxWxH]. The set requirements for the motor torque and no-load speed was also fulfilled with motors that have a torque of higher than 0.8 Nm and a no load speed higher than 150 rpm.

When it comes to the robot functions the most primary requirement was for it to be able to balance on its own. This requirement was not completely fulfilled even though the simulations results indicated that the system should be very stable and not sensitive to disturbances. Since the balancing was not fulfilled none of the requirement for the robot's balancing could be cleared. Because of this, the different requirements for the movement could not be tested in a correct way. This also resulted in unfulfilled requirements.

To be able to locate a ball according to colour when turning a whole revolution was the primary requirement for the object identification part. This could not be tested with the robot standing

in its upright position as the accelerometer had broke, instead, the motors were connected to the Arduino via the motor shield, and the Pixy camera and the ultrasonic sensor were connected to the Arduino. In this case the ball moved instead of the robot and the robot was able to slow down while closing up and stop the motors when the ball was located straight in front of it, which indicates that the algorithm for locating a ball was fully functional.

Last, the requirements for the robot was for it to withstand falls from an upright position and also to have a lifespan with regular use during the whole project (15 weeks), these requirements where also cleared and the robot is still as functional as it was to start with.

4.8 Budget

The final budget is found in figure 27. To make sure that the costs would not exceed the budget of 5000 SEK, the prices in the preliminary budget were slightly exaggerated and the total cost was limited to a maximum of 4000 SEK.

While working with electrical components there is always a risk that something breaks and needs to be replaced or that some components might not work as expected. Both these things occurred during the project. One accelerometer broke during testing and the first motor shield turned out to be too weak for the two motors. A few extra cables and screws were needed and two sets of wheels were ordered as well since the first ones were determined to be too small.

What had not been considered was the expensive shipping prices when extra components had to be ordered separately. Since extra expenses had been considered from the start the budget still held throughout the whole project and the final amount of money spent was less then 4700 SEK. In the end the total amount of unused components were one pair of wheels and one motor shield.

Component	Description	Amount	Cost SEK/component	Total cost SEK
Motor	Pololu 50:1 Met GM 37Dx54L mm 64CRP	2	339,00	678,00
Motor shield	Arduino Motor Shield Rev 3	1	269,00	269,00
	Dual VNH5019 Mot Drv Shield	1	409,00	409,00
L-brackets	Pololu 37D Met. Gearm Br.Pair	1	69,00	69,00
Gyroscope	ARDUINO T000062 MODULE, GYROSCOPE, TINKERKIT	1	194,25	194,25
Accelerometer	ARDUINO T000020 MODULE, ACCELEROMETER, TINKERKIT	2	241,22	482,44
Ultrasonic sensor	HC-SR04 Ultrasonic Range Finder (from old project)	1	0,00	0,00
Battery pack	RC Batteripack (LiPo) DJI LiPo Stick 11.1 V 2200 mAh	1	299,00	299,00
Battery charger	LiPo- /LiFe charger e4 VOLTCRAFT e4 (SK-1000555) 3 A	1	199,00	199,00
Bread board	(From old project)	1	70,00	70,00
Cables	BUD INDUSTRIES BC-32625 JUMP WIRE BUNDLE, 65 WIRES	1	100,00	100,00
	Artkl: 87901, Kjell & Company	1	44,00	44,00
	XT60	2	67,00	134,00
Header list	(Pins for motor shield)	1	20,00	20,00
Safety bag (battery)	Modelcraft LiPo-Safety-Bag	1	129,00	129,00
Wheels	Pololu Wheel Pair 70x10mm Blk	1	79,00	79,00
	Pololu Wheel Pair 90x10mm Blk	1	89,00	89,00
Mounting hubs for wheel	Uni. Alu. Hub 6mm, M3, 2-Pack	1	69,00	69,00
Control unit	Arduino Mega ADK (from old project)	1	0,00	0,00
Camera	Pixy CMUcam5	1	650,00	650,00
Switch	"Vippströmbrytare" 5A/250V, Kjell & Company	1	40,00	40,00
Relay	(From old project)	1	0,00	0,00
Screws	(For the Arduino, Järnia)	7	-	10,00
			Shipping fees:	600,00
Budget: Project 'Robot football' 2015 Chalmers SSYX02-15-22				
			Total:	4 633,69

Figure 27: The total budget for the project

5 DISCUSSION

All the goals seemed reasonable in the beginning of the project but was not fulfilled. This shows that the goals were either set too high, that the project was not carried out in a profitable way or a combination of the two. The setup of the project, the cooperation within the group, aspects of what could have been done differently and also what resources and what support the project could have benefited from, is covered in this chapter.

5.1 Development, product and result

A common problem and one of the major problems in this project is the insufficient amount of time set aside for every task and unexpected time consuming aspects that appear along the way. In this project lack of experience and time have been the prominent hold-backs.

An early decision in the project was to use a mathematical model from another project instead of creating a new one. If the chosen model was not accurate enough, this early decision may have affected the final outcome in a way that was negative for the balancing of the robot. However creating a mathematical model ourselves was not a reasonable option since that demands deep knowledge and experience in modelling of non-linear systems. It would have taken far too much time from the rest of the project while still not assuring a better result.

Searching for a suitable model, creating simulations of a stable system with multiple inputs and outputs and finding the right components took longer time than expected. The construction of the robot could not be finished until all components had been delivered and this proved to take longer time than expected as well. Long delivery time caused problems again when more orders had to be placed. The time between the first order and the last delivery, when the robot could finally be completed and tested, was significantly longer than what had been planned for. This was also due to the high number of orders and broken components. It is hard to say what change would have saved more time in this regard. To make an even more thorough survey and minimise the need of additional component orders, or to make a more shallow survey and be able to place all the orders much sooner. The project would have benefited from making few larger orders instead of many smaller ones, both economically and for the purpose of saving time. In the end, though, it is hard to get everything perfect in the first order when there are as many different components as there have been in this project.

Before implementation could be done properly, there were a lot of general tests done to see that all components worked as expected and to get more familiar with their functions, settings etc. Time had been set aside for this but the understanding for the components was not gathered in a day but rather developed over a much longer time. What had not been considered was how slowly the rest of the project would proceed in the beginning due to the ongoing learning process. This may be a reason for why the project at this time advanced more slowly than expected.

When the robot was finally ready to try the first concept of the balancing algorithm implemented on the Arduino, it behaved as predicted. It was not even close to achieving balance. This was of course something that should be expected since it is not reasonable to get an advanced balancing algorithm to work on the first try. Something we did not consider though, was that the accelerometer would break down on the second day of trying. This event was very inconvenient since we had just started working on the main goal for the project and waiting for a new accelerometer to arrive delayed the balance work yet another week.

When the robot could not be tried further for the moment, the focus was instead put on the construction of the robot body and the code for both the balancing and the object identification of the robot, which were also necessary but quite hard to do without all components or a working robot. This work did minimise the time loss as much as was possible, but when the new accelerometer arrived there were only twenty days left until the deadline of the final report, where only nine of those days were scheduled for project work. Nine days were not enough time for us to fully evaluate

and tune a balancing system and at the same time evaluate the whole project and complete the final report.

The framework of the robot, or the robot body, was constructed within the given time frame and without any major mishaps and setbacks. The resulting framework is slightly twisted though since all the the screw nuts were tightened in the same direction. It still appears to be symmetric in weight but we cannot know for sure that this does not affect the balance. The same goes for the components. Even though they are light weighted and placed as symmetric as possible they might affect the balance. For an optimal design and to eliminate possible errors those things could have been considered and possibly corrected.

When programming there is always the time consuming task of troubleshooting. It is likely that we felt a bit stressed when most of the programming was done, but had the programming been carried out a bit more slowly and with proper commentary, it might have saved some time spent on troubleshooting and would perhaps have created a more efficient program. Another improvement that could have been done for the code is timing it to all components and to the control algorithm. This is also a complex and time demanding thing to do but it seems to be done for many other balancing robots and it would most likely improve stability.

Results from the simulation of the chosen model presented data that confirmed a stable system that could withstand disturbance when the LQR controller was applied to it. But the variables that worked perfectly in the simulation did not work on the robot. Of course tuning always has to be done when transferring results from a simplified simulation to a more complex reality. Experiments had to be done to learn how the different weighting coefficients affected the robot's movement pattern and on that basis the variables were adjusted in order to achieve balance for the robot. But even though all remaining time focused on the tuning the robot never became completely stable.

LQR is sensitive to changes and useful for complex systems like ours. But without sufficient knowledge it is very hard to use since there are many complex parameters that can vary and they all depend on each other. To tune a system like this is very hard and time demanding. Since the tuning was done as the final step of the process the understanding of how hard it was came rather late. If the knowledge of that had come clear earlier the attempt for another method might have come sooner or from the beginning.

The second model, a more simple cascade PID-controller, was as mentioned introduced rather late in the project to replace the LQR-controller. This since the LQR proved a bit too complex in its tuning and also since the mathematical model for it was too different from the actual robot. The robot construction that was modelled for the PID control was similar to the project robot and the tuning for it was simpler since there were not as many variables. The knowledge within the group about PID controllers were also a bit more deep and thorough.

The implementation of the PID controller presented similar results as the LQR, and the robot was not able to fully balance on its own. Since these results were accomplished during very little time compared to the tests with the LQR the robot could probably have achieved a better balance with the cascade PID controller if there had been more time to work with it.

The lack of desired results makes the verification part of the project quite short. Many of the requirements for the robot and hence the verifications, depended on the balancing to work. The results that could be verified though are very reliable since they are mostly gathered from data that never required questioning.

Many of the set backs had been considered in the planing and time had been set aside. What is clear now is that this time was not enough and that there should also have been more room within the project for unexpected events.

5.2 Process and project in general

The setup of a project and a project group as well as how to go through with the project, can ad advantages and restraints to both results and to the project process. The structure of a project is therefore very important to evaluate.

The scope of this project was very large including many different areas like modelling, simulation, construction, controlling, object identification and path planning for the robot. It would therefore probably be enough for a much larger project like a master's thesis. If the focus would have been set on a smaller area such as only the balancing or only the football playing by image-recognition and path-finding, it would probably have been more likely for this project to fulfil its goals.

Simple balancing methods with few variables were not compatible with the more complex methods like LQR, where completely new mathematical models had to be made. The more complex methods were however crucial for making the robot fully autonomous and capable of performing all required movements. This project skipped the basic models and developed a complex control system with LQR, in hope of getting all the desired functions to work at once. Without enough experience that task proved to be too complex. Had this project only focused on the balancing part, and not on building a robot or playing football, there would have been more time to develop both a basic balancing system and a more complex one and to gain more knowledge about control methods for balancing.

The project group only consisted of four members and the project had an upper limit of six students. This could be a factor to why the project seemed so large and why there never seemed to be enough time. But to be only four group members had its advantages in some ways. We could work very closely all four and everyone knew what was going on within the project at all times. Also, it probably helped us reach consensus while discussing and making decisions. In the writing process it was easy to get everyone involved, to get a good flow and to get everyone likewise minded through the whole text.

The project work and collaboration over all worked very well and the project progressed and pushed on the whole time with a relatively constant workload. Something that could have been done differently in terms of the whole project and the setup would be the beginning. It was quite hard to know where to put the limits, where to start and what to do. This gave the project a slow start and it took time to get to the real production and executive part. The slow start did probably affect the outcome even though a substantial amount of time was put into the project.

It was hard to gain deep knowledge and understanding since there were so many fields, few group members and not sufficient time for all that to be done. A large reduction of the project description, in the beginning of the project, and more constraints on what parts of the project to investigate would probably have been the best solution to better reach our goals. A meeting for this could have been requested with the examiner and the supervisor at the beginning of the project. It is likely that this would have provided a better and more efficient start as well. The project has however been very exciting and instructive with all the different fields that had to be examined and the group has communicated and cooperated more then well along the way.

5.3 Further development

To continue this work and to fulfil the goals there are two different ways to begin. One is to take a few steps back and start with evaluating the mathematical model more closely and to gather even more knowledge about the most simple balancing systems. In this way the work can go on and after a while reach more complex and interesting control systems and in that way more relevant results and functions. This is, however, close to redoing the whole project. Another would be to evaluate the robot as it is right now and see how the current modelling, programming, construction and tuning could be improved. It might be that it is quite close to reach stability, but there might also be some small error that cannot be fixed by minor changes.

When the robot is functioning as expected and the goals of the project have been fulfilled, there are plenty of ways to go from there. To make it function together with other similar robot and actually make a full football team could be one way. Another way could be to develop the path planning part for the robot to make it move smoother toward the ball and even catch a ball in movement. This part could also involve for the robot to locate the ball and the goal and calculate how it should hit the ball to score, without moving the ball from its location before scoring. That would demand some quite advanced path planning.

6 CONCLUSION

The major reasons for not reaching the goals were the magnitude of the project and the high set goals, as well as the lack of experience about this kind of work. Controlling a modelled ideal system might seem easy enough, but that differs a great deal from controlling a real unstable system.

Even though the project proved to be bigger and harder than what was first believed and that the reality turned out to be more difficult to mathematically model and simulate than had been expected the project has been very rewarding. The learning gained from the process has been notable and a big realisation has been that the reality is not much like any school example but rather consist of trial-and-errors and is best solved through a lot of preparation and experience.

References

[1] FIRA, "Federation of international robot-soccer association," http://www.fira.net/main/, 2015-05-18.

[2] J. Berg, M. Demant, B. Kallman, and M. Nordstrom, "Robotfotball," http://publications.lib.chalmers.se/records/fulltext/200493/200493.pdf, 2015-05-18.

[3] Chalmers, "Chalmers policies," http://www.chalmers.se/en/about-chalmers/policies-and-rules/Pages/default.aspx, 2015-05-18.

[4] R. P. M. Chan, K. a. Stol, and C. R. Halkyard, "Review of modelling and control of two-wheeled robots, 2013," http://dx.doi.org/10.1016/j.arcontrol.2013.03.004, 2015-05-18.

[5] B. Mahler and J. Haase, "Mathematical model and control strategy of a two-wheeled self-balancing robot," *IECON Proceedings (Industrial Electronics Conference)*, 2013.

[6] B. Bonafilia, N. Gustafsson, P. Nyman, and S. Nilsson, "Self-balancing two-wheeled robot," http://sebastiannilsson.com/wp-content/uploads/2013/05/Self-balancing-two-wheeled-robot-report.pdf, 2015-05-18.

[7] J. R. Leigh, *Control Theory - A Guided Tour*. Institution of Engineering and Technology, 3 ed., 2012.

[8] T. Glad and L. Ljung, *Control Theory: multivariable and nonlinear methods*. Taylor and Francis Ltd., 1 ed., 2000.

[9] B. Thomas, *Modern Reglerteknik*. Liber AB, 4 ed., 2008.

[10] E. L. Hall, "Computer image processing and recognition, 1979," https://books.google.se/books?hl=sv&lr=&id=XEsB7FSIegC&oi=fnd&pg=PP1&dq=importance+of+image+recognition&ots=hBHdov59Bu&sig=sPU1qZaOteIqsT2L29x8nUFQYk#v=onepage&q=importance%20of%20image%20recognition&f=false, 2015-05-18.

[11] G. Novak and M. Seyr, "Simple path planning algorithm for two-wheeled differentially driven (2wdd) soccer robots, 2004," http://tinyphoon.com/rainbow/_tinyphoon/Documents/SimplePathPlanning.pdf, 2015-05-18.

[12] F. C. Vieira, "Position and orientation control of a two-wheeled differentially driven nonholonomic mobile robot, 2004," http://www.dca.ufrn.br/ adelardo/artigos/ICINCO04a.pdf, 2015-05-18.

[13] Arduino, "Arduino mega adk," http://arduino.cc/en/Main/ArduinoBoardMegaADK, 2015-05-18.

[14] Minebea, "Brushed dc motors," http://www.nmbtc.com/brush-dc-motors/, 2015-05-18.

[15] "WapCaplet", ""electric motor cycle 3"," http://en.wikipedia.org/wiki/Image:Electric_motor_cycle_3.png, 2015-05-19.

[16] Dynapar, "Dynapar technology," http://www.dynapar.com/Technology/Encoder_Basics/Motor_Encoders/, 2015-05-18.

[17] Pololu, "50:1 metal gearmotor 37dx54l mm with 64 cpr encoder," https://www.pololu.com/product/1444, 2015-05-18.

[18] D. S. Nyce, *Linear position sensors: theory and application*. John Wiley & Sons, Inc., 1 ed., 2004.

[19] B. Academic", "Accelerometer," http://academic.eb.com/EBchecked/topic/2859/accelerometer, 2015-05-18.

[20] H. G. Min and E. T. Jeung, "Complementary filter design for angle estimation using mems accelerometer and gyroscope," http://www.academia.edu/6261055/Complementary_Filter_Design_for_Angle_Estimation_using_MEMS_Accelerometer_and_Gyroscope ,2015.

[21] CMUcam, http://cmucam.org/projects/cmucam5, 2015-05-18.

[22] Kickstarter, "Pixy (cmucam5): a fast, easy-to-use vision sensor," https://www.kickstarter.com/projects/254449872/pixy-cmucam5-a-fast-easy-to-use-vision-sensor/description, 2015-05-18.

[23] E. freaks", "Ultrasonic ranging module hc - sr04," http://www.micropik.com/PDF/HCSR04.pdf, 2015-05-18.

[24] Chalmers, "Reglerteknik M Inlämningsuppgift 3,"

[25] Balanduino, "Balanduino," http://www.balanduino.net/about-balanduino, 2015-05-18.

[26] Pololu, "50:1 metal gearmotor 37dx54l mm with 64 cpr encoder, specifications," https://www.pololu.com/product/1444/specs, 2015-05-18.

[27] Pololu, "Pololu dual vnh5019 motor driver shield for arduino," https://www.pololu.com/product/2507, 2015-05-18.

[28] Conrad, "Rc batteripack (lipo) dji lipo stick 11.1 v 2200 mah," http://www.conrad.se/RC-Batteripack-%28LiPo%29-DJI-LiPo-Stick-11.1-V-2200-mAh.htm?websale8=conrad-swe&pi=1297839&ci=SHOP_AREA_211999_1209075, 2015-05-18.

A Appendix

A.1 List of requirements

Requirements	Quantitative measurements
Functions:	
Stability:	
Be able to balance on it's own	Full automation
Straighten up from a maximun angle	25 degrees
Be able to stand still	Within 10cm
Settling time (time between start and full balance)	3 s
Movement	
Go forward and backwards	maximum average angle deviation of 5 degrees
No load speed	At least 150rpm
Stall torque	At least 0,8Nm
Rotate a whole revolution	Within a circle with diameter that is 5 cm more then its own width
Advance while changing course	turn 90 degrees on a 1 m long run
Push the ball forwards	Transport a ball from A to B
Stay within a specific area	2x3 m
Location	
Locate a kind of ball according to size/colour	Should be located after a full revolution
Strength:	
Withstand minor falls	Tipping over from upright position
Lifespan with regular use in indoor temterature that exceedes the project deadline	10h use/ week for 15 weeks
Size:	
Low weight	Maximum of 2 kg
Dimensions that facilitates the rest of the work	Maximum 400x300x200 mm[h*w*d]
Costs:	
Project budget	Maximum cost of 5000 SEK + 2000 SEK for building the body

Requests	Quantitative measurements
Straighten up from a maximun angle	30 degrees
Be able to stand still	Within 5cm
Locate the goal according to size/colour	Should be located after a full revolution
Score the ball in the goal	From the same distance as the width of the goal

Figure 28: The list of requirements of the finished robot.

A.2 Defined verifying test for the robot

Balance			
Requirements	**Request**	**Verification method**	**Result**
Be able to balance		The robot can balance on its own, without falling if started in upright position	Not passed
Straighten up from 25 degrees	Straighten up from 30 degrees	Let the robot be still and push it 25/30 degrees and se if it can straighten up	Not passed
Be able to stand still (maximum movement +-5 cm)	Be able to stand still (maximum movement +-2.5 cm)	*See requirement of settling time*	Not passed
Have a settling time of 3 seconds		Measure the deviation from still position after 3 seconds	Not passed
Movement			
Requirements	**Request**	**Verification method**	**Result**
Go forward and backwards with a maximum avarage angle deviation of 5 degrees		Measure the real path compared to the desired path for 1m of traveling	Not passed
No load speed at least 150 rpm		Datasheet of motors	Passed
A stall torque of at least 0.8 Nm		Datasheet of motors	Passed
Rotate a whole revolution within a circle with diameter 25 cm (5 cm more than it's own width)		Watch it rotate on a marked circle	Not passed
Be able to turn at least 90 degrees on a 1 m long run		Measure the degrees of turning after a 1m long run	Not passed
Push the ball forwards from point A to point B with a distence of 20cm between the points		Observe	Not passed
Stay within a specific area of 2x3 m		Observe	Not passed
	Score the ball in the goal from the same distance as the width of the goal	*Not relevant for testing*	Not passed
Object identification			
Requirements	**Request**	**Verification method**	**Result**
Locate a ball according to colour when turning a whole revolution	Locate the goal according to colour when turning a whole revolution	Place the robot and the ball on a specified area, described in the section boundaries. Start the robot and se if it locates the ball within a revolution	Not passed
Strength			
Requirements	**Request**	**Verification method**	**Result**
Withstand falls from upright position		Hold the robot in upright position and let it fall to the ground. After this the robot should still be fully funcional	Passed
Lifespan with regular use in indoor temterature that exceedes the project deadline, 10h use/ week for 15 weeks		Accomplished if the robot is operative on the last day of the project (19 may)	Passed
Size			
Requirements	**Request**	**Verification method**	**Result**
A maximum weight of 2 kg		Place the robot on a scale and read the value	Passed
A maximun size of 400x300x200 mm[h*w*d]		Measure the robots dimensions when assembled	Passed
Cost			
Requirements	**Request**	**Verification method**	**Result**
Don't exeed the project budget of 5000 SEK + 2000 SEK for building the body		Calculating and sum all the purchases	Passed

Figure 29: Defined test for verifying the results of the finished robot.

A.3 Layout

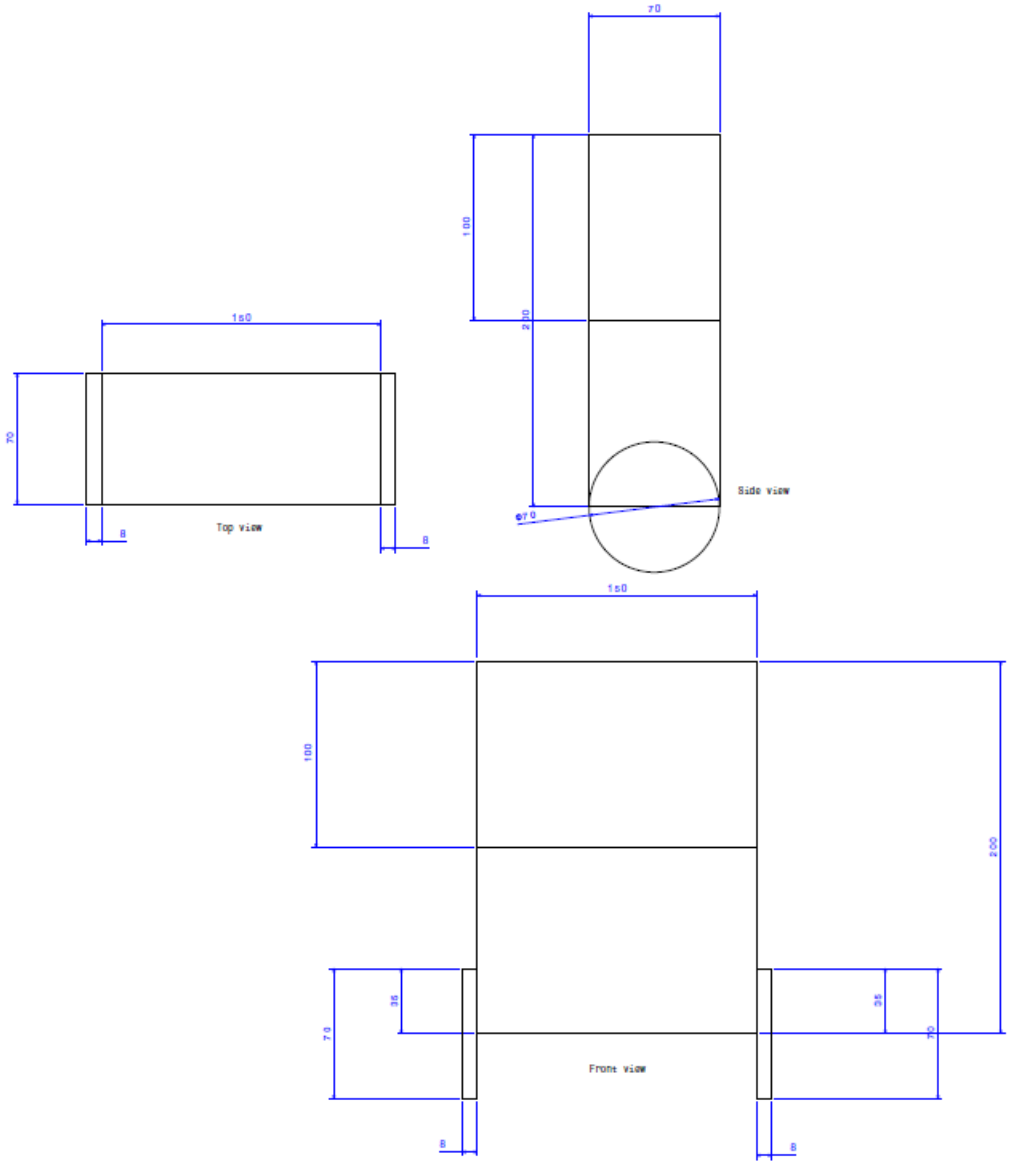


Figure 30: The layout of the robot.

A.4 Software code

```
#include <DualVNH5019MotorShield.h> // Motor driver library,
https://github.com/pololu/dual-vnh5019-motor-shield
#include <TinkerKit.h> // Tinkerkit sensor library,
https://github.com/TinkerKit/TinkerKit
#include <SPI.h>
//http://www.cmucam.org/projects/cmucam5/wiki/Hooking_up_Pixy_to_a_Microcontroller_
%28like_an_Arduino%29
#include <Pixy.h>
//http://www.cmucam.org/projects/cmucam5/wiki/Hooking_up_Pixy_to_a_Microcontroller_
%28like_an_Arduino%29

//* Pin assignments *//

// Accelerometer
#define accPinX I7 // Mega ADK pin A7
#define accPinY I8 // Mega ADK pin A8

// Gyroscope
#define gyroPinX I9 // Mega ADK pin A9
#define gyroPinY I5 // Mega ADK pin A5

// Motor encoders
#define intM1A 5 // Mega ADK pin D18
#define intM1B 4 // Mega ADK pin D19
#define intM2A 3 // Mega ADK pin D20
#define intM2B 2 // Mega ADK pin D21

// Ultrasound sensor
#define pingEcho 34 // Mega ADK pin D34
#define pingTrig 35 // Mega ADK pin D35

//* Global variables *//
volatile boolean dirM1 = true; // Direction of motor 1
volatile boolean dirM2 = true; // Direction of motor 2
volatile long cntrM1 = 0; // Encoder counter of motor 1 [counts]
volatile long cntrM2 = 0; // Encoder counter of motor 2 [counts]
volatile long cntr = 0;
volatile unsigned long time = 0; // Timer for gyroscope sampling [ms]
unsigned long oldTime = 0; // [ms]
long oldPos = 0; // [counts]
int diff = 0; // initial diff for diffSpeed.
double posR = 0.0; // initial position reference.
double previousGyroAngle = 0; // Variable for storing gyroAngle between function calls [deg]
double previousGyroRate = 0; // Variable for storing gyroRate between function calls [deg/s]
double angle = 0; // Tilt angle [deg]

// Cascade controller
// Outer PI loop
double r_v = 0.0; // Reference for velocity
double y_v = 0.0; // Velocity value from sensors
double e_v = 0.0; // Error value
double P_v = 0.0; // Part of controller (P-part)
double Kp_v = -0.0655335; // Calculate proportional regulator
double angleSet = 0.0; // Angle reference signal as output from velocity control.
double I_v = 0.0; // Part of controller (I-part)
double Ki_v = -0.043768; // Calculate integral regulator
double eold_v = 0.0; // Old error.

// Inner PID-loop
double r_a = 0.0; // Reference for angle
double y_a = 0.0; // Angle value from sensors
double e_a = 0.0; // Calculate error value
```

```
double P_a = 0.0; // Part of controller (P-part)
double Kp_a = 60.1; // Calculate proportional regulator
double D_a = 0.0; // Part of controller (D-part)
double Tf_a = 0.080; // Time constant for low-pass filter
double Kd_a = 0.5; // Calculate derivate regulator
double I_a = 0.0; // Part of controller (I-part)
double Ki_a = 0.0; // Calculate integral regulator
double eold_a = 0.0; // Old error.
int uold = 0;

// Cascade controller
boolean firstIteration = true; // First iteration of path planning algorithm.
const double sampleTime = 5; // How often do we sample gyro angle? [ms]
const double alpha = 0.90; // Time constant variable for complementary filter
const double accCalX = 538; // Accelerometer calibration variables for x and y
const double accCalY = 538;
const double gyroThresh = 2*PI*(180/PI); // Threshold for sampling [degrees/s].
const double k1 = -0.7746; // Weighting coeff. for state feedback controller (K)
const double k2 = -12.8184;
const double k3 = -4.0725;
const double k4 = 0.3173;
const double kr1 = -0.7746; // Precompensation coeff. for position reference signal (Nbar).
const double kr2 = 0;
const double pixyCal = 0.3; // Least distance where ball still visible for Pixy (measured
experimentally when Pixy mounted with angle 30 degrees towards highest shelf).
const int turnSpeed = 100; // Highest speed the robot can turn (only turn) and still manage to
balance, should be experimentally determined.


//* Objects *//
Pixy pixy; // Pixy object.
DualVNH5019MotorShield shield; // Motor shield object.
TKGyro gyro(gyroPinX,gyroPinY, TK_X4); // Gyroscope object
TKAccelerometer acc(accPinX,accPinY); // Accelerometer object


//*SETUP*//
void setup() {
  pinMode(19,INPUT); // We want to read from encoder M1B
  pinMode(21,INPUT); // And read from encoder M2B
  pinMode(pingTrig,OUTPUT); // Ultrasonic sensor trigger pin will “talk”
  pinMode(pingEcho,INPUT); // Ultrasonic sensor echo pin will “listen”
  attachInterrupt(intM1A,getEncM1A,RISING); // Attach the correct ISR to the correct pins and
interruptmodes
  attachInterrupt(intM2A,getEncM2A,RISING);
  gyro.calibrate(); // Calibrate gyro in start position (robot pitch angle=0)
  time = millis(); // Set time to current time since program started [ms].
  shield.init(); // Initialize motor driver object
  pixy.init(); // Initialize Pixy object
}

//*LOOP*//
void loop() {
  setAngle(getAccAngle(),getGyroAngle());
  setRobotSpeed(getDiffSpeed(diff, 320), getControlSignal(posR , 0, getRobotPos(),
getRobotSpeed(), angle, previousGyroRate));

  int search_for_ball = 1; // Set to 1 if the robot should turn around and start path planning
algorithm, 0 else. Should be determined by a physical switch circuit -> digitalRead()..

  switch(search_for_ball) {
   case 0:
```

```
    //Do not search for ball just balance.
    diff=0;
    posR=0.0;
    break;
   case 1:
    //Search for ball, path plan.
    if(pixy.getBlocks() == 0) { // Have not found ball.
     shield.setM1Speed(turnSpeed);
     shield.setM2Speed(turnSpeed);
     cntr = cntrM1;
     if(cntr<0) {
      cntr = -cntr;
     }
     while(cntrM1-cntr<3556) { // A complete revolution was determined to be 3556
counts/pulses from encoders.
      if(pixy.getBlocks()) { // Found object during revolution. Align then balance.
       while(getCenterDiff() != 0) { // Align robot towards ball. Assume only detectable object is a
ball.
        shield.setM1Speed(turnSpeed);
        shield.setM2Speed(turnSpeed);
       }
       diff=0;
       posR=0.0;
       break;
      }
     }
     if(cntr>=3556) { // Revolved an entire revolution, set motors to stop.
      shield.setM1Speed(0);
      shield.setM2Speed(0);
      delay(5000); // While motors are turned off, wait 5 seconds until someone can switch off
the power.
     }
    }
    else {
     if(firstIteration) {
      while(getCenterDiff() != 0) { // Align robot towards ball if first iteration to be sure that the
ultrasonic sensors can sense the distance properly. Assume only detectable object is a ball.
       shield.setM1Speed(turnSpeed);
       shield.setM2Speed(turnSpeed);
      }
      firstIteration=false; // Set to false as first iteration has been done.
     }
     posR = getDistance()/100.0; // Transform distance to [m].
     if(posR<=pixyCal) {
      if(posR==0.0) {
       shield.setM1Speed(0); // Turn off the motors.
       shield.setM2Speed(0);
       delay(5000); // Wait 5 seconds for someone to turn off the power.
      }
      else {
       diff = 0; // Robot near enough (Pixy can barely see ball), assume ball and robot aligned
enough.
      }
     }
     else {
      diff = getCenterDiff(); // Control the difference from center of view, if nonzero make the
robot turn.
     }
    }
    break;
   default:
    break;
```

```
 }
}

//* Function modules *//

// Get difference between x-coordinate and center of Pixy's 2D camera view.
int getCenterDiff() {
  int pixelDiff;
  if(pixy.getBlocks() == 0){
   pixelDiff = 0; // Return 0 if not found.
  }
  else {
   pixelDiff = (int)(pixy.blocks[0].x)-160; // Assume only one detectable object and it is a ball,
hence index 0 (there is no need to check for signature).
  }
  return pixelDiff; // Return difference.
}

// Get angle in [deg] from accelerometer (The sensor is mounted with x in pos. vertical direction
and y in pos. longitudinal direction)
double getAccAngle() {
  double accValX = analogRead(accPinX) - accCalX; // Read and compare X and Y to calibration
value (analogRead map voltage as points between 0-1023)
  double accValY = analogRead(accPinY) - accCalY;
  double accAngle = (atan2(accValY,accValX)+PI)*RAD_TO_DEG; // Calc. value of angle
  accAngle-=180; // Correction for upside down sensor
  return accAngle; // Return angle in [deg]
}

// Get angle in [deg] from gyroscope (The sensor is mounted flat with the x axis aligned with
wheel axis).
double getGyroAngle() {
  double gyroAngle;
  if (millis() - time > sampleTime){ // Take a sample every 'sampleTime' [ms].
   time = millis(); // Update time to current time [ms].
   double gyroRate = gyro.readXAxisRate(); // Read gyro and determine angle if
angular velocity is greater than gyroThresh [deg/s].
   if (gyroRate >= gyroThresh || gyroRate <= -gyroThresh){
    gyroAngle = ((previousGyroRate + gyroRate) * sampleTime) / 2000; //Integrate with
trapezoid method to find gyro angle in [deg]
   }
   else{
    gyroRate = 0;
    gyroAngle = previousGyroAngle; // If the rate has changed less than gyroThresh, the old
angle still holds.
   }
   previousGyroAngle = gyroAngle;
   previousGyroRate = gyroRate; // Save gyroRate for next iteration
  }
  return previousGyroAngle; // Return angle in [deg]
}

// Calculates the sensorfused angle using a complementary filter. Sets robot's pitch angle in
[deg].
void setAngle(double accAngle, double gyroAngle) {
  angle = (alpha*(angle+gyroAngle*(sampleTime/1000))) + ((1-alpha)*accAngle);
}

// Compares the last function call with the current and returns robot longitudinal speed [m/s].
double getRobotSpeed() {
  long newPos = cntrM1; // Determine current position [counts]
  unsigned long newTime = millis(); // Determine current time [ms]
```

```
  double travDist = (newPos-oldPos)*((2*PI*45)/1600); // Compare ref. cntr with last cntr and multiply with millimeter/cnt to find traveled distance [mm].
  double robotSpeed = travDist / ((newTime-oldTime)); // Calculate speed [m/s] using traveled distance [mm] and elapsed time [ms]
  oldPos = newPos; // Update reference position
  oldTime = newTime; // Update reference time
  return robotSpeed;
}

// Calculates how far from reference position in [m] the robot has travelled. Returns position.
double getRobotPos() {
  return cntrM1*((2*PI*0.045)/1600); // Wheel circumference / counts per revolution.
}

// Get the distance from object straight ahead of the robot using the ultrasonic sensor. Returns distance in [cm].
int getDistance() {
  int duration, distance;
  digitalWrite(pingTrig, LOW); // Make sure trigger pin is LOW
  delayMicroseconds(2);
  digitalWrite(pingTrig, HIGH); // Send an impulse to trigger ultrasound
  delayMicroseconds(10); // For 10 µs
  digitalWrite(pingTrig, LOW); // Set trigger LOW again
  duration = pulseIn(pingEcho, HIGH); // Sets duration to the echo pulse length in microseconds, from pingEcho-pin as the pulse is HIGH.
  distance = (duration/2)*340*(100/1000000); // Transform data into cm by multiplying the half of the time it took to receive echo (one way) with the speed of sound, transformed into [cm/µs].
  return distance; // Returns distance in [cm].
}

// Weights state variables and reference signals according to state feedback controller (LQR & Pole placement). Returns motor voltage [V].
int getControlSignal(double posRef, double angleRef, double robotSpeed, double robotPos, double angle, double previousGyroRate) {
  int u = (kr1*posRef+kr2*angleRef)-(k1*robotPos + k2*robotSpeed + k3*(angle*(PI/180)) + k4*(previousGyroRate*(PI/180)));
  return u;
}

// Returns motor voltage according to cascade PI-PID control.
int getControlSignalCascade() {
  int u;
  if (millis() - time > sampleTime) {
   time = millis();
   // Outer PI-loop
   r_v = 0; // Reference for velocity
   y_v = getRobotSpeed(); // Velocity value from sensors
   e_v = r_v - y_v; // Calculate error value
   P_v = Kp_v*e_v; // Calculate proportional regulator
   angleSet = P_v + I_v; // Determine total control signal
   I_v = I_v + Ki_v*sampleTime/1000*e_v; // Update integral part
   eold_v = e_v; // Remember error value from last sample.
   // Inner PID-loop
   r_a = angleSet; // Reference for angle
   y_a = angle*(PI/180); // Angle value from sensors
   e_a = r_a - y_a; // Calculate error value
   P_a = Kp_a * e_a; // Calculate proportional regulator
   D_a = (Tf_a/(Tf_a+sampleTime/1000000)) * D_a + (Kd_a/(Tf_a+sampleTime/1000000)) * (e_a – eold_a);
   u = P_a + I_a + D_a; // Determine total control signal
   I_a = I_a + Ki_a*sampleTime/1000000*e_a; // Update integral part
   eold_a = e_a; // Remember error value from last sample.
```

```
   uold=u;
  }
  return uold;
}

// Calculates differential speed needed for turning (P-controller). Returns differential speed.
double getDiffSpeed(int diff, int maxDiff) {
  //diff : essentially pixelDiff from Pixy & path planning code but is used to determine the speed of the turn.
  //maxDiff : the maximum diff can be, for example, if diff = pixelDiff then maxDiff should be 320 [320 points on Pixy's x-axis].
  double width = 0.2; // Distance between the robots wheels.
  double Kp = PI/6; // For example PI/6 rad/s max turn speed.
  double omega = Kp*(diff/maxDiff);
  double vDiff = omega*(width/2);
  return (vDiff/(Kp*(width/2)))*400; //[(points/vel)*vel]->[points]
}

// Calculates motor speed and sets differential speed for the two motors.
void setRobotSpeed(double diffSpeed, int controlSignal) {
  double motorSpeed = 400/11.1 * controlSignal; // Transform controlSignal [V] to motorSpeed [points] by multiplying the controlSignal [V] with maximum [points/V], points indicate the values of which the motor shield sets the motor speed.
  shield.setM1Speed((int)((motorSpeed+diffSpeed))); // Set the motor speeds as the same if diffSpeed = 0 but make them different (the same amount "different") if diffSpeed nonzero.
  shield.setM2Speed((int)(-(motorSpeed-diffSpeed)));
}

//* Interrupt routines *//
// Encoder interrupt routines. Tracks rotation of the wheel which is used for position and speed of the robot.
void getEncM1A() {
  // Read digitalpin 19 for encoder M1B. If it is HIGH when M1A triggers, the motor is moving backwards. And vice versa.
  if (digitalRead(19)){
   dirM1 = false; // false = Backwards
   cntrM1 = cntrM1 - 2; // Interrupt triggers on rising, so need to correct for both channels
  }
  else{
   dirM1 = true; // true = Forwards
   cntrM1 = cntrM1 + 2; // Interrupt triggers on rising, so need to correct for both channels
  }
}

void getEncM2A(){
  // Read digitalpin 21 for encoder M2B. If it is HIGH when M2A triggers, the motor is moving backwards. And vice versa.
  if (digitalRead(21)){
   dirM2 = false; // false = backwards
   cntrM2 = cntrM2 - 2; // Interrupt triggers on rising, so need to correct for both channels
  }
  else{
   dirM2 = true; // true = forwards
   cntrM2 = cntrM2 + 2; // Interrupt triggers on rising, so need to correct for both channels
 }
}
```