LUND UNIVERSITY

Projects in Automatic Control 2018

Cervin, Anton; Maggio, Martina; Soltesz, Kristian

2019

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Cervin, A., Maggio, M., & Soltesz, K. (Eds.) (2019). *Projects in Automatic Control 2018*. (Technical Reports TFRT-7659). Department of Automatic Control, Lund Institute of Technology, Lund University.

Total number of authors:
3

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Projects in Automatic Control 2018

Anton Cervin
Martina Maggio
Kristian Soltesz
Editors

Lund University

Department of Automatic Control

Technical Report TFRT-7659
ISSN 0280–5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset.
Lund 2019

Preface

The Department of Automatic Control at Lund University annually gives a project course in Automatic Control. The course is given at the advanced level and comprises 7.5 ECTS credits. In this course, the students work in small teams, to achieve a common goal. The projects typically involve a real-world estimation or control problem with relevance to industrial or other applications.

In this course, the students get an opportunity to explore implementational aspects of concepts they have learned in previous control systems courses. With a faculty member or doctoral student as advisor, the groups get to independently formulate an objective and a time plan. Subsequent activities typically involve modeling, controller design, implementation, documentation and verification. The students present their work through two feedback seminars, a demonstration session and a written report. The reports of the 2018 edition of the course are collected in this booklet. distr Each year, a jury consisting of the teaching staff rewards members of two groups with an award and small prize. In 2018, awards were handed out for “Best control engineering and demonstration” to Nicklas Norberg Persson, Dennis Dalenius, Per Josefsson and Daniel Johansson for their work on the project “Ball Catching Robot” and for “Best report and documentation” to Albert Anderberg, Josiah Wong and Rebeca Homssi for their work on the project “Autonomous Driving F1/10 Car”.

Doctoral students Marcus Greiff, Marcus Thelander-Andrén and Nils Vreman have served the course as project advisors. We would also like to thank our research engineers Leif Andersson, Pontus Andersson, Anders Blomdell and Anders Nilsson who have supported the groups throughout their projects. Finally, we would like to thank Mika Nishimura for her help with student registration and related matters.

To find out more about the course, please visit https://www.control.lth.se/education/engineering-program/frtn40-project-in-automatic-control/.

Lund, January 2018
Kristian Soltesz, on behalf of the teaching staff

Contents

Lego Segway **7**
Eriksson, Knutsson, Karlsson, Nilsson

Obstacle Avoidance using a LEGO Mindstorms EV3DEV vehicle **15**
Bladh, Grahovic, Rosicki, Tamilinas

Autonomous driving F1/10 car **25**
Homssi, Anderberg, Wong

UAV Control **31**
Green, Karlin, Stanger, Voigt

Autonomous parallel parking - LEGO truck and trailer **37**
Andrén, Hjertberg, Ståhlbom, Schyllert

Ball Catching Robot **53**
Johansson, Dalenius, Norborg Persson, Josefsson

CSTR **63**
Bolin, Franzon, Klint, Persson

Ball and Plate **71**
Artursson, Farmängen, Lundbladh, Pérez

Lego Segway

Robin Eriksson[1] **Axel Knutsson**[2] **Niklas Karlsson**[3] **Emma Nilsson**[4]

[1]elt14rer@student.lu.se [2]elt14akn@student.lu.se [3]elt14nka@student.lu.se
[4]his10eni@student.lu.se

Abstract:
This is a project about a Lego segway modeled as a pendulum. It is built entirely in Lego and the brick and motors are from the brand EV3 while the sensors are from NXT. The sensors used were one gyroscope and one accelerometer which was used together with a complementary filter to get a good estimation of the angle and angular speed of the robot. To control the robot an LQR state feedback design was used.

1. Introduction

The inverted pendulum is a popular control problem due to its unstable and nonlinear dynamics and is often used to test different control strategies and to demonstrate how a inherently unstable system can be stabilized using closed loop feedback control. The purpose of this project is to design and build a two-wheeled inverted pendulum robot, as seen in Figure 1, using the Lego Mindstorms kit and making it balance itself.

Figure 1. Final version of the robot

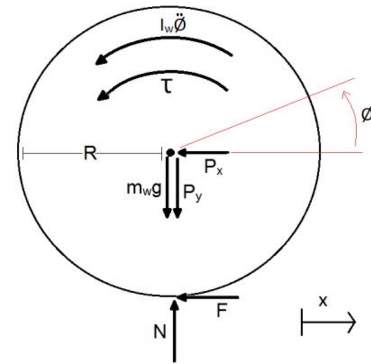


Figure 2. Model of the wheels. Image courtesy of [**1**]

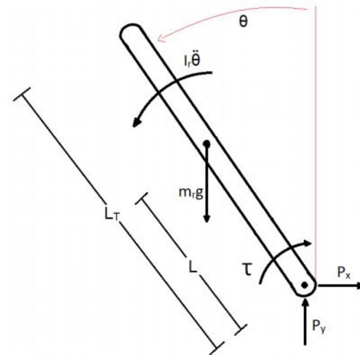


Figure 3. Model of the pendulum. Image courtesy of [**1**]

2. Modeling

In order to simplify the calculations a simplified model of the system is used, the process is modeled as a freely rotating pendulum with two wheels as its base. The variables used for modeling can be seen in Figure 2 and Figure 3 and are summarized in Table 1. In Table 2 the measured constants of the robot can be found.

In order to derive the equations of motion for the system Lagrange equations are used as following [**1**].

Kinetic energy of the wheel

$$T_w = \frac{1}{2}m_w\dot{x}^2 + \frac{1}{2}I_w\dot{\phi}^2. \tag{1}$$

Kinetic energy of the rod

$$T_r = \frac{1}{2}m_r(\dot{x} - L\dot{\theta}cos(\theta))^2 + \frac{1}{2}m_r(L\dot{\theta}sin(\theta))^2 + \frac{1}{2}I_r\dot{\theta}^2. \tag{2}$$

Potential energy

$$V = m_r gLcos(\theta). \tag{3}$$

Table 1. Definitions of variables used in the model

m_w	Mass of wheels	kg
m_r	Mass of rod	kg
I_w	Inertia of wheel	kgm^2
I_r	Inertia of rod	kgm^2
τ	Torque from motors	Nm
R	Radius of wheel	m
L	Length to the center of mass	m
L_r	Length of rod	m
g	Gravity	m/s^2
θ	Angle of the segway	rad
$\dot{\theta}$	Angular speed of the segway	rad/s
ϕ	Angle of the wheels	rad
$\dot{\phi}$	Angular speed of the wheels	rad/s

Table 2. Measured constants

m_w	$2 \cdot 0.032\ kg$
m_r	$1.213\ kg$
I_w	$0.082944\ kgm^2$
I_r	$0.066320775\ kgm^2$
R	$0.036\ m$
L	$0.182\ m$
L_r	$0.405\ m$

Using Euler-Lagrange Equation

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q} = 0, \tag{4}$$

where the Lagrangian is written as

$$\mathcal{L} = T_w + T_r - V. \tag{5}$$

With the general coordinates

$$q = (x \quad \phi \quad \theta)^T.$$

Then deriving from Lagrange the equation of motion can be written on the form [1]

$$O^T M(q) O \dot{v} + O^T F(q, \dot{q}) = O^T G \tau, \tag{6}$$

where O and G are

$$O = \begin{bmatrix} -R & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad and \quad G = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix},$$

with

$$M(q) = \begin{bmatrix} (m_w + m_r) & 0 & -(m_r L cos(\theta)) \\ 0 & I_w & 0 \\ -(m_r L cos(\theta)) & 0 & (I_r + m_r L^2) \end{bmatrix},$$

$$q = \begin{bmatrix} x \\ \phi \\ \theta \end{bmatrix}, \quad \dot{q} = \begin{bmatrix} \dot{x} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix}, \quad \dot{v} = \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \end{bmatrix},$$

$$F(q, \dot{q}) = \begin{bmatrix} m_r L \ddot{\theta} sin(\theta) \\ 0 \\ -m_r g L sin(\theta) \end{bmatrix}.$$

Putting everything together yields the following equations of motion for the system [1].

$$\begin{bmatrix} (m_w + m_r)R^2 + I_w & m_r R L cos\theta \\ (m_r R L cos\theta) & (I_r + m_r L^2) \end{bmatrix} \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} -m_r R L \theta^2 sin\theta \\ -m_r g L sin\theta \end{bmatrix} = \begin{bmatrix} \tau \\ -\tau \end{bmatrix}. \tag{7}$$

These equations are nonlinear and are therefore linearized around the pendulums upright position ($\theta = 0$ and $\phi = 0$).

With the state space vector

$$z = (\phi \quad \dot{\phi} \quad \theta \quad \dot{\theta})^T, \tag{8}$$

and together with (7), the equations of motion can be rewritten as a state space model using Jacobian

$$J_f(x) = A = \begin{bmatrix} \frac{\partial \dot{\phi}}{\partial \phi} & \frac{\partial \dot{\phi}}{\partial \dot{\phi}} & \frac{\partial \dot{\phi}}{\partial \theta} & \frac{\partial \dot{\phi}}{\partial \dot{\theta}} \\ \frac{\partial \ddot{\phi}}{\partial \phi} & \frac{\partial \ddot{\phi}}{\partial \dot{\phi}} & \frac{\partial \ddot{\phi}}{\partial \theta} & \frac{\partial \ddot{\phi}}{\partial \dot{\theta}} \\ \frac{\partial \dot{\theta}}{\partial \phi} & \frac{\partial \dot{\theta}}{\partial \dot{\phi}} & \frac{\partial \dot{\theta}}{\partial \theta} & \frac{\partial \dot{\theta}}{\partial \dot{\theta}} \\ \frac{\partial \ddot{\theta}}{\partial \phi} & \frac{\partial \ddot{\theta}}{\partial \dot{\phi}} & \frac{\partial \ddot{\theta}}{\partial \theta} & \frac{\partial \ddot{\theta}}{\partial \dot{\theta}} \end{bmatrix}$$

and

$$J_f(u) = B = \begin{bmatrix} \frac{\partial \dot{\phi}}{\partial u} \\ \frac{\partial \ddot{\phi}}{\partial u} \\ \frac{\partial \dot{\theta}}{\partial u} \\ \frac{\partial \ddot{\theta}}{\partial u} \end{bmatrix},$$

to get the system of the form

$$\dot{x} = Ax + Bu, \tag{9}$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -316 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 57 & 0 \end{bmatrix},$$

$$B = \begin{bmatrix} 0 \\ 5827 \\ 0 \\ -467 \end{bmatrix},$$

and $u = \tau$, which is the torque acting on the wheel. The nonlinear state equations are linearized around the upright position and are therefore only valid for small deviations from this position.

2.1 Motor

Since the model uses torque τ, as the input of the system but the motors takes voltage as input a translation was needed from torque to voltage. After some research a mathematical model of the motor could be found [11]

$$\begin{cases} \dot{\omega} = \frac{K_\tau I - B\omega - A_r - \tau_d}{J} \\ \dot{I} = \frac{U - R_a I - K_b \omega}{L_a} \end{cases}. \tag{10}$$

With the assumption that the current changes so fast the derivative would be zero, the following expression could be computed

$$U = \frac{R_a \tau}{K_\tau} + B\omega \iff \tau = \frac{UK_\tau}{R_a} - \frac{BK_\tau}{R_a}\omega, \quad (11)$$

with the variables declared in table 3.

Table 3. Definitions of variables used in (10) and (11)

I	Current	A
U	Voltage	V
τ_d	Shaft's load torque	Nm
R_a	armature resistance	Ω
K_τ	Torque constant	Nm/A
K_b	Back electromotive force coefficient	$V/(rad/s)$
J	Rotor's moment of inertia	kgm^2
B	Viscosity resistance coefficient	$Nm/(rad/s)$
A_r	Dry friction force	Nm

τ was then replaced in the old model (7) with (11) and the linearization was recalculated. This gave the new A and B matrices

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.0304 & -141.1684 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0.0026 & 30.8700 & 0 \end{bmatrix},$$

$$B = \begin{bmatrix} 0 \\ 41.8679 \\ 0 \\ -3.5432 \end{bmatrix},$$

with the eigenvalues

$$\text{Eigenvalues} = [0 \quad -0.0187 \quad -5.5620 \quad 5.5502]. \quad (12)$$

As seen in (12) the eigenvalues of the system includes two mirrored poles one being unstable at 5.5502, this is typical for a inverted pendulum model. Since the closed loop system includes an unstable pole some type of control system is needed to stabilize it.

3. Electro-Mechanics

The robot is constructed using the EV3 brick for computations combined with the NXT gyroscope and accelerometer sensors for taking measurements of the angle and angular velocity and is then driven by two EV3 large motors, the rest of the frame is constructed using simple Lego parts. A taller and heavier pendulum is easier to control since it falls slower and there is more time to correct for errors, therefore the center of mass was put as high as possible without making the construction unstable.

At the start of the project there were issues with backlash in the motors. So when the robot tried to balance around its upright position and there was a small error in angle to correct, the motors would turn but not the wheels due to the backlash and this made it hard to balance. To solve this two motors were used on each side instead of one, and then each wheel was only rotated in one direction since the backlash problem only occurs when changing direction of the wheels.

3.1 Specification for EV3

These are the specifications for the different parts used for building the robot.

- EV3 Brick [**2**]
  - Operating System - ev3dev (Debian Linux-based)
  - Language used - Python
  - Processor - 300 MHz ARM9 Controller
  - Flash Memory - 16 MB
  - RAM - 64 MB
  - USB 2.0 Communication to Host PC - Up to 480 Mbit/sec
  - Micro SD Card - Supports SDHC, Version 2.0, Max 32 GB
  - Power - 6 rechargeable AA batteries
- EV3 Motors [**3**]
  - Motor max speed at 170 rpm
  - Running torque of 20 Ncm
  - Stall torque of 40 Ncm
- NXT Acceleration Sensor [**4**]
  - Three axes of measurement labeled x, y, and z
  - The acceleration measurement for each axis is refreshed 100 times per second
- NXT Gyro Sensor [**5**]
  - Measures angular velocity in one direction

4. Control

4.1 LQR

To control the segway an LQR (Linear Quadratic Regulator) controller was used. This method was chosen since it guarantees a stable closed loop system and it also guarantees robustness with a phase margin of 60°. The theory used for the implementation of the LQR comes from [**6**]. The parameters in the LQR was calculated in Matlab using the cost function

$$J = \int_0^\infty (x^\intercal(t)Q(t)x(t) + u^\intercal(t)R(t)u(t))dt, \quad (13)$$

with

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 6 \cdot 10^{10} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

which is a diagonal matrix with a weighted penalty for each state. $Q(3,3)$ represents the weight for θ, the angle between the normal of the horizontal and the robot, see Figure 3, and have the highest penalty since it is the most important state.

The penalty for the input signal is

$$R = 10^6.$$

This together with the linearized system was used to solve the Riccati equation

$$0 = Q + A^T S + SA - SBR^{-1}B^T S. \quad (14)$$

After solving for S the LQR gain is calculated as

$$L = R^{-1}B^T S, \quad (15)$$

and the optimal cost as

$$J^\star = x(0)^T Sx(0). \quad (16)$$

The optimal controller is then

$$u = -Lx. \quad (17)$$

So that the state feedback system with the LQR gain looks like

$$\dot{x} = (A - BL)x, \quad (18)$$

which gives the optimal pole placement for the system.

4.2 Complementary filter

When measuring the angle θ and angular speed $\dot{\theta}$ in Figure 3, both an accelerometer and a gyroscope were used. The accelerometer measures the difference in gravity in three dimensions, and after removing an offset and scaling the signal the angle of the robot was retrieved. The gyroscope measures the angular velocity which can be numerically integrated to get the same angle.

The accelerometer and the gyroscope is consequently used to calculate the same angle. The reason for using both of these two sensors is due to how each sensor operates. While a gyroscope is efficient at measuring fast changes it does however have the drawback that when integrating the angular velocity there are small errors causing the angle to drift with time. The accelerometer on the other hand is very good at measuring the angle after a long time but is instead poor at measuring the angle in one point in time as the measurements can be very noisy. In order to minimize both of these problems a combination of the two sensors are used in a complementary filter [7].

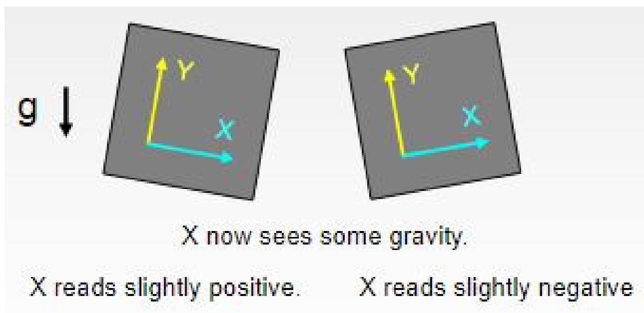


Figure 4. The impact of gravity in x and y dimension. Image courtesy of [7]

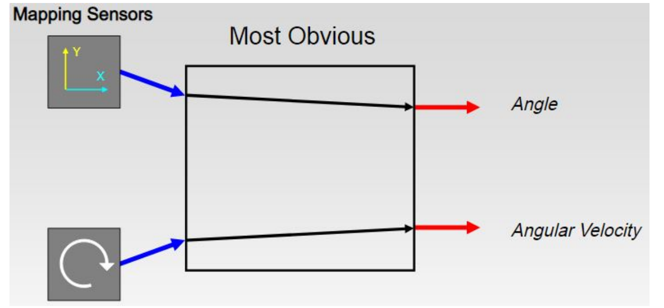


Figure 5. Implementation without filtering. Image courtesy of [7]

In Figure 4 a graphical representation of how the accelerometer measures gravity demonstrates that for measuring angles in the robots upright position the x-component will give the most significant readout. As the robot only operates within a small variance of angles, the measurements of the x-component was estimated as the angle θ, instead of $sin\ \theta$.

As is shown in Figure 5, the most obvious choice of θ is the readout from the accelerometer but as stated above, that is not the optimal angle measurement.

A far more superior way to obtain accurate and none drifting value of the angle θ is to use a complementary filter as shown in Figure 6.

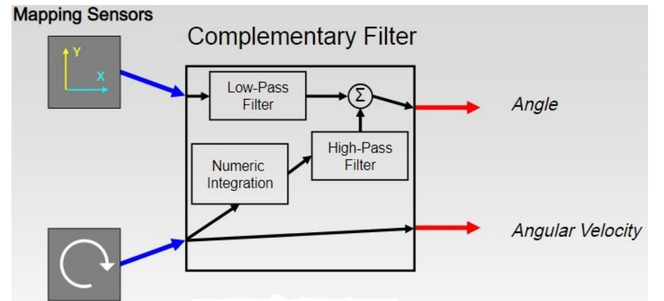


Figure 6. Implementation with filter. Image courtesy of [7]

The generalized equation of this complementary filter is

$$\text{angle} = (a) \cdot (\text{angle} + gyro \cdot dt) + (1 - a) \cdot (x_{acc}), \quad (19)$$

where a is calculated as

$$\beta = \frac{a \cdot dt}{1 - a} \iff a = \frac{\beta}{\beta + dt}, \quad (20)$$

and β is a time constant depending on how fast the measurements drift and $dt = \frac{1}{f_s}$, where f_s is the sampling frequency.

5. Implementation

5.1 Model

To be able to make calculations and create a controller for a physical process a model is needed. The model is a simplified version of reality where assumption has been made to make the model simple enough to work with. For instance, friction on the motors and air resistance has been neglected.

The model was linearized and then discretized using Matlab. First the model in Equation (7) was entered together with measured constants, the nonlinear system was calculated with Matlabs `syms` and with matrix calculation to receive $\dot{x} = f(x)$, where x is the state vector in (8). The Matlab function `Jacobian` was then used to obtain the A- and B-matrix in (9). To discretize the state space system the Matlab function `c2d` was used.

5.2 LQR

To calculate the LQR gain matrix, Matlabs `LQR` function was used with the previously calculated system matrices A, B, Q, R. The vector L consists of four weighted constant numbers used to calculate the control signal as stated in (17). The L vector was calculated to

$$L = [0 \quad -10 \quad -1405.4 \quad -249.7].$$

When implementing the above presented control structure a problem with delivering the correct output to the motor was found, as presented in Section (2.1).

With the new A- and B-matrices the final values of the LQR was calculated to

$$L = [-0.0010 \quad -0.0476 \quad -254.3399 \quad -12.5356].$$

5.3 Complementary filter

When implementing the complementary filter, as described above, some calibrations of the sensors were needed. These calibrations were made to make sure the sensors would return values in the correct range and that the angle that was expected was given. For this the following equations were used

$$acc = acc_{value} \cdot acc_{scale} - acc_{offset}, \tag{21}$$

$$gyro = (gyro_{value} - gyro_{offset}) \cdot gyro_{scale}, \tag{22}$$

were acc_{value} is the value read from the accelerometer and $gyro_{value}$ is the value read from the gyroscope.

The scale is to convert the sensor values into radians and radians/s. The scale value should be found by reading the product description of the sensors, but since no product descriptions could be found, another article solving the same problem was used for inspiration [**8**, **9**].

The offset in the accelerometer consist of two parts. First to change the range from $0 < \theta < 2\pi$ to $-\pi < \theta < \pi$ and secondly due to the robots center of gravity not being exactly at $\theta = 0$. The second offset is calculated by placing the robot in its upright equilibrium position where the robot is balancing. This should be where the angle is 0, if the sensors read a different value than zero, this is the offset.

The gyroscope's offset was easy to find and given by the value received when the robot was not moving. The final values used were

$$acc_{offset} = 0.0480\ rad,$$
$$acc_{scale} = 0.1760,$$
$$gyro_{offset} = 2971\ mV,$$
$$gyro_{scale} = 0.2084.$$

To find the weighting of the different parts of the complementary filter the desired time constant β, was determined by trial and error and since the gyroscope was giving the most accurate representation of the angle, most weight was put on the integrated gyroscope signal. The sampling frequency of the sensor was given in the data sheet. The final implementation of the filter is

$$\text{angle} = (0.93) \cdot (\text{angle} + gyro \cdot dt) + (1 - 0.93) \cdot (x_{acc}). \tag{23}$$

5.4 Program structure

The implementation of the program running on the EV3 brick on the robot was done in Python. To set up a good program structure a flowchart was made to get a better understanding of what programs were needed, see Figure 7.

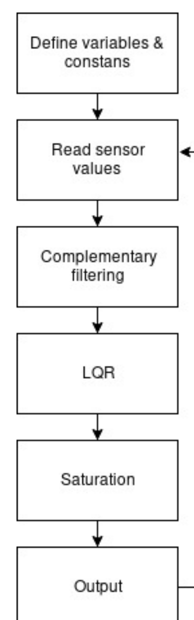


Figure 7. Flowchart for program structure

In the beginning of the program variables and constants are declared. After this the sensor- and motor-objects are created. This is implemented using the ev3dev library [**10**]. Here classes for both reading and writing to the different components of the Lego Mindstorms set can be found.

When everything is declared the loop starts. The loop is run from the main file and is set to keep running as long as the robot angle isn't larger than a set amount.

At the top of the loop the sensor values are read and processed the complementary filter is used to calculate the estimation of the angle. The obtained states are then used with the calculated LQR gain and the proper control signal is calculated.

```
while loop:
    t1 = time.time()

     #reading gyro value
    gyro = gyroSensor.value()

    #reading accelerometer value
```

```
    accelraw =  4 * accelSensor.value(n=2)
    + accelSensor.value(n=5)

    #defining values between 211-360 degrees
    #as -149-0 degrees
    if accelraw > 600:
        accelraw = accelraw - 1023

    #scaling accelerometers values to degrees
    accel = accelraw * acc_scale

     #scaling gyro and removing offset
   angularSpeed = (gyro - gyro_offset) * gyro_scale

    #complementary filter with weights
    angle = (0.95) * (angle - angularSpeed * dt)
    + (0.05) * (accel - 4.2)
    angle = angle

    #reading speed and position of wheels
    motorEncoderStop = rightWheelC.position
    wheelSpeed = rightWheelC.speed

    #calculate the angle of wheels
    wheelangle = motorEncoderStart
    - motorEncoderStop

    #LQR feedback
      u=(0.0010*wheelangle  -0.0476 *wheelSpeed
      -254.3399*angle + 12.5356*angularSpeed)
```

To guarantee that the control signal is within the bounds of the motors the control signal needs to be saturated. The maximum input signal the motors can handle is 100 but the motors start to act in a nonlinear way meaning that each percent of duty cycle does not give the same rpm output as early as 80. The saturation makes sure the value never exceeds 100 if the limit is chosen to 100. If the nonlinear part becomes unmanageable the saturation can be set to 80 instead.

The saturated signal is then given to the motors and the loop time is measured to guarantee that each loop iteration takes the same amount of time and after that the loop starts over.

```
    motorinput = 10*u

    #counteract saturation
    if motorinput > 90:
            motorinput = 90

    if motorinput < -90:
            motorinput = -90

    #decide which motor to run to counter backlash
    if motor:

        if motorinput > 0:

            if direction == 1:
              leftWheelB.run_direct(duty_cycle_sp
                = motorinput)
             rightWheelC.run_direct(duty_cycle_sp
                = motorinput)
             leftWheelA.stop(stop_action="coast")
            rightWheelD.stop(stop_action="coast")

            else:
              leftWheelB.run_direct(duty_cycle_sp
                = motorinput)
             rightWheelC.run_direct(duty_cycle_sp
                = motorinput)
             leftWheelA.stop(stop_action="coast")
            rightWheelD.stop(stop_action="coast")
            direction = 1

        if motorinput < 0:

            if direction == -1:
              leftWheelA.run_direct(duty_cycle_sp
                = motorinput)
             rightWheelD.run_direct(duty_cycle_sp
                = motorinput)
             leftWheelB.stop(stop_action="coast")
            rightWheelC.stop(stop_action="coast")

            else:
              leftWheelA.run_direct(duty_cycle_sp
                = motorinput)
             rightWheelD.run_direct(duty_cycle_sp
                = motorinput)
             leftWheelB.stop(stop_action="coast")
            rightWheelC.stop(stop_action="coast")
            direction = -1

    #stops the motors if the segway falls
    if angle > 60:
        leftWheelA.stop(stop_action="coast")
        leftWheelB.stop(stop_action="coast")
        rightWheelC.stop(stop_action="coast")
        rightWheelD.stop(stop_action="coast")
        loop=False

    t2 = time.time()
    tdiff = t2 - t1

    #sleep for the rest of the sample time
    if tdiff < dt:
        time.sleep(dt - tdiff)
```

6. Results

In Figure 8 the estimated angle from complementary filter, gyroscope and accelerometer is shown on the actual robot when the motors are turned off and the angle changed by hand. As seen in the figure the complementary filter works as intended since it gives almost the same angle as the integrated gyroscope value except for the drift which can be seen after some time.

The current implementation of the robot is not able to achieve the goal of self-balancing but it is very close. When the robots angle is close to zero, it is too sensitive to small changes in angle and the motors drives too aggressively. On the other hand when the robot is falling over and the angle is large the motors isn't aggressive enough to correct the error.

Even though during the course of the project different steps was taken to be able to achieve these goal since the robot still displays an oscillating behavior. The different steps taken to achieve the goal of self-balancing can be found in Section 7.

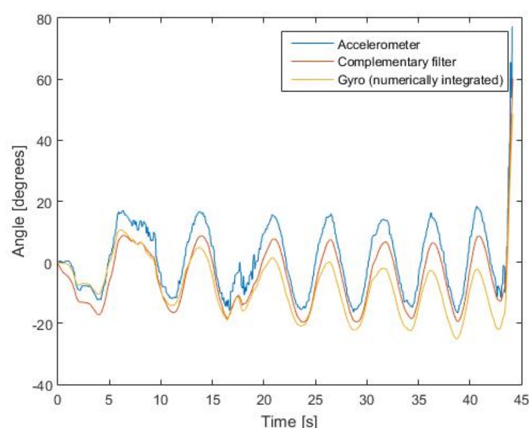


Figure 8. Estimated angle from the Complementary filter, Gyroscope and accelerometer at different times

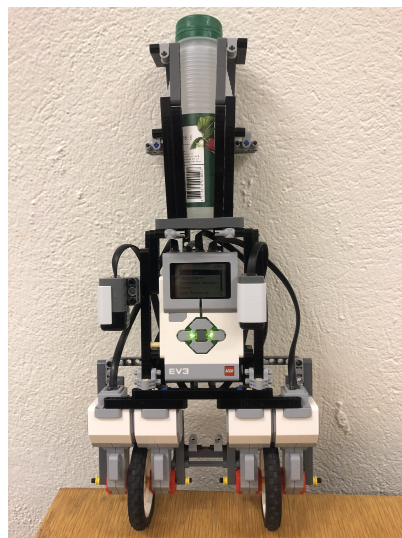


Figure 9. Rebuilt version of the robot with higher center of mass

7. Discussion

To balance the robot at the upright position a fast system is needed. This means that the iteration time of the loop needs to be as low as possible. From doing similar projects an initial guess for the desired iteration time was made for 20 *ms*. However since it takes too long to make all the necessary readings of sensor- and motor values and the calculations before an output is sent. The fastest sample time achievable after all the calculations are made was closer to 30 *ms* which should still be enough to achieve self-balancing. One theory on why the robot is unstable to balance is that it actually takes longer than 30 *ms* due to the inner workings of the EV3 Brick that doesn't show up when timing the code. There was also a strange problem with the code running at different speeds when using different computers even though all of the code is run on the actual EV3 Brick and not the computer.

One way to make the time constant of the system bigger is by making the robot taller thus reducing the necessity for a fast controller. This was implemented during the project since it was hard to achieve balancing. Thus changing the robot design from the version shown in figure 1 to the version shown in figure 9. The robot was built about 10 *cm* higher which moved the center of mass with 3 *cm*. This didn't have the desired control improvement on making the robot more stable and less oscillating around the upright position.

Trying to reduce the loop time to make the system go faster was made by installing an new version of the current operating system which was still in beta[**10**]. This new version reduced the loop time of the system by 6 *ms*. Making the new loop time to be around 23 *ms*. Reducing the loop time showed an small improvement on being able to self balance. However the robot is still unstable and thus unable to achieve self balancing.

Implementing the second pair of motors were made to reduce the robots jerky and oscillating behavior when trying to balance. This behavior was guessed to be coming from the backlash of the motors. Since the backlash was around 1-5° it was thought that when an small output was sent to the motors, to correct a small change in inclination, the backlash would prevent the wheels from actually moving even though the motors moves. It was then implemented that one motor on each wheel would only drive in a certain direction, while the other motor would apply a small opposite force towards the other direction. Making the motors apply a constant tension on the wheels thus reducing the backlash.
This provided less effect on the oscillating behavior than expected and had little to none effect on solving the problem.

Additional tries to reduce the oscillating behavior were made by tuning the Q and R matrices, by increasing/decreasing the penalty put on the θ angle, thus changing the behavior of the controller. So far no change in parameters has made the oscillating behavior disappear and making the robot be able to self-balance. The model of the process was also recalculated using different methods and programs but yielded the same result.

As shown in Figure 8 the signal from the accelerometer is a bit noisy and that is the reason behind trusting the value given from the gyroscope more than the accelerometer.

Bibliography

[1] Saam Ostovari, Nick Morozovsky, Thomas Bewley, "The dynamics of a Mobile Inverted Pendulum", UCSD, Last time visited: 2018-12-20, URL: `http://renaissance.ucsd.edu/courses/mae143c/MIPdynamics.pdf?`

fbclid=IwAR006fWrmt6sLgNowsJR6UNM9SPKqPM_Gf2ZLQfz9vI8piTBW7M2jwhfqKc

[2] Mindstorms, Last time visited: 2018-12-02, URL: https://shop.lego.com/en-US/product/EV3-Intelligent-Brick-45500

[3] Mindstorms, Last time visited: 2018-12-02, URL: https://shop.lego.com/en-LT/product/EV3-Large-Servo-Motor-45502

[4] NXT, Last time visited: 2018-12-02, URL: https://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NAC1040

[5] NXT, Last time visited: 2018-12-02, URL: https://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NGY1044

[6] Anton Cervin, "Linear-quadratic control", LTH, 2018, Last time visited: 2018-12-14, URL: http://archive.control.lth.se/media/Education/EngineeringProgram/FRTN10/2018/L09_2018.pdf

[7] Shane Colton, "The Balance Filter", MIT, 2007, Last time visited: 2018-12-20, URL: https://robottini.altervista.org/wp-content/uploads/2014/04/filter.pdf

[8] Laurensvalk, "Parameters.py", 2017, Last time visited: 2018-12-19, URL: https://github.com/laurensvalk/segway/blob/master/ev3/ev3dev/python/parameters.py

[9] Laurensvalk, "Tutorial: Self-Balancing EV3 Robot", Last time visited: 2018-12-16, URL: http://robotsquare.com/2014/07/01/tutorial-ev3-self-balancing-robot/?fbclid=IwAR2lOxWq1LaX-Prm0Yt5fSrKEo490pqSAECLDeNPrlIzlRuGzA8AvvYHkJ8

[10] Ralph Hempel et al, "Python language bindings for ev3dev", 2015, Last time visited: 2018-11-27, URL: https://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/stable/

[11] Maxim, "Mathematical Model of Lego EV3 Motor", 2015, Last time visited: 2018-12-20, URL: http://nxt-unroller.blogspot.com/2015/03/mathematical-model-of-lego-ev3-motor.html

[12] Laurensvalk, "Parameters.py", 2017, Last time visited: 2018-12-20, URL: https://github.com/laurensvalk/segway/blob/master/ev3/ev3dev/python/parameters.py

Obstacle Avoidance using a LEGO Mindstorms EV3DEV vehicle

Johan Bladh[1] **Mia Grahovic**[2] **Madeleine Rosicki**[3] **Tomas Tamilinas**[4]

[1]mas14jbl@student.lu.se [2]elt14mgr@student.lu.se [3]har12mr1@student.lu.se [4]mas14tsz@student.lu.se

Abstract: The objective of this project is to avoid obstacles with a LEGO Mindstorms EV3 vehicle. The vehicle is equipped with a regular webcam that is connected to a PC. By image processing, both the position and the relative angle between the obstacle and the vehicle can be determined. The used controller is a simple condition-based P-regulator and the commands to the vehicle are based on the results from the image feed. The communication between the vehicle and the PC is achieved using the UDP communication protocol and all code is written in Python. A model of the used control system has been modelled in Simulink. The resulting avoidance sequence follows the following steps: the vehicle is to approach the obstacle as long as the obstacle is far away, when it has reached a close distance it should strive to keep a relative angle to the obstacle until reaching an even closer distance to the obstacle. Finally, a hard-coded sequence is initialized to complete the avoiding maneuver and getting back on the initial vehicle heading without any information about the obstacle. The final result is limited by the large delays in communication between the PC and the LEGO vehicle, and also by a large play in the steering of the vehicle.

1. Introduction

This project is done for the course Project in Automatic Control FRTN40. The team consists of two M-students and two E-students. The project is about obstacle avoidance using a LEGO Mindstorms EV3 vehicle at low speed. The available hardware is a LEGO vehicle and a simple webcam. The vehicle was built beforehand and a small platform for holding the webcam in place on top of the vehicle was built and installed. By image processing, which is done with the OpenCV library, both the position and the relative angle between the obstacle and the vehicle can be determined. A controller is designed that bases its control signal on the results from the image feed. The control signals from the regulator are sent to the LEGO vehicle. The communication between the vehicle and the computer is achieved by using the UDP communication protocol. All code is written in Python. Simulink in MATLAB is used to create a model of the used control system prior to the actual application of the system. A scenario example can be seen in Figure 1.

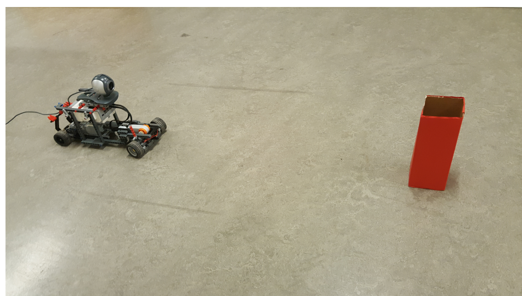


Figure 1. Scenario example of the LEGO vehicle with obstacle.

The initial aim of the project is to navigate the LEGO vehicle around obstacles that appear in certain positions relative the vehicle. Multiple obstacles are a possibility for further development. The project team members' prior relevant educational backgrounds are within automatic control, mechanics and real time programming. This knowledge has been applied upon designing the controller of the vehicle as well as during programming the communication between the client and the server.

The subject explored in this project is relevant for future research of automotive vehicles and can be seen as a roadmap for improving current technologies. Improvements can be made by using additional sensors together with or instead of a camera on the vehicle.

2. Modelling

Before the project is executed, a model of the LEGO vehicle is created in order to simplify the designing of a satisfactory control algorithm but also to simply show that a solution can be achieved using the given tools and the available information about the obstacle. The result from the modelling is to be a trajectory describing how the LEGO vehicle moves around the obstacle, as seen from above, in the xy-plane. Although all measuring and regulating in the final solution is to be relative the obstacle to be avoided, global xy-coordinates are needed to simulate and plot the vehicle path, but not necessarily the regulation of the real process.

2.1 Modelling of the LEGO vehicle

The movement of the LEGO vehicle is to be modelled in the xy-plane and therefore is conveniently represented as a dot in the plane. The dot is conventionally placed at the center of

the vehicle. In order to determine how this dot moves in the xy-plane as the front wheels rotate a kinematic model of the vehicle is made. An already existing model of a vehicle made by R. Rajamani [3] is used in this project. The used signals are listed in Table 1 below.

Table 1. All parameters used in final model

Signal	Unit	Description
δ_{f}	[rad]	Steering angle front wheels
δ_{f0}	[rad]	Front steering angle offset
$\mathcal{L}_{\mathrm{f}}$	[m]	Front axle to vehicle center distance
$\mathcal{L}_{\mathrm{r}}$	[m]	Rear axle to vehicle center distance
$\mathcal{X}$	[m]	Vehicle center x-coordinate
$\mathcal{Y}$	[m]	Vehicle center y-coordinate
Ψ	[rad]	Vehicle orientation with x-axis
$\mathcal{V}$	[m/s]	Vehicle center velocity

The model is based on a simplification of a four-wheeled vehicle. Figure 2 below shows a bicycle model of the vehicle. In this model, the front wheels are represented by one single wheel at point A and in the same manner the rear wheels are represented by one central wheel at point B. The steering angles for the front and rear wheels are represented by δ_{f} and δ_{r}, respectively. Furthermore, $\mathcal{L}_{\mathrm{f}}$ and $\mathcal{L}_{\mathrm{r}}$ are the distances between the front wheels and the center of the vehicle and the rear wheels and the center of the vehicle, respectively. The angle β in Figure 2 refers to the slip angle of the vehicle.

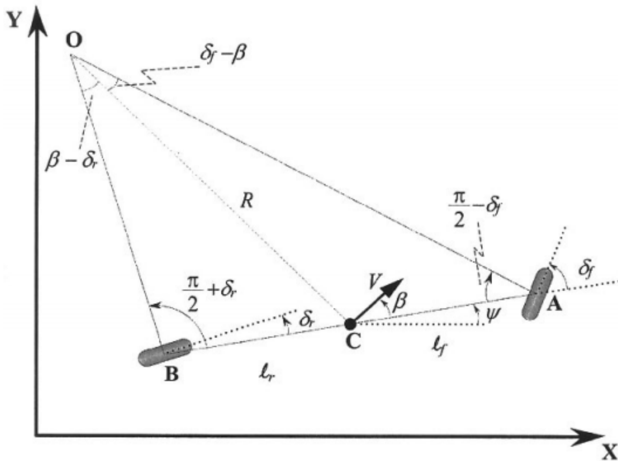


Figure 2. Kinematics of vehicle motion. Source: Rajamani [3].

The values of interest are the following: current x-coordinate of the vehicle ($\mathcal{X}$), current y-coordinate of the vehicle ($\mathcal{Y}$), and the orientation of the vehicle relative to the longitudinal axis (Ψ).

For the model of the LEGO vehicle the value of δ_{r} is set to zero as there is no rear wheel steering on the vehicle. A major assumption used in the development of the kinematic model is that the velocity vectors at points A and B in Figure 2 are in the direction of the front and rear wheels, respectively. This is equivalent to assuming that the slip angles at both wheels are zero. Furthermore, due to the LEGO vehicle travelling at low speeds the slip angle of the vehicle is also set to zero.

Due to a noticed offset in the steering angle of the LEGO vehicle, an additional term, δ_{f0}, defining the angular offset of the steering is inserted into the model.

The resulting equation system defining the movement of the vehicle is seen in (1).

$$\begin{aligned} \dot{\mathcal{X}} &= \mathcal{V} \cdot \cos \Psi \\ \dot{\mathcal{Y}} &= \mathcal{V} \cdot \sin \Psi \\ \dot{\Psi} &= \frac{\mathcal{V}}{\mathcal{L}_{\mathrm{f}} + \mathcal{L}_{\mathrm{r}}} \cdot \tan\left(\delta_{\mathrm{f}} + \delta_{\mathrm{f0}}\right) \end{aligned} \tag{1}$$

The parameters that are controlled by the regulator, and thus the only way of controlling the vehicle, are the velocity of the vehicle, $\mathcal{V}$, and the steering angle of the front wheels, δ_{f}.

2.2 Discrete model approximation

System modeling usually requires discretization since the implementation of real systems are mostly done digitally. The discretization can affect the outcome and theoretically even the model should be discretized. Another case to consider is the linearization of the model since linearization often is used to make calculations easier. The resulting model in section 2.1 is a continuous-time nonlinear model of the vehicle. This model can be discretized and linearized. To determine what type of vehicle model that should be used in this case, four vehicle models are studied. The models are the following:

- continuous-time nonlinear model
- discrete-time nonlinear model
- continuous-time linearized model
- discrete-time linearized model.

The system response for a varying input signal for both the speed and the steering angle on the front wheel for the four different models is shown in Figure 3.

The results in Figure 3 show that the responses of the discretized models are almost equal to the one of the continuous model. The same applies to the linearized models. Any of the four models above can then be used in the final result. Due to the simplicity of the continuous-time model, the original equations in (1) are used.

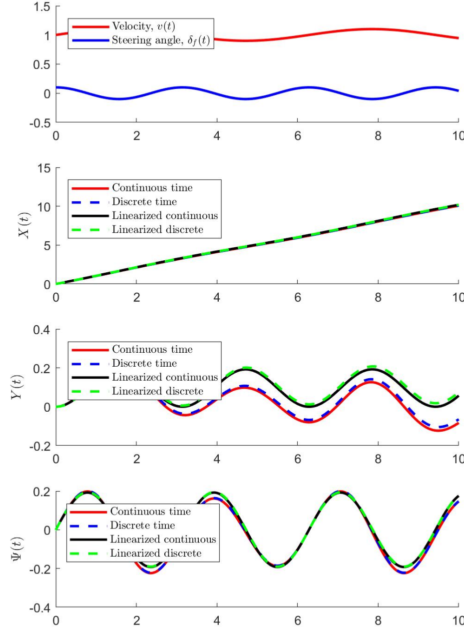


Figure 3. Vehicle dynamics response for different vehicle models. Source: Marcus Greiff [1].

3. MATLAB implementation

To design the controller and to study the behavior of the system a model is developed in MATLAB Simulink. The overall structure of the system can be seen in Appendix A. The model consists of three major function blocks: the controller, the vehicle model and the block representing the camera.

The controller is a simple P-controller where if-statements are used to determine the correct reference values for each state. The P-controller is used on the error in the relative angle to the obstacle. The error is calculated as the difference between the reference angle and the measured angle. The angle here refers to the relative angle between the moving vehicle and the obstacle. The absolute value of the angle is zero when the obstacle is directly in front of the vehicle and it increases once the obstacle is further away from the center view of the vehicle. The output signals of the controller are the velocity and steering angle of the front wheels of the vehicle.

When the vehicle is very far away from the obstacle the vehicle should steer towards it and the reference angle between the obstacle and the vehicle axis should therefore be set to 0 degrees. It seems a bit counterintuitive to steer the vehicle towards the obstacle but the reason for doing this in the Simulink simulation is to force the problem in order to be able to avoid the obstacle using the provided solution. The case where the vehicle is already oriented in such a manner that it will avoid the obstacle if it was to go forward without any steering are not of interest in this particular simulation.

When the vehicle has come within a reasonable range to the obstacle, a new reference angle value is set to initiate avoiding the obstacle. When the case is neither of the above, the reference angle to obstacle is then set to 180 degrees resulting in the vehicle searching for the obstacle.

The second block called vehicle model is where the previously derived equations in section 2.1 are implemented.

The third block, called camera, emulates the camera mounted on the vehicle. This block has two input parameters which are the state vector and the obstacle location. Using the x- and y- coordinates (of the vehicle) contained in the state vector as well as the coordinates of the obstacle, both the distance and the angle to the obstacle can be calculated. These values are output signals from the camera block and are fed to the controller. The resulting path during obstacle avoidance is seen in Figure 4.

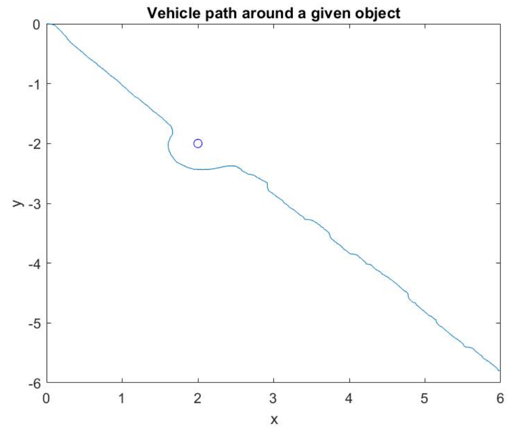


Figure 4. Vehicle path around a given obstacle.

4. Image processing

The two input parameters to the regulator are: distance and relative angle between the vehicle and the obstacle. In order to obtain these two values continuously during the process a webcam is used. This device is mounted on top of the LEGO vehicle and aims in the direction the vehicle is oriented. The information output from this device is an RGB (red, green and blue additive color mixing) image feed with an approximate speed of 30 frames per second (FPS), i.e. 30 Hz. The RGB color space is illustrated in Figure 5.

In order to calculate the distance and relative angle to the obstacle the obstacle itself needs to be identified on the images that are taken. Due to the lack of proximity sensors or similar devices used in the project, the obstacle is colored in a distinct hue to make it stand out in the raw RGB image. The chosen colors in this project are, as seen in Figure 1, red, green and yellow. Other colors can be used with the only requirement being that they are distinctive enough from the surrounding environment. The software used for image processing in this project is a library of functions called OpenCV. This library contains all the needed functions described below.

The sequence of filters used on the original image in order to distinguish the obstacle within the picture begins with a Gaussian blur. This is done to reduce image noise that would otherwise affect subsequent processing steps of the image, an example of the resulting image can be seen in Figure 6. The second step is converting the image from the RGB to the HSV

color space. HSV stands for hue, saturation and value and is an alternative representation of the RGB model. In order to filter out all but the obstacle's distinct colors within the image, a range of colors defining the acceptable values of the individual pixels needs to be defined. The value of hue defines the color while the values of saturation and value in HSV defines how intense the color is and how bright it is, respectively, see Figure 5. This type of grouping of colors allows for simple defining of the obstacle's color. Furthermore, by using the HSV color space, different lightning conditions are accepted by slightly extending the acceptable range for values of saturation and value. Defining the acceptable color range in the RGB color space would require multiple ranges and more complicated coding.

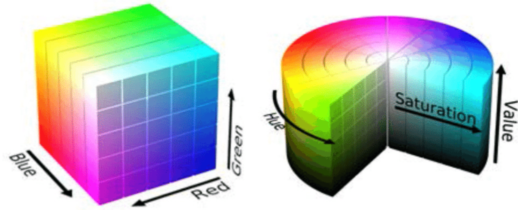


Figure 5. RGB (left) and HSV (right) color space. Source: Roza et al. [4].

By defining the range with two vectors that represent the lowest and the highest acceptable values of HSV, the program goes pixel by pixel for each image and determines whether the pixel color is within the defined range. After this is done, the image is converted into a binary image with white pixels where the colors were within range and black pixels for the rest. Anything but the obstacle is therefore sorted out. An example of the resulting image is shown in Figure 6. The used ranges for the values hue, saturation and value for the different colored obstacles are shown in Table 2 below.

Table 2. Acceptable range for HSV-values

Range	Threshold (value range: [0,255])
Red, low	[0,110,110]
Red, high	[8,255,255]
Yellow, low	[20,120,120]
Yellow, high	[30,255,255]
Green, low	[30,30,30]
Green, high	[75,255,255]

After the previous step the image is smoothed once again with Gaussian blur. Due to the geometry of the obstacles, the image should ideally now only consist of a white rectangle placed somewhere within the image. The next vital step in the image processing sequence is to find the outer edges of this rectangle. To do this, an algorithm developed by Canny J. called Canny is performed on the image. The resulting image, see Figure 6, is a binary image containing only the edges of the obstacle. This binary image is then subjected to the two morphology operators Dilation and Erosion to define the edges more clearly.

The next step within the image processing is to mathematically define all the contours in the binary image obtained

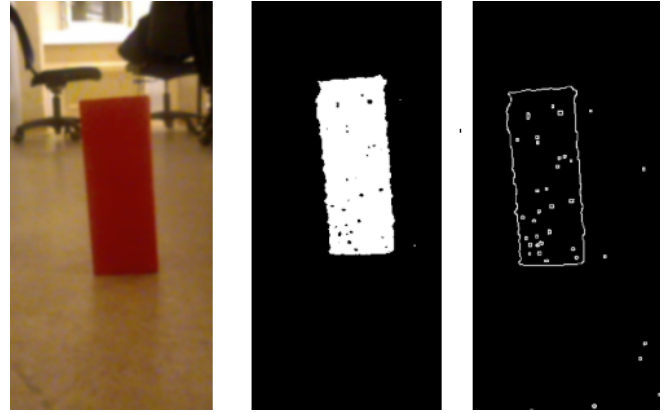


Figure 6. Images from the webcam: Gaussian blur in RGB, color filtering with HSV, Canny edging.

after the previous step. This is done with a function called `cv2.findContours` within the `OpenCV` library. Subsequently, the function `cv2.contourArea` is used on the result from the previous step. The function tries to find all enclosed contours and approximates all of these as rectangles. The function returns an array of information about all enclosures it finds where the following information is included: width, height, orientation and the box's central x- and y- position in the image. If the camera and all filters worked perfectly, there would only be one returned enclosure, that of the obstacle. Nevertheless, due to some remaining noise in the image, there will be multiple enclosures that the method finds and returns. In order to find the rectangle that represents the obstacle, the enclosure with the biggest area value is selected. This way, all the other enclosures created by either noise or different smaller obstacles of the same color are ignored. By now there is enough information available about the obstacle: the obstacle's width, height, and x- and y-position of the obstacle on the image.

4.1 Calculating distance and relative angle

The aforementioned information about the obstacle are all given in pixels. There is a need of further calculations in order to obtain the two needed measurements (distance in meters and relative angle in degrees). All of the parameters and variables included in these calculations are listed in Table 3 below.

Table 3. All terms used in distance and angle calculation

Term	Unit	Description
L_f	[cm]	Parameter used for distance measuring
$L_{\mathtt{ref}}$	[cm]	Reference distance to obstacle
$h_{\mathtt{px,ref}}$	[px]	Obstacle height in reference image
$h_{\mathtt{px}}$	[px]	Obstacle height in subsequent images
$d_{\mathtt{obj}}$	[cm]	Calculated distance to obstacle
$W_{\mathtt{im}}$	[px]	Image width
$B_{\mathtt{c}}$	[px]	Obstacle's box x-position
DFC	[px]	Obstacle distance from center of image
$FOV_{\mathtt{cam}}$	[deg]	Webcam's field of view
$deg_{\mathtt{obj}}$	[deg]	Relative angle to obstacle

As can be seen in Table 3, the height of the obstacle is used in the calculations. This is because the height of the obstacle is larger than its width, therefore making the calculations less prone to minor pixel errors.

In order to calculate the distance to the obstacle there needs to be a reference value that is stored on the computer. This reference dictates how high the obstacle is within the image when it is placed a known distance, $L_{\mathtt{ref}}$, away from the webcam. It is this value that the height of the obstacle in all images will be compared to in order to figure out how far away the obstacle is relative to the webcam. The L_f value needs to be calculated only once for a certain camera/obstacle setup. The value is calculated as in (2).

$$L_f = h_{\mathtt{px,ref}} \cdot L_{\mathtt{ref}} \tag{2}$$

After this value has been calculated, the distance to the obstacle in the images taken thereafter can be calculated as (3).

$$d_{\mathtt{obj}} = \frac{L_f}{h_{\mathtt{px}}} \tag{3}$$

In order to calculate the relative angle between the obstacle and the webcam (and therefore the vehicle), the x-position of the obstacle's center on the image is used. Due to the obstacle's x-position being relative to the leftmost part of the image, the distance between the center of the image and the obstacle center x-position is calculated as in (4).

$$DFC = B_{\mathtt{c}} - \frac{W_{\mathtt{im}}}{2}. \tag{4}$$

As can be seen, this value is positive if the obstacle is on the right-hand side of the image. To convert this value from pixels to angles, the webcam's field of view needs to be known. The conversion is done as in (5).

$$deg_{\mathtt{obj}} = \frac{DFC}{W_{\mathtt{im}}} \cdot FOV_{\mathtt{cam}} \tag{5}$$

4.2 Distance - obstacle height correlation

The calculated distance in the previous section was calculated by taking the fraction of the current obstacle height, measured in pixels, and the reference obstacle height which is also measured in pixels. This fraction multiplied by the reference distance is the resulting current distance to the obstacle. The assumption made here is that there is a linear correlation between the measured obstacle height and the distance to the obstacle, i.e. if the measured obstacle height in the image is half the reference height, the distance to the obstacle is twice the reference distance. This does not necessarily need to be the case. The true correlation between these two parameters can be obtained by polling several data points and plotting them. By doing this study with the webcam mounted on the vehicle, the result in Figure 7 is obtained.

As seen in Figure 7, the distance cannot be approximated linearly as previously done without getting large errors in the distance measured. This can, for example, be solved by creating a lookup table and using linear interpolation on the data points in Figure 7. The trick done here to solve this discrepancy is to instead adjust the parameters in the regulator to match the imperfections obtained when approximating the distance linearly. When the regulator is to react to a real distance of, for example, 50 centimeters, but the measured distance at this real distance is 35 centimeters, the condition to react is instead changed to 35 centimeters. This fix is possible since the regulator does not use the measured distance continuously, but instead measures it continuously and it triggers activities once the value becomes small enough.

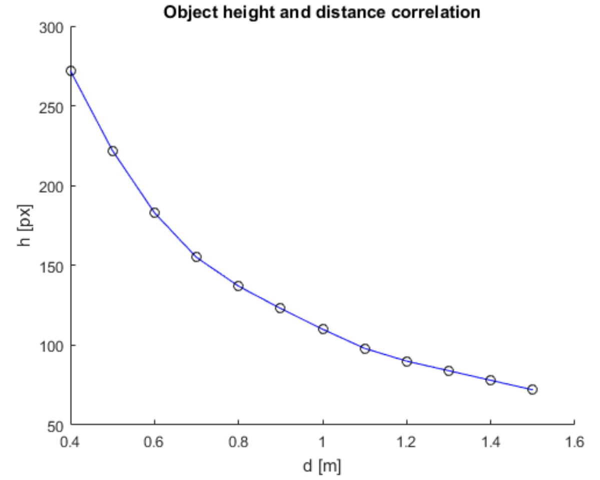


Figure 7. Distance and obstacle height correlation.

4.3 Filtering calculated values

Sometimes other obstacles of the same color in the image will dominate over the obstacle and the distance calculation will be done on this obstacle instead. This results in the value of the distance (and the relative angle) to the obstacle to change to a significantly different value compared to the calculated distance from the last frame. These large jumps in the measured values would cause major fluctuations in the regulator output signal. To counteract this, a specific type of filter is implemented on the calculated values of the distance and the relative angle.

When an image is taken, and the distance and angle is calculated, the value of the distance is stored in memory. When the next image frame is taken, and the distance is calculated, the new value is compared to the last value of the distance. If the difference in these distances are unrealistic (recall that the system is assumed to be running at 30 Hz, and with a relatively slow vehicle the distance between the obstacle and the moving vehicle cannot change that much between two frames) the new distance and angle calculations are ignored, and a new image is taken.

During start-up, it is important that the first calculation of the distance and relative angle is made on the obstacle and not on a disturbance. Therefore, a range of acceptable distances is defined. The vehicle needs to be placed this far away from the obstacle to ensure that it will lock-on to the obstacle. If the distance is shorter or longer than within the defined range, a new frame is taken and the old distance calculation is never stored. Once within the acceptable range, the calculated distance is stored and both it and the calculated relative angle

can be used as an input to the regulator.

A different approach to filter incorrect image frames and to achieve smoother measurement input to the regulator is to use a Kalman filter. This is not implemented during the project but is a possible improvement in the future. A Kalman filter is an algorithm that uses a series of measurements and estimates the next value of parameters with consideration to statistical noise and other inaccuracies [2]. The basic implementation of this in the project is that the four defining parameters of the box (height, width, x-position and y-position in the image) are modelled as having their second derivatives set to zero. The model is then ZOH-approximated (Zero-Order Hold). ZOH is a mathematical model that reconstructs sampled digital signals to analog signals. By approximating a constant change of these parameters as the vehicle moves, an estimation and an error interval of the next values of the parameters can be calculated. The incorrect measurements can then be filtered by ignoring the results that gives parameters outside of this error interval.

5. UDP Communication

There are two different ways of communicating between the PC client and the vehicle host, these two are the TCP and UDP protocols. The TCP protocol is a slower but a safer method of sending data packages in the sense that the sender after each sent data pack reassures with the receiver that the package has been correctly received. With the UDP protocol communication, data packages are continuously sent to the receiver. If the receiver fails to correctly receive a data package it ignores that package and instead continues onto the next. This results in minimum delay in the communication between the vehicle and the PC. Since in this project the data that will be sent from the PC to the vehicle will contain simple instructions of the same format, and the frequency of these instructions define the regulator speed, the quickest possible communication is desirable. This means that the communication between the vehicle and the PC is achieved using UDP protocol.

During the start-up phase in the program, the communication is set up. The client refers to the PC and the server refers to the vehicle. First, the IP-address of the vehicle and an arbitrary server port number is defined. The server port number must be the same as the client port number in order to make the communication work. The common rule of thumb for the port number is to pick ports between 1024 and 10000. If the port is occupied, try another.

Commands are sent to the vehicle using a function with attributes such as the steering angle, velocity and the UDP socket. The commands consist not only of information of how the vehicle should drive but also a signal declaring whether the vehicle should be running at all or standing idle.

6. Control

As can be seen in section 3., the initial regulator is a simple distance-dependent P-regulator. This simple regulator is enough to control the vehicle satisfactory up until the point where the obstacle is too near to the vehicle to be detected by the camera. The reason for using this simple controller is that simple functional solutions to a problem are good to implement initially before more complex solutions are explored.

Due to the process being quite slow, an integration part in the regulator would cause an integration during a long period before the error would change sign. If an I-part was used in the current regulator there would be an initial overshoot of the value. The reason for not applying this in the current regulator is that the P-regulator works sufficiently.

The reason for not having any D-part in the used regulator is that there are no significant overshoots in the controlled system that need to be hindered. The simple P-regulator causes the vehicle to turn in a decent manner. Furthermore, a slight overshoot is acceptable.

When using only a P-regulator there will be a stationary error. This, however, is acceptable due to the regulated value being the relative angle between the vehicle and the obstacle. The small stationary error only means that the obstacle will be passed with a slightly different angle between the obstacle and the vehicle.

7. Result

In the final implementation the initial controller had to be modified. In the model, the distance and relative angle to the obstacle is assumed to be determinable at all conditions, this does not apply to the real solution with a camera. The field of view of the used camera limits how close to the obstacle the vehicle can be before the obstacle covers the entire image. The field of view also determines the maximum relative angle to the obstacle before the obstacle disappears from the camera's sight completely. The final controller therefore needs to, at some point during the avoidance sequence, switch over to a mode where no inputs are used. In the final implementation this is the hard-coded sequence.

The hard-coded sequence is triggered upon reaching a predetermined distance to the obstacle. This distance was experimentally found to be 48 centimeters and represents the distance to the obstacle at which the obstacle almost entirely covers the height of the image feed and therefore the calculations become unpredictable. Once the hard-coded sequence is triggered a timer is started. For a certain amount of time the vehicle is commanded to steer away as much as possible at a certain direction (this direction being away from the obstacle), then for a certain amount of time to steer in the opposite direction in order to correct itself to get back on the vehicle's initial heading.

The complete avoidance sequence follows the following steps:

1. approach the obstacle as long as the obstacle is far away,
2. strive to keep a relative angle to the obstacle until reaching a predefined distance to the obstacle,
3. initiate hard-coded sequence of completing the avoidance maneuver and getting back on the initial vehicle heading,
4. go straight.

In the final implementation the vehicle can steer around the obstacle on both sides. The direction of the avoidance maneuver is determined upon approaching the obstacle. Once the

vehicle initiates step 2 in the sequence above, the relative angle to the obstacle is sampled and depending on the angle's sign (the relative angle value has different signs whether the obstacle is to the left or to the right of the vehicle) the vehicle either performs step 2 and step 3 by avoiding the obstacle on the right-hand-side or in a mirrored sequence on the left-hand-side of the obstacle.

8. Discussion

The vehicle does not always need to perform the sequence listed in section 7. As mentioned in section 3., the need for any avoidance sequence is specified by the initial orientation of the obstacle relative to the vehicle. If the vehicle is far away and the relative angle to the obstacle is big enough, the vehicle could just drive straight forward in its initial heading and still avoid the obstacle. This was implemented during the project but was not included in the final implementation. The implementation consisted of the program constantly checking whether the distance from the obstacle and the relative angle to it was large enough. If the conditions were met at any point, the control would be overridden by a command to only travel forward and ignore the obstacle. The implementation was quick, short and initially seemed to work. The issue with this, however, is the steering characteristic of the LEGO vehicle. If the LEGO vehicle is commanded to steer straight forward, the actual resulting steering would be dependent on the previous set steering angle. If the previous steering angle was left, the offset once commanding it to steer straight would be a bit to the left. If the previous steering angle was right, the offset would lean to the right instead. The issue here is how the physical vehicle is built. This issue could be resolved by a more robust construction with less or no play in the steering.

Further issues with the LEGO vehicle were the large delays in the communication between it and the computer controlling it. The control signals were calculated from the image feed quickly, but there was a delay varying between 0 and 3 seconds from when the signal was sent to the vehicle until the motors responded. The issue was never resolved but the fault occurs most likely between the EV3 and the Wi-Fi USB module or in the EV3 itself. There are already noticeable delays when remotely connecting to the EV3 from the computer.

A different approach to the regulation methodology was realized during the late stages of the project. Instead of measuring and keeping a certain relative angle to the obstacle depending on the distance, the distance to the obstacle once the vehicle is just about to avoid (parallel to) the obstacle can be the regulated value instead. When the vehicle is far away, the vehicle is instead to strive to be a certain distance away from the obstacle once passing it. By using the relative angle to the obstacle and the distance to the obstacle, the current distance during the avoiding maneuver can be calculated and regulated so that the vehicle instead passes the obstacle with the chosen distance.

The webcam used plays an essential role in how good the avoidance sequence is. The frame rate is the sampling rate and determines the control stability. The resolution of the image feed also increases accuracy in the distance calculation due to more pixels representing the obstacle. The resolution can however not be too high due to the increase in required computing power to perform the image processing sequence. The field of view of the camera is of utmost importance for the performance of the avoidance sequence. The wider the field of the view, the more can be seen in both height and width. The observable width plays a significant role during the late stages of the avoidance maneuver because this parameter alone determines how far into the sequence the obstacle can still be detected. The wider the field of view, the more the controller can see. This leads to the idea of using a 360-degree camera. By being able to see in all directions at all times, the problem is significantly easier due to the availability of information about the obstacle at all times. The controller in the MATLAB could theoretically be implemented as is with a 360-degree camera. Instead of a 360-degree camera, a 360-degree LIDAR sensor could be used. This eliminates any need for image processing and any errors that occur during the image processing sequence. The LIDAR sensor can also determine the distance to any obstacle without any beforehand knowledge of the obstacle dimension and it can also detect not only multiple obstacles but the entire physical environment.

References

[1] Marcus Greiff. *Notes on kinematic models for the 2018 project groups in frtn40*. 2018.

[2] T. Hägglund. Institution of Automatic Control, Lund University, 2015.

[3] R. Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011. ISBN: 9781461414322.

[4] Felipe Roza, Vinicius S. Ghizoni, Patrick J. Pereira, and Douglas W. Bertol. "Modular robot used as a beach cleaner". *Ingeniare* **24**:4 (2016), pp. 643–653. ISSN: 0718-3305.

A MATLAB Simulink Model

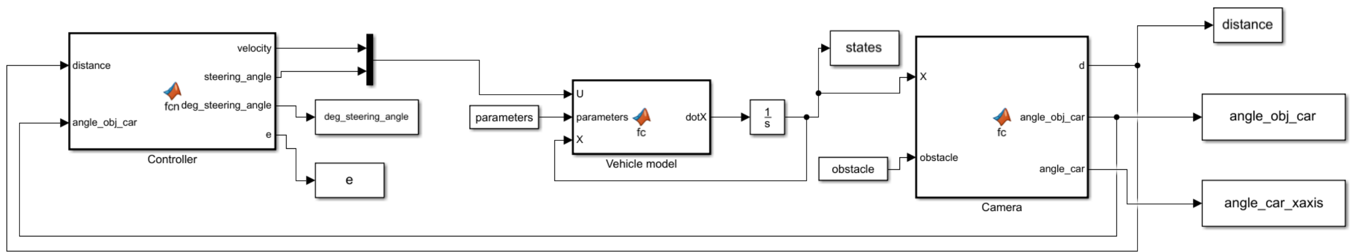


Figure 8. Simulink model of the vehicle and the obstacle.

B Written code summary

The software written in the project are two Python scripts, one that is to be run on the client (PC) and one on the host (vehicle). These two scripts can be seen as two threads and the communication between these is implemented by using sockets. A short description of the code is written here, for further details please open the Python scripts as they are thoroughly documented.

client.py: this code is to be run in the Terminal on the client PC. It opens a separate window showing the video feed of the chosen camera. The window also displays the distance to and the outline of the obstacle if an obstacle was found. The obstacle needs to be: red, green or yellow. A communication via sockets is opened to the EV3 vehicle, the server code should be running before the client is started. The user can write commands in the Terminal once the client code is running and appropriate commands are then sent to the EV3 vehicle. There are different modes of control, automatic control or manual override. For the automatic control mode, the input from the camera is used in a regulator and the outputted control signals are then sent to the EV3. For the manual control mode a signal is sent to the EV3 every time the user inputs a command. Some of the main functions are listed below.

- `setup_socket()`: Sets up the socket communication with the EV3 vehicle
- `regul(distance, angle)`: The used regulator. If the obstacle has not been passed and if the vehicle is far away: steer towards target. If the obstacle has not been passed but the vehicle is close: keep constant relative angle to obstacle. If the obstacle has not been passed and vehicle is even closer that the camera is about to lose the target on its view: trigger a hard-coded avoidance sequence.
- `user_input()`: Listens to user input in the Terminal. If acceptable command is read, set the internal variable to that order.
- `send_command(steering_angle, UDPCLIENTSOCKET)`: Sends the command type stored in an internal variable, and the steering angle to the corresponding socket.
- `find_contours(image)`: Converts raw RGB-image into three vectors containing found contours in the image, one for every searched color (red, green, yellow).
- `find_biggest_marker(contour_red, contour_green, contour_yellow)`: Finds best obstacle match from the three different candidates.

server.py: this code is to be run on the EV3. The vehicle is assumed to have one motor for steering and one for thrust. The server listens to commands from the client socket and consequently sets values on the two mounted motors. Some of the main functions are listed below.

- `setThrust(speed)`: Sets speed on thrust motor.
- `setSteering(angle)`: Set fixed angle on steering motor.
- `readSetSteering(message)`: Attempt to find angle from message and set it on the steering motor if it was found.

C External libraries and software version

The operating system on the LEGO EV3 is ev3dev-jessie. The code is executed in Python version 2.7.6 and the version of the external library of functions OpenCV is 2.4.8.

ev3dev-jessie available at: https://www.ev3dev.org/downloads/ as of 20-12-2018.
Python 2.7.6 available at: https://www.python.org/downloads/ as of 20-12-2018.
OpenCV 2.4.8 available at: https://opencv.org/releases.html as of 20-12-2018.

Autonomous driving F1/10 car

Rebeca Homssi[1] **Albert Anderberg**[2] **Josiah Wong**[3]

[1]tpi14rho@student.lu.se [2]elt14aan@student.lth.se [3]jo7036wo-s@student.lu.se

Abstract: The purpose of this project is to construct an autonomous car that is able to overtake another car that is driving with a constant velocity on a straight path. One prerequisite is that a map of the track is has to be generated. One subgoal, which has been achieved, is to drive the car at a constant distance from the wall at a constant speed. Real-time LiDAR scans of the surrounding environment measure the current angle and distance to the wall which can then be used as an input signal to a P-controller controlling the steering servo. This works well for straight walls but are not robust enough for turning corners due to the relatively high speed of the car. For this purpose both a map and path planning algorithm must be implemented to determine how to successfully navigate corners. Using Hector slam a map has been constructed in which the car can localize itself, see Figure 1. However, larger maps become progressively distorted and while progress has been made to implement a particle filter to compensate for such odometry error a complete solution is yet to be achieved. A chronic setback during the project is incompatibilities with the different software packages used, and is a central reason that the main goal has not been reached.

1. Introduction

The automation of cars has become both an ubiquitous and heavily pursued topic today: seemingly every car company in the world currently is developing prototypes of self-driving car that can handle a wide array of situation on the road. To date, much development has been accomplished and there currently exists cars on the market that are partially self-driving.

In this project, the F1/10 car will be used to gain a deeper understanding of modern automation techniques. The aim of this project is to better understand the modeling, control, and design process necessary to transform a traditional automobile into a self-driving vehicle. The goal of this project is to create a controller and a path planning model that can navigate the F1/10 car through a known path while avoiding obstacles that can appear on the way. To be able to fulfill this goal multiple sub-tasks have been set up. The first task is to create a controller which will control the car's general trajectory to prevent collisions with the wall. The next task is to use the LiDAR sensor to get information of the environment. The modeling and design will also be based on feedback from the IMU.

1.1 Background and Prior work

In 1977 the first automated car that could track streets were launched in Japan. Since then, many companies and organizations has developed prototypes with different levels of automatic driving. The level of automation can be described on a 0-5 scale, where level 5 is a completely self-driving car with no need of human steering. But the development has gone slowly until year 2012 when parts of the United States began allowing testing of autonomous cars on public roads. Today there is over 100 self-driving cars on the public streets in the US [6]. In 2017 in Gothenburg, Sweden, Volvo Cars launched a project where 100 households were selected to use Volvo's new (level 4) self-driving cars in their daily life [3]. However, there is yet to exist a level 5 autonomous car within a commercial setting.

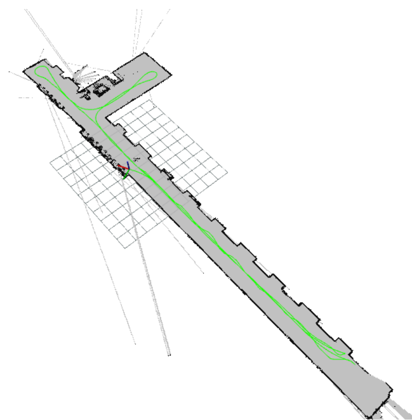


Figure 1. Map of a corridor in the basement in the M building using hector_slam package together with LiDAR.

Creating a self-driving car is a technical challenge. There are many sensors and control systems needed to be able to get accurate data from the surroundings. Typical sensors that are used is LiDAR, stereo vision, GPS and IMU. Based on the information from the sensors, the car must determine an accurate localization of itself and its surroundings and then create a localized trajectory to follow. An effective path planning algorithm tries to find the optimal balance between safety, speed, and efficiency from one point to another. There have been many studies on how to design the path planning model. In the master thesis "Motion Planning using Positively Invariant Sets on a Small-Scale Autonomous Vehicle," the authors used invariant sets to create a path planning model which could safely navigate a small car to overtake another car [1]. Further, in the article " A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles," a thorough

comparison has been made between 11 different path planning methods. The article shows that the different methods are well developed but should be used with a feedback controller that stabilizes the obtained path [4].

2. Modeling

Control of the car was modeled as a MISO system, with the two inputs being control signals sent to the speed controller and steering servo motor, and the single output being a weighted combination of the car's distance from the (right-hand side) wall and (scaled) angle relative to the wall.

While the final control system was relatively straightforward and required minimal structured modeling, it is important to note that this system's success was based upon a few critical assumptions, namely:

- The car is an inherently stable system
- Inputted voltage (PWM) is proportional to resulting speed
- Non-linearities related to steering may be neglected

The first assumption is necessary for the latter two assumptions to even be considered, for a stable system is much easier to heuristically tune compared to an unstable system. The second and third assumptions justify the usage of a simple PID controller, for the scope of this system does not require near-instantaneous response times that would necessitate a more complex, robust modeling of the car's steering (angular) dynamics.

3. Electro-Mechanics

The car is based on version 1 of the F1/10 car from the F1/10 project. Some of the more important parts can be seen in the list below and a short description can be seen in the following sections.

- Electric speed control
- Steering servo
- Teensy control board
- NVIDIA Jetson TK1 developer kit
- Hokuyo UST-10LX LiDAR
- SparkFun 9DoF Razor IMU M0

3.1 Electric speed control and steering servo

The car is based on the Traxxas 1/10th Car platform which is a normal remote controlled car. However, since this project's and the F1/10 project's goal is to control it from a computer some modifications has to be made. The standard configuration contains three different parts, the radio transceiver, the steering servo and the electric speed controller (ESC). In the default configuration the steering servo and the ESC is directly connected to the radio transceiver which generates the control signals. The control signals are pulse width modulated (PWM) signals where different duty cycle represents different steering angles and different velocities. A control board inserted between the radio transceiver and the actuators will generate the required PWM signals.

3.2 Teensy control board

The control board is based on the Teensy micro controller. Figure 2 contains the very simple schematic for the control board. The different three pin connectors all contain the same signal types: VCC, GND and a PWM signal. JP1 and JP2 are connected to the ESC and steering servo, respectively. JP3 and JP4 are connected to the original control signals from the transceiver. The switch S1 can be used to determine which control signal is used: either the one from the radio transceiver or the one generated by the Teensy board.

Not included in the schematic in Figure 2 is the USB connection between the Teensy and the Jetson board. To generate the PWM signals the 16-bit timers on the Teensy are used and the PWM signals are generated by the Teensy's hardware. The required control signals are received from the Jetson through the emulated serial port over the USB connection.

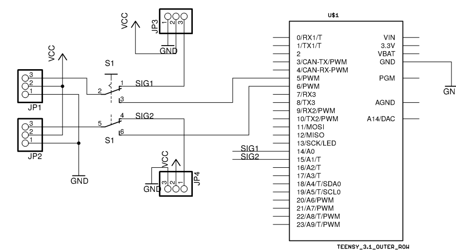


Figure 2. Electrical schematic for the control board based on the Teensy micro controller. Image courtesy of the F1/10 project, [2].

3.3 NVIDIA Jetson TK1 developer kit, LiDAR and IMU

The so-called "brain" of the car is the NVIDIA Jetson TK1 developer kit, hereafter simply referred to as Jetson. The Jetson runs Ubuntu 14.04 and ROS (Robot Operating System) Indigo. The IMU and the Teensy controller board are connected through USB to the Jetson. The LiDAR is connected to the Jetson through an ethernet connection. Since the onboard ethernet connection is already in use by the wireless access-point Ubiquiti PicoStationM2HP, a separate USB to ethernet adapter is used to connect the LiDAR. Since two ethernet interfaces are used they must be bridged so that the LiDAR is accessible from the rest of the network. This bridge runs and is configured in the Jetson.

4. Control

At the high level, the Hector SLAM algorithm leveraging the on-board LiDAR was utilized to localize the car while mapping its environment. On an implementation level, a simple single-channel PID controller was determined to be sufficient for stabilizing and maintaining the car's intended trajectory parallel to the wall. On the software side, the ROS platform was used to simultaneously run the path planning algorithms and trajectory controllers.

4.1 Software structure/design/implementation

Software in ROS is divided into different packages. These packages could be local packages written by the user or external packages written by the ROS community that has been downloaded either as a part of the standard ROS installation or separately by the user. A package can contain any number

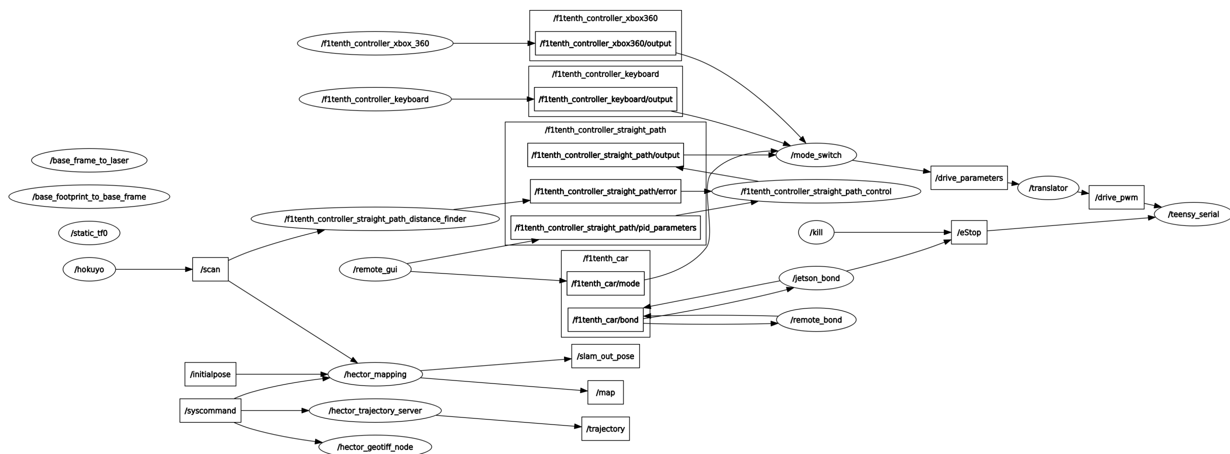


Figure 3. The graph contains a visualisation of the software structure and how the different parts are communicating with each other. The graph has been generated using the `rqt_graph` utility.

of nodes which contain the code that is running. A package can also contain launch files that describes how and when certain nodes should be run. A launch file is not limited to exclusively running nodes from the same package but can run any node that is available in the ROS environment. Communication between nodes are done using topics. A node can subscribe to a topic to receive messages published to this topic by another node. A message can contain different data dependent on which message definition is used. A simple message can consist of a single boolean, such as an on/off switch, but also more complicated data such as arrays containing distance measurements from a LiDAR.

ROS also has native network support which enables different nodes to be run on different hosts. To be able to use multiple hosts one specific host must be designated as the master. After setting the remote machines' reference ID to the master host, all nodes can then be run in the same way as though they were running on the same machine. However, there are some limitations when running on multiple hosts, such as hardware-communicating nodes must be running on the same host as the machine operating the hardware.

The code for this project can be divided into three different parts, which consist of multiple packages. The first part is the code running on the Jetson contains of all the nodes that interact with hardware and the majority of the controller nodes. The second part is the nodes running on the remote computer, such as a laptop. A graphical user interface (Figure 4), and visualization tools are included in this part.

The third part could be seen as an extension of the second as it only contains the emergency stop (e-Stop) code for the remote system. However, there is an advantage to keep this functionality as isolated as possible from the rest of the remote control since common safety protocol dictates that the e-Stop continues functioning regardless of the operating condition of the main remote/control software system. The best solution would be creating a hardware-based e-Stop solution that is completely independent of the remote host but such a solution requires additional hardware that is outside the scope of this project.

Using the `rqt_graph`[5] utility the image in Figure 3 can be generated when the software is running. Generally

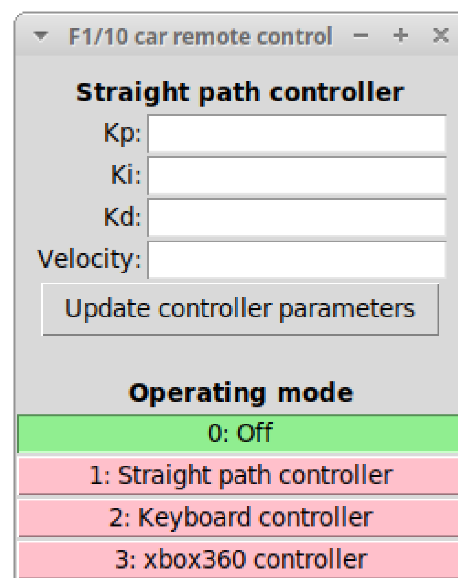


Figure 4. A screenshot of the graphical user interface used to control the car. The top half contains the functionality to set different parameters on the controller and the bottom half contains buttons to select the different modes.

data/information flows from left to right in the figure. At the far right side of the graph the `/teensy_serial` node can be found. This node is responsible for the communication with the Teensy control board and subscribes to the `/eStop` and `/drive_pwm` topics. The `/kill` node is emergency stop node running on the remote computer and publishes to the `/eStop` topic. The second publisher on the `/eStop` topic is the `/jetson_bond` node which runs on the Jetson. Together with the complimentary `/remote_bond` node, running on the remote computer, these nodes keep track of the connectivity between the different computers using a heartbeat signal. If the connection is lost the `/jetson_bond` node will engage the emergency stop by publishing the value true on the `/eStop` topic.

Going up the other branch we pass through the `/translator` node and find the `/mode_switch` node. The `/translator` nodes objective is to translate the control values

from the mode switch, which are in the range -100 to 100, to the values used by the Teensy board. The /mode_switch nodes objective is to, as the name suggest, switch between different operating modes. The different operating modes could be manual control, such as the /f1tenth_controller_keyboard node, or automatic control, such as the /f1tenth_controller_straight_path_control node. One mode that is not visible in the graph is the off mode which simply is used to turn off all actuators. The mode switch is controlled by the remote graphical user interface using the /f1tenth_car/mode topic. From the graphical user interface it is also possible to change the parameters for the straight path controller. The straight path controller is divided into two nodes, /f1tenth_controller_straight_path_control and /f1tenth_controller_straight_path_dist_finder, which together finds the distance to the wall using LiDAR data from the /scan topic and controls the car so that the distance to the wall is kept constant. The second subscriber to the /scan topic is the /hector_mapping node. This node together with the other hector nodes is used to build the map and current trajectory.

4.2 Sensors and measurement unit

One of the sensors used on the cars is the LiDAR sensor, which sends out pulsed laser light to measure distances. In the beginning of the project the LiDAR was used to measure the distance to a wall and allow the controller to maintain the car at a trajectory parallel and a set distance to the wall. Once achieved, the LiDAR sensor is to be used not only to measure distance to a certain wall but also to map the distances to walls and objects around the car. Using the data from the LiDAR, a map of the environment can be created. More about how this is done in section 4.3.

The measurement unit used is the inertial measurement unit (IMU). The IMU measures and reports the speed and acceleration of the car. While currently not implemented, the IMU will become central to sensor fusion when when the car is freely driving around in the basement or when the car is overtaking another car the IMU will become more important.

4.3 Hector - SLAM

Mapping out the environment while driving requires substantial computation time; however, since the the scope of the car's environment is limited to a specific building's basement a static map of that area may be created and used when driving. To be able to build up a graphical view of the environment measurements obtained from the LiDAR the hector_slam package in ROS is used. In this package exists three nodes, the hector_mapping, hector_geotiff and the hector_trajectory_server. The hector_mapping node uses simultaneous localization and mapping (SLAM) to learn the map. The hector_geotiff node saves the map and trajectories and the hector_trajectory_server saves the multiple coordinate frames over time, which in ROS are called tf.

The package initializes with a single scan of the environment and initial position. Then a threshold change in position must occur before the current scan is matched and a new position of the car is estimated using these measurement. Continuing this process a map of the environment will be obtained.

4.4 Particle filter

Now that a map has been obtained, the car needs to localize itself inside the map. This is done using a particle filter. The particle filter starts with an odometry pose and then adds some Gaussian distributed noise to get a set of possible poses. A LiDAR scan orientation is made and for each of the poses the correlation between the scan and the map from hector is computed. The pose with the highest correlation is chosen to be the position of the car. For each scan a position update is done.

4.5 Path Planning

A three-step process will be implemented to overtake another car. At a high level, it can be seen as follows:

1. Follow the other car at a set distance at the same speed
2. Determine the "safe distance from wall" threshold required to safely pass the car
3. If safe, execute a pre-designed trajectory
4. Check for changes in another car's (straight) trajectory, and abort if detected

The first step simplifies the path planning process by creating a standard "initial condition" upon which a specific trajectory planner may be designed. This will be achieved either by (a) a mounted webcam checking for a marker on the back of the other car, or (b) LiDAR detecting abnormalities within the pre-constructed static map of the environment. In either case, control signals will be sent to calibrate the car's orientation and distance relative to the other car.

The next step calculates the distance required to safely surpass the other car and avoid a collision with the forward-facing wall during execution. Given a pre-designed passing trajectory, it becomes a simple, relatively linear function proportional to the velocity of the other car.

Assuming the threshold distance value is met, the main car will execute a pre-designed trajectory to pass the other car. This is a justified decision, given the known dynamics and dimensions of the other car. This can be accomplished by providing localized and dynamic "waypoints" within the map for the car to follow and must pass with a certain velocity.

Lastly, because the path planner assumes the other car remains at a fixed velocity and orientation (i.e.: straight driving), the main car must check for either changes in the other car's angular position relative to its own as well as increases in the other car's velocity, either of which will result in an abortion of the intended passing trajectory and "resetting" of the Algorithm to Step 1.

4.6 Control

For each time step, a two-channel PID controller will be implemented to achieve the various trajectories required in the above algorithm. One channel will control the steering, while the other will control the velocity.

The error term in the steering controller is a combination between the distance to the wall and the steering angle. The error term is

$$e = -(y + L\sin(\theta))$$

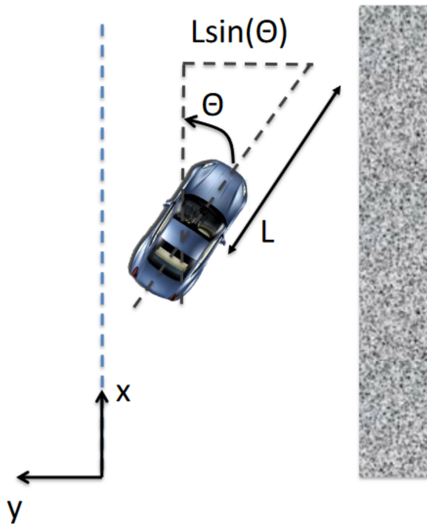


Figure 5. The graph contains a visualisation of the error term in the steering controller. As can be seen the origo is set at the desired distance to the wall. When $y = 0$ and $\theta = 0$ the car is driving on the right distance and parallel to the wall.

where y is the distance to origin (that is, the desired distance to the wall), θ is the angle of the car relative to the x-axis, and L is the distance the car will have travelled given the current speed. Tuning is yet to be done on both controllers. Because the voltage level input to a DC motor is proportional to its steady state speed, the PID velocity controller is expected to work with iterative tuning. Waypoints will be utilized to define the car's time-sensitive position and velocity states and serve as inputs to the controllers.

5. Results

In figure Figure 1 a map of the basement in the M-building can be seen. The map was obtained using the `hector_slam` package described above together with the LiDAR sensor. The car was driving with the PID controller along the long straight corridor trying to keep a distance of 1 meter to the wall on its right side. The parameters in (1) where used in the PID controller. During the turns and the shorter corridors the car was controlled manually. The green lines in the map is the driving path made when obtaining this map. In the figure, just slightly to the north-west of the center, the current position of the car can be seen. The car is represented as the blue and red lines. The red line is the x-axis and the blue line the y-axis whereas the origin is the LiDAR.

$$\begin{aligned} K_p &= 7 \\ K_i &= 0 \\ K_d &= 0 \end{aligned} \tag{1}$$

6. Discussion

As can be seen from the results the map obtained is quite accurate with the real world – the walls connect and appear reasonably in the right places. But as also can be seen there are some incorrect measurements. These errors are quite easy to detect since many of them point outside of the map. There are also some blurry points adjacent to the car which is due to a cabinet with glass doors. The glass doors allow some of the LiDAR particles to go through and others to reflect, based upon the angle of refraction. These reflected particles result in a blur of points in the map, and will ultimately affect the driving of the car. Proceeding further, such errors must be accounted for either by covering such glass objects or by defining a more robust sensor fusion system.

A P controller is used to control the steering of the car. Using this controller, the car is able to quickly find its own position, steer to the desired distance to the wall, and keep that distance. Therefore, it is unnecessary and potentially extraneous to implement a more advanced controller. However, it must be noted that the F1/10 website recommends a PD controller when the car is self driving in a race. Some K_d values was tested but did not result in any tangible improvements when driving in the corridor. The controller implementation in the car is still a PD controller and for each launch the user is able to chose the K_p and K_d values. The ability to change K_d was kept in the implementation to provide robustness for future controller development but in this project the value is kept to zero ($K_d = 0$).

6.1 Struggles

Working with this project a large amount of problems have been encountered. The largest difficulty was compatibility between the computers and the car since they were running different versions of Ubuntu. This resulted in large struggles to make all the ROS packages work as intended.

When driving with the car in the basement the thick concrete walls affected the signal strength. To avoid this problem the driving area were restricted to a small part of the corridor to not lose control of the car. Later, when the car can take corners and drive more autonomous this area can be expanded.

Another problem encountered was that the scans of the environment became worse when driving with higher velocities. Even using the smallest possible velocity gave a quite bad mapping. The bad mapping is due to the low frequency of the LiDAR scans. The problem was avoided by driving on the same path several times until a good map of the environment was obtained and then use this map when driving.

6.2 Further work

The powerful sensor combination of the already-implemented LiDAR and not-yet-utilized IMU coupled with the ROS framework and packages makes it easily possible to further develop the car substantially. Immediate and tangible improvements may be seen with implementing a successful particle filter or sensor fusion with the IMU to mitigate odometry error. On the software side, defining high-level goals and cost functions may prove to provide better robustness for a broader range of environments and applications. While unattainable given the time resources available during this course, overtaking another car is quite achievable and can be accomplished in multiple ways; for example, by leveraging either the LiDAR scans or adding a camera to then image process the car's immediate surroundings. On the control side, creating a more robust controller may become necessary as the scope of the car's application

increases. Multi-channel PID controllers or state-space/LQR controllers may provide a scalable and robust source of stable control should additional actuators be added to the car.

References

[1] R. Bai and K. F. Erliksson. *Motion Planning using Positively Invariant Sets on a Small-Scale Autonomous Vehicle*. eng. Student Paper. MA thesis. Lund University, Faculty of Engineering, Department of Automatic Control, 2018.

[2] *Car assembly (version 1)*. URL: `http://f1tenth.org/car-assembly` (visited on 2018-12-04).

[3] *Intellisafe autopilot technology*. URL: `https://www.volvocars.com/au/about/innovations/intellisafe/autopilot` (visited on 2018-12-03).

[4] B. Paden, M. Cáp, S. Z. Yong, D. Yershov, and E. Frazzoli. "A survey of motion planning and control techniques for self-driving urban vehicles". *IEEE TRANSACTIONS ON INTELLIGENT VEHICLES* **1**:1 (2016), pp. 33–55.

[5] *Rqt_graph*. 17, 2018. URL: `http://wiki.ros.org/rqt_graph` (visited on 2018-12-04).

[6] *Self-driving car*. URL: `https://en.wikipedia.org/wiki/Self-driving_car` (visited on 2018-12-03).

UAV Control

Sebastian Green[1] **Andreas Karlin**[2] **Lukas Stanger**[3] **Jonas Voigt**[4]

[1]sebastianpetergreen@gmail.com [2]bas11aka@student.lth.se [3]lu7710st-s@student.lu.se
[4]elt13jvo@student.lth.se

Abstract: The main goal of the project was to implement a position controller for a UAV, in this case a Crazyflie quadcopter. The controller was implemented in python using the Python API provided by BitCraze. The drone itself uses gyroscope and an accelerometer and combined with the LOCO positioning system this makes the needed measurements available by the use of a Kalman filter. The LOCO position system used six anchors in this project. The drone has its own controllers which controls the roll, pitch, yawrate and thrust of the drone, thus the outer controll loop for the position was to be implemented. Further goals included moving between several fixed setpoints and follwing a trajectory rather than step changes in the setpoints. The process was simulated while the basic software was implemented followed by tuning of the controller. The controller was tuned rather than calculated as this was assumed to be easier to get a good controller. Simulations were performed to get a estimate of appropriate values. The controller(s) was chosen to be of type PID or PD.

1. Introduction

The main goal was to be able to control the position of a unmanned aerial vehicle (UAV). The UAV was a Crazyflie quad copter and the controller(s) were implemented in Python. The measurements used to control the UAV came both from the IMU onboard and the local position system in the room, also provided by Bitcraze[1]. A PD controller were designed and implemented for every direction (X,Y & Z). If there was time left another controller could be designed and implemented, e.g an LQ-controller and possibly do a comparison between the different controllers.

1.1 Goals

The baseline objective was to reliably control and move the drone remotely from a PC client through keyboard commands or pre-planned routes. If time allows, the project could be extended to perform a fictional scenario such as finding a forest fire and dropping water, or moving in a specific pattern to avoid obstacles. Another possible scenario would be picking up a small physical package and dropping it in a mailbox.

2. Equipments and material

Both the Crazyflie and the additional equipment was from Bitcraze AB. The mentioned prices can be found on their website. [1]

2.1 Quadrocopter

- Crazyflie 2.0 ($ 225.00)

2.2 Additional equipment

- Crazyradio PA 2.4 GHz USB dongle ($37.50)
- Loco Positioning deck ($100.00)
- 6 x Loco positioning node ($187.50/each)

3. System Design

The control system consists of two separate platforms working in tandem. On the drone, a number of very fast (250/500Hz) cascading PID controllers are used to orient the drone according to certain setpoints for roll, pitch, yaw and combined thrust. These setpoints are in turn controlled by a regulator on the PC client. The drone also has an internal Kalman filter estimating its position and attitude. This estimator is based on a sensor fusion of a gyroscope and an accelerometer on the drone. Six loco positioning nodes are also set up and calibrated to function as anchors for the drone to relate its position to.

3.1 Drone Firmware

Bitcraze supplies a continuously updated version of the crazyflie firmware on their github, and this firmware provides an excellent base to work from. Adequate controllers for the attitude and thrust of the drone were already implemented, and thus no changes of the firmware have been made. Another added benefit is that a system for sending system state information and receiving control signals was already implemented through the motion commander and logging frameworks. This made interacting with the drone very straightforward and much of the headache associated with communication protocols and wireless connection were avoided. Logging was done by simply specifying which parameters should be monitored and at which rate. The drone then sends the requested information at regular intervals according to the rate.

3.2 Python Client

The client utilizes a number of modules to allow for easy access to control signals and measurements for tuning and testing. It provides a GUI for scanning for, connecting to and otherwise interacting with the drone.

GUI To simplify tuning and physical handling of the drone, a simple GUI was implemented. The GUI consists of three main windows each corresponding to a different aspect of the

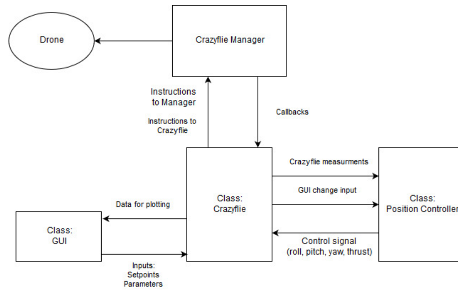


Figure 1. Python Client Overview

python client. A primary window shows buttons for scanning and connecting/disconnecting to the drone. This window also presents the user with the option to plot certain parameters and state information in real time. What parameters to show is specified dynamically by adding the relevant data to the log manager. The graph was configured to show the last 10 seconds of flight, and is accessible even when the drone is no longer connected. To save computation power, activation of the graph can be delayed until after the flight has concluded provided the operator is only interested in the last 10 seconds.

The parameter window was used to dynamically modify control parameters in flight. Tuning parameters used by the controller such as Kp, Ki and Kd for roll/pitch and thrust can all be modified and applied at the same time, making tuning much more efficient.

Another window is used for defining control references in flight. This is achieved by allowing the user to input an explicit position in the global X-Y-Z coordinate system. Changes are applied simultaneously, which makes it very easy to study not only step responses in a single direction, but also steps in multiple dimensions. If no position is specified for one of the dimensions, the currently active reference will remain active.

CrazyFlie Manager The Crazyflie managager is the automated communication between the drone and the python client. The manager is based on a callback system which will throw callbacks whenever a certain event occurs. These callbacks have to be implemented in the python client, which is done in the *Crazyflie* class. Notable examples of callbacks are *_connected*, *_disconnected* and *_connection_lost* which handle obvious important and frequent events.

The crazyflie manager also allows for the creating of a *logger*, which will request parameters from the crazyflie at a fixed interval. Once new data is acquired the callback *_stab_log_data* is called. As this is done with a fixed interval this will also be used as the main control thread in the program.

The Crazyflie class acts as the hub class for the python client acting as the interface between the manager, the GUI and the position control class. An overview of the classes can be seen in figure 1.

Controller The position control will be handled by a single class. The position controller will use three separate PID controllers using the library simple-pid [3]. These controllers take the current value of the system as input and ouputts the calculated control signal. Any parameters must be set initially but can be changed in real time. The Position Controller will be responsible for the conversion of quaternions into Euler angles. To guarantee mutual exclusion when addressing control parameters such as setpoints (reference values) and the PID parameters (K, Ki and Kd) python Lock objects from the threading library were used [2].

Figure 2. CrazyFlie Drone

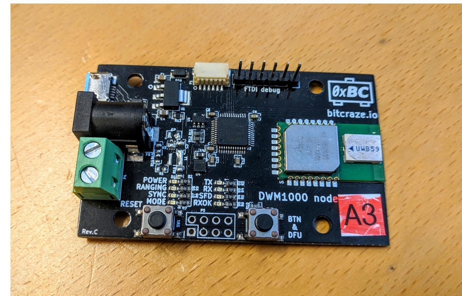


Figure 3. Loco Positioning Node

4. Modeling

So far, no modelling has been done. Premade Simulink models were provided by the supervisor to allow simulations of the crazyflie. These would give some insight into the craziflies behaviour.

5. Electro-Mechanics

The main component of this system is of course the CrazyFlie drone, mounted with a positioning board as seen in Figure 2. For positioning, we use 6 loco positioning nodes placed as the corners of a triangular prism. An image of a single node can be seen in Figure 3. Finally a CrazyRadio PA was used to communicate with the drone by connecting the radio to a USB port. The radio can be seen in Figure 4.

6. Control System Design

The control system which was implemented consists of a position controller cascaded with a attitude and thrust controller, an overview of the control system can be seen in figure 6.

The inner controller controls the attitude and the thrust of the quadcopter, it consists of four PID's which controls the thrust, roll, pitch and yaw of the quadcopter. The outer control loop controls the position of the quadcopter, i.e the position in X,Y and Z. To control the position three PID's have been designed, one for every direction. The PID was chosen since it

Figure 4. CrazyRadio PA

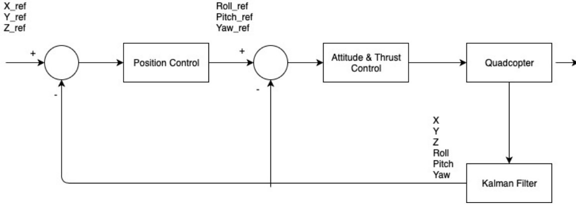


Figure 5. Control System

is easy to implement and relative easy to tune. The control law for the PID can be seen in equation 1. There is also a Kalman filter implemented by the manufacturer. This estimates the position of the drone respect to the global frame, its velocity in the body frame and the attitude in terms of quaternions.

To control the position of the quadcopter an control error is created between the desired position in each direction, e_x, e_y and e_z. These control errors needs to be remapped to angles representing the rotations around X,Y and Z axes, i.e roll, pitch and yaw, ϕ, θ, ψ. The control signals, u_x, u_y from position controller will then act as a reference signal to the attitude controller. The rotation matrix for this transformation can be seen in the equation 2. For the Z-direction the control signal u_z was sent directly as reference to the attitude controller.

$$u(t) = K_p e(t) + \frac{K_i}{T_i}\int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{1}$$

$$\begin{bmatrix}\phi \\ \theta\end{bmatrix} = \begin{bmatrix} sin(\psi) & -cos(\psi) \\ cos(\psi) & sin(\psi)\end{bmatrix}\begin{bmatrix}u_x \\ u_y\end{bmatrix} \tag{2}$$

The position measurement from the Kalman filter showed to be quite noisy, causing the time-derivative of the position error in the D-part of the controller to be large at times which resulted in bad performance. This was avoided by using the estimated velocity of drone instead and a reference velocity for the D-part instead since the velocity estimation is more smooth than the position measurement. However this velocity is measured in the body frame of the drone and needs to be remapped to the global frame. This was done with equation 3 which shows the rotation matrix from body frame to global frame which was used for remapping the drones velocities to the global frame. This matrix is the inverse of the rotational matrix generated from a unit quaternion. Note that this only applies when the rate of change for angles (or quaternions) is small. The rotational matrix is denoted R_v and is seen in equation 3 and the rotation estimation in equation 4.

$$\begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2+q_3q_0) & 2(q_1q_3-q_2q_0) \\ 2(q_1q_2-q_3q_0) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_1q_0) \\ 2(q_1q_3+q_2q_0) & 2(q_2q_3-q_1q_0) & 1-2(q_1^2+q_2^2)\end{bmatrix} \tag{3}$$

$$\begin{bmatrix} dx_g \\ dy_g \\ dz_g \end{bmatrix} \approx R_v \begin{bmatrix} dx_b \\ dy_b \\ dz_b \end{bmatrix} \tag{4}$$

6.1 Controller Implementation

For the implementation of the PID-controller a PID library have been used called `simple-PID`. It works by sending the current measurements as arguments to the PID-object and assigned the references to the desired value. The D-part is calculated with the velocity error instead of the time derivative of the position error. The estimation velocity which is described in the drones body frame is first remapped with the rotation matrix in equation 3 to the global fixed frame before it's sent to the D-part. Then the P and D part is summed together and later remapped into the angles roll and pitch according to equation 2. The procedure can be seen in figure 6. The last step shows an offset of 38000 and a scaling of 5000. This will be discussed in the tuning of the controllers in section 7.2.

6.2 Simulation

Simulations in Simulink was made to get a sense of the behavior of the quadcopter. Models were provided by the supervisor. In simulink the controllers were tuned although these settings would not apply on the real process. A trajectory planning was also implemented and tested in Simulink, more about this later. The tuning in the simulations was done by choosing a pole,p and then using the following relationship between the P- and D Part.

$$K_p = p^2 \tag{5}$$

$$K_d = 2p \tag{6}$$

6.3 Trajectory Planning

One attempt to a simple trajectory planning have been implemented in Simulink. Where the idea is to specify the initial point A and the final point B. Choose a velocity,v for the quadcopter that it should travel with and calculate the time T it will take to travel the distance between the two points A and B. The algorithm is shown in equation 7. This will give a ramp reference change rather than a step.

$$P(t) = A + \frac{B-A}{\|B-A\|}tv \tag{7}$$

The velocity and the direction from equation 7 was used as a velocity reference for the D-part in the controller. As the velocity becomes a pulse the velocity reference is low pass filtered to smooth out the edges, which is especially importan when reaching the setpoint, which can be seen in figure 7 . It is also given a "headstart" to make sure the velocity has already decreased sufficiently when the drone reaches the setpoint.

This method also makes it possible to make larger steps without getting too large angles and causing the drone to get unstable, since the step is divided by to smaller steps this way.

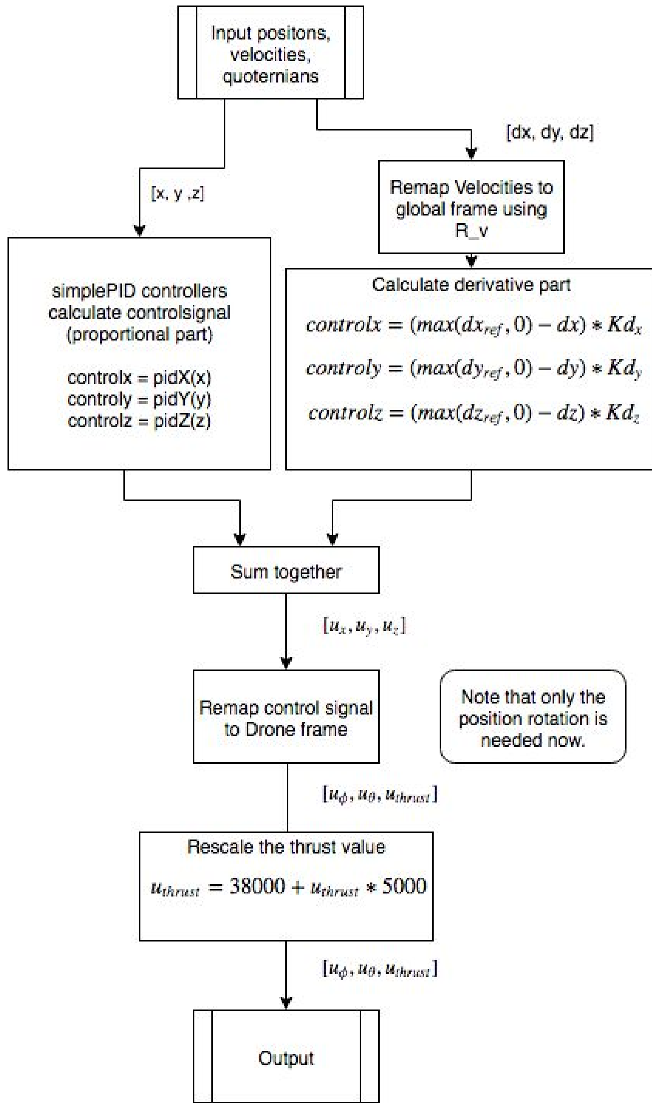


Figure 6. Flow from input values to output

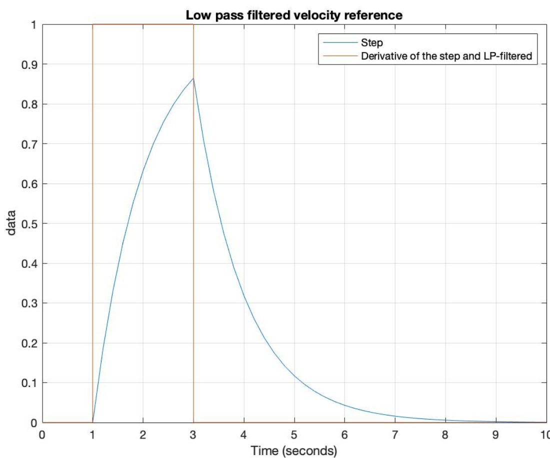


Figure 7.

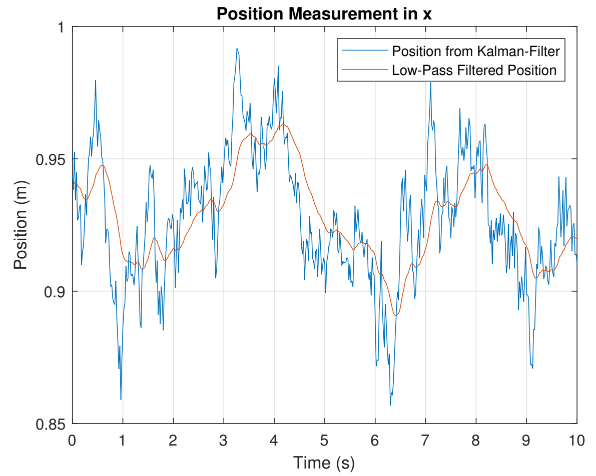


Figure 8. Low-pass filtering of position measurements

7. Results

In this section the results from the tuning and the results from flying attempts will be shown.

7.1 Position Measurements

As described above, the position signal we receive from the Kalman filter is quite noisy. Therefore we decided to apply the following first order low-pass filter to the position measurements:

$$y_k = ay_{k-1} + (1 - a)u_k \tag{8}$$

where u is the position estimate from the Kalman filter and y is the low-pass filtered position. To get the desired filtering of the measurements, the pole was placed at $a = 0.05$. Figure 8 shows the results of the filtering of the x position. While performing this experiment, the drone was lying on the ground without any movement. It is important to note here that we decided not to alter the tuning of the on-board filter, but rather attempt to filter at a later stage. As mentioned above in the case of the position-derived velocity being jittery, it is possible that a more stable filter could be achieved by also factoring in drone accelerations and velocities.

7.2 Tuning of controller

Although models for simulating the drone was available and gave valueble input on how the drone responds, these only gave rough estimations of suitable parameter values. A large amount of time was spent on tuning the controllers.

Before discussing the tuned parameters it is necessary to know that the drone accepts a control signal on the form of (roll, pitch, yawrate and thrust) where the thrust value needs to be in the range [10000 - 60000]. As the error is measured in meters, the gains of the controllers was intially quite large.

Initially z-position controller was tuned to get the drone of the ground. Once the drone was in the air the control signal would reveal a suitable offset value, an approximate minimal value for the drone to hover. Once the z-controller could hold the drone in the air the x and y controllers could be tuned to a working level. With all three controllers working to to

Param	pidX	pidY	pidZ
K	30	30	13
Ki	0	0	0
Kd	14	14	5

Table 1. PID parameters. Note that the I part is intentionally turned off

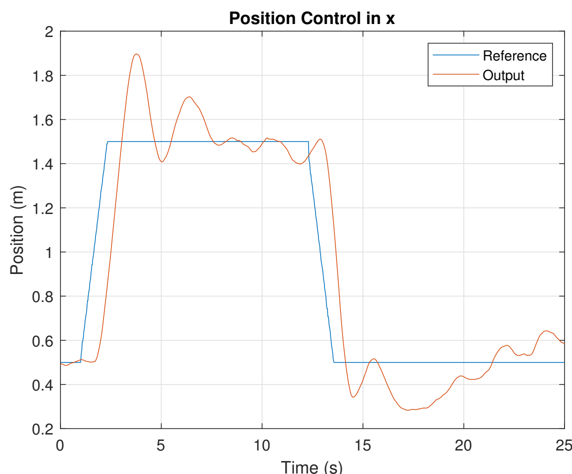


Figure 9. Step Response in x-direction

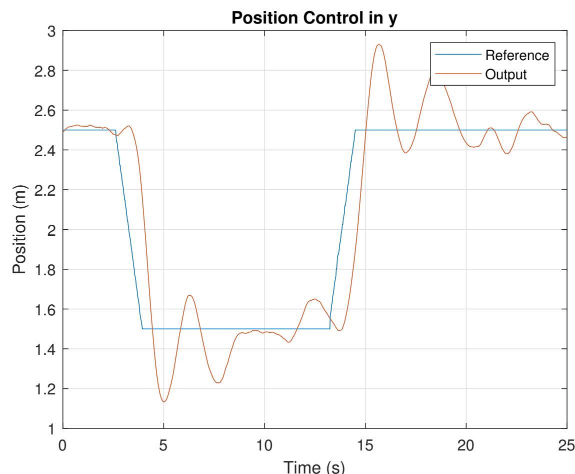


Figure 10. Step Response in y-direction

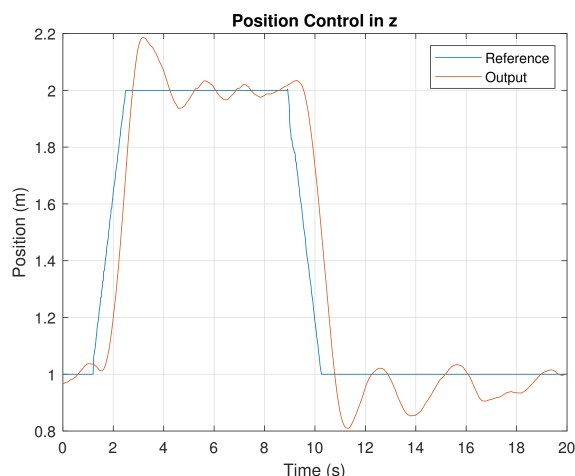


Figure 11. Step Response in z-direction

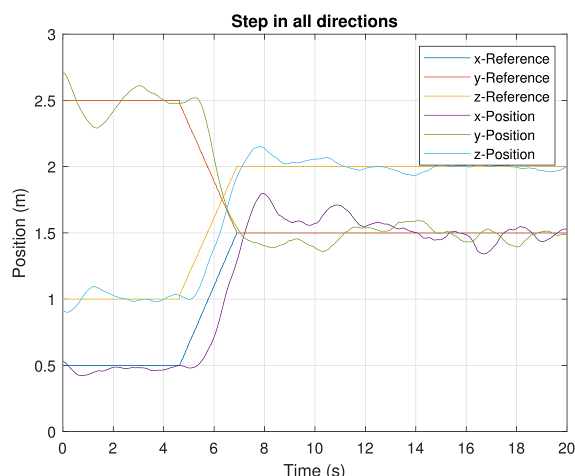


Figure 12. Step in x, y and z-direction

some extent they could all be further tuned to achieve better performance.

To reduce the magnitude of the controller parameters a secondary gain, or scaling, of 5000 was used to get the control signal of the thrust to be in the appropriate range. The offset was set to 38000 as this was the necessary value for the drone to hover.

The final parameters used for the controllers is seen in table 1. These were tuned as a tradeoff between the stability and speed for all three positions.

7.3 Drone Control

The controllers seen in the tuning section were able to stabilize the drone in the air. Due to large variations from position measurement it was hard to make drone hover in one point, this also affected the drones ability to track a desired trajectory.

To show the results of the designed controller, steps were performed in all directions, x, y and z, that can be seen in figure 9, figure 10 and figure 11.

Figure 12 shows a step performed in all directions at the same time, to see the coupling between the different positions.

Moreover we performed an experiment, where we sent several steps in x and z-direction to the drone, so that the drone's movement followed the shape of a rectangle. Every 4 seconds a new step was sent to the drone. The result can be seen in figure 13 where 4 loops was performed.

8. Discussion

The position measurements have offsets on them, for example the z measurements thinks 0.5 m is the ground. These can in regard to control performance be neglected although its necessary to remember it when observing graphs.

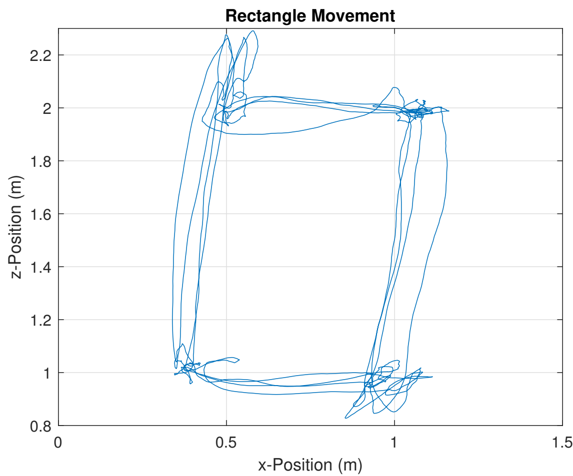


Figure 13. Rectangle movement in x- and z-direction

The noise on the position measurements seen in figure 8 can be seen as a great concern if an improvement in the controller is to be attempted. This project used a first order lowpass filter of the signal to reduce the effects. The choice of pole became a trade off between a high damping of the noise and the ability for the filter to be able to keep up once the drone was actually moving. This filter could have been improved with perhaps an increased damping due to another choice of pole placement or by implementing a higher order filter. The measurements based on the Loco positions system could have benefited from an increased amount of anchors.

The drone calibrates itself whenever it starts with the import note on the yaw angle which is defined in the direction the drone is pointing. Any deviation from the true (defined by user) yaw which should be in the direction of the xaxis will cause the drone to have a slight difference in its reference frame causing slow shifts.

As seen in figures 9, 10 and 11 its seen that each controller overshoots slightly indication to aggressive control. However, attempts to remedy any of these controllers resulted in the others becoming more unstable. Adjusting is probably possible but will require all adjusting all three controllers.

The rectangle movement seen in figure 13 shows a drift whenever rising and the opposite while falling. This could be due to the so-called ground effect. This means whenever close to the ground the drone will experience a floating effect. With the offset in z-values mentioned 1 meter in z position is actually 0.5 m. This may be close enough for the drone to start floating increasing the difficulty for the controller. At the higher values of z = 2 (or 1.5) the done reaches in x position much better (although overshooting on the z position). No deeper studies on the ground effect has been made so it cant be stated whether the symmetry of this floating confirms the result.

The ramp reference allowed the control to be much faster and allowed for larger step changes without the risk of instability and has been a great benefit. The ramp could possibly have implemented similar to the low pass filtering of the velocity reference by low pass filtering the step change.

9. Conclusion

The drone is working satisfactory although the controllers still overshoot to much and should be tuned more. Improving the filtering of the position measurements or supplementing them with additional input should be evaluated as they are key to a good position control.

The project hoped to evaluate other problems such as fictional scenarios, picking up packages. These have not been achieved or even attempted as the precision of the drone is not yet adequate.

In all the project group is happy with the result and the lessons learned during the project.

References

[1] *Bitcraze ab*. `https://www.bitcraze.io/`. Accessed: 2018-11-08.

[2] *Python lock objects*. `https://docs.python.org/3/library/threading.html\#lock-objects`. Accessed: 2018-11-20.

[3] *Simplepid*. `https://pypi.org/project/simple-pid/`. Accessed: 2018-12-04.

Autonomous parallel parking - LEGO truck and trailer

Wilhelm Andrén[1] **Ella Hjertberg**[2] **Henrik Ståhlbom**[3] **Eric Schyllert**[4]

[1]mas14wan@student.lu.se [2]mas14ehj@student.lu.se [3]he0275st-s@student.lu.se
[4]mat11esc@student.lu.se

Abstract: In this project, a process used to control a LEGO built truck and trailer with the help of a LEGO Mindstorms EV3 kit is developed. The project aim was to parallel park the vehicle using image analysis together with a control system. The image analysis was enabled using OpenCV for Java. The control system was created using a pure pursuit and PI controller, which parameters were calculated using Matlab and Simulink. This method along with some manual tuning, lead to a stable closed-loop control system with the theoretical ability to park the vehicle. The control system was implemented using the firmware leJOS that is supported by the EV3 kit. The embedded system was connected through sockets to enable communication between the EV3 brick and the computer. The result of the implementation enabled the trailer of the vehicle to follow a path into a given parking space. However, the whole vehicle did not park itself, only the trailer was able to go into the parking space. Therefore, systematic troubleshooting was made, together with discussion of possible improvements.

Further work would include making the whole vehicle park itself by straightening it up inside the parking space. This would most easily be done by hard coding the truck to drive with specific commands. This would eventually result in a parked vehicle.

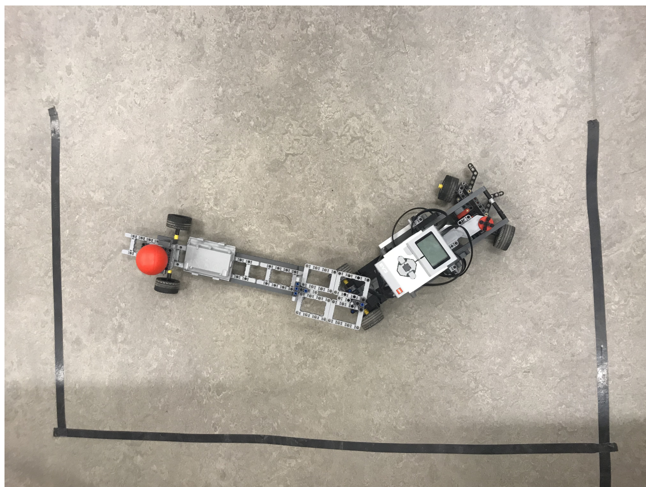


Figure 1. Picture of robot

1. INTRODUCTION

The aim with this project is to design, build and control a *LEGO Mindstorms* process. In this particular project, the process will be a LEGO Mindstorms EV3 truck with a trailer. The purpose of this project is to enable the vehicle to parallel park autonomously. The goal is for the truck to parallel park successfully, avoiding any obstacles such as other cars or the sidewalk. The truck will be built using LEGO Mindstorms EV3 available in the labs at the department and the software that is going to be used to implement the truck and trailer is the firmware Java leJOS together with Matlab and Simulink for the control design part of the project. A camera will be mounted above the truck to be used for image processing in the implementation. The image processing will be executed using the open source library *OpenCV*.

2. MODELING

2.1 Design

The truck and trailer are both built in *LEGO*. The truck has two sets of wheels, which are placed in the front and in the rear of the truck. Both of the wheel pairs are connected to one servomotor each. The embedded system, i.e. the EV3 programmable "brick" is mounted on top of the truck, in the center. The reason for this is to have the center of gravity as close as possible to the middle of the truck for smooth movements, as well as for aesthetic purposes.

The trailer has one set of wheels, located in the rear part of the construction, to resemble a "real world" trailer as much as possible. This set of wheels is not connected to any servo motor, thus not able to drive by itself. The trailer is fastened in the rear axle on the truck using a LEGO "cylinder piece" working as an axle to allow for the trailer to turn freely around the fastening point. All six wheels in the construction are of the same type, small, wide rubber standard wheels from LEGO. The truck along with the trailer are symmetrical along the central axis, with the four rear wheels equally wide apart. The front wheels are slightly wider apart, to enable for the front wheels to rotate with a greater angle. In the rear part of the trailer, above the wheels, a red ball is placed, to allow for object tracking via the image processing.

Motors The servo motors used for the wheel sets of the truck were included in the LEGO Mindstorms EV3 kit. The servo motor connected to the front pair of wheels, is placed in the middle of the two wheels, "horizontally", see Fig.2.

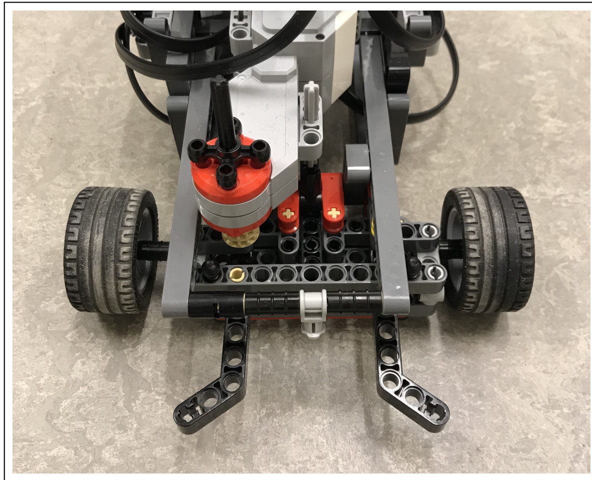


Figure 2. Front part of the vehicle

The motor is only attached to one of the wheels, but a set of gears connects the wheel motions. The motor is placed in the shown way to allow for the front wheels to turn. In this manner, it is possible to steer the truck to the left and right by rotating the wheel pair in a desired angle. Important to notice here is that the front wheels cannot drive the truck forwards or backwards, only steer the direction of the truck.

The rear pair of wheels also has a servo motor placed between them. This motor is not placed horizontally as the front one. The construction is the same in the sense that the motor is only attached to one of the wheels and connected to both of them via a set of gears. This motor allows for the truck to drive forward and backwards, and are therefore the driving wheels of the vehicle construction.

Sensors An angle sensor is mounted on the rear axle of the truck, to allow for angle measurements between the truck and the trailer. The trailer is fixed with an axle in the angle sensor, which enables the sensor to read the angle of the trailer in comparison to the truck.

The camera used for image processing acts as an external source, and is mounted above the entire construction, in the roof of the lab used as demonstration location. The webcam used is a *Logitech c270* with a maximum capacity of 30 FPS [2]. The group believes that lower a FPS will be sufficient for this application, why the chosen camera is considered to fit the purpose.

Parking space The parking space designed for the process was chosen to be the double length of the trailer, which leaves enough margin for the entire vehicle to fit in the space. This decision was based on the relative size of a life-size parking space for a truck with a trailer. The parking lot was created using black tape that was fastened on the floor in the lab.

3. CONTROL

To enable the vehicle to parallel park without any human interference, the process must be controlled. The accessible variables available from the vehicle are the angle between the truck and trailer, the reversing speed and the front wheel rotation angle. Furthermore, the position of the trailer can be retrieved through image processing, where the red ball is tracked. Additional items needed in order to park are

- a path for the vehicle to follow into the parking space
- a control system that compels the vehicle to stay on this path

In order to be able to design the control system, the dynamics of the vehicle had to be derived.

3.1 Mathematical model

A nonlinear model for the truck with trailer is derived with the inspiration from a previous similar project [3], and a master thesis covering the topic of a reversing truck and trailer [4]. The mathematical model is derived based on the assumption that the truck has rear-wheel drive and front-wheel steering while the trailer has no inputs, and is completely controlled by the truck. Also, rolling slip of the wheels is disregarded in the model. A graphic representation of the truck with the trailer is shown in Fig. 3 with the denotation of the variables shown in Table. 1.

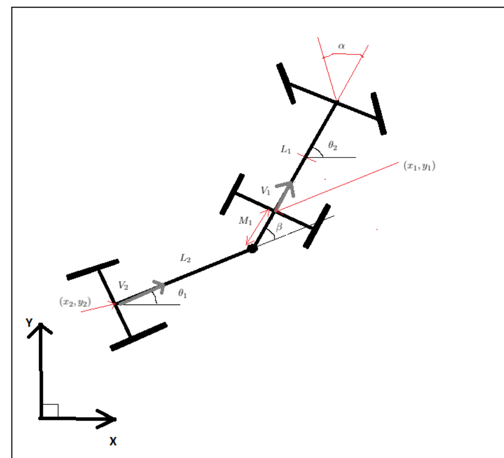


Figure 3. Graphical representation of the mathematical model of the truck with trailer

Equation (1) through (9) shows the derivation of the variables shown in Fig. 3. The direction of the trailer is controlled by the angle between the pivot point that connects the truck and trailer, β. The dynamics of this angle can be seen in (10) and is a nonlinear differential equation. Since this angle is measurable by the angle sensor placed on the truck, the main idea is to control this angle. This would enable controlling the path leading into the parking space. In order to control a nonlinear system one can either use a nonlinear controller or linearize the system around a stationary operating point. The second option - linearization - is the chosen method for this project due to simplicity and satisfying control.

$$\dot{\theta}_1 = \frac{v_1}{L_1}\tan(\alpha) \tag{1}$$

$$\dot{x}_1 = v_1\cos(\theta_1) \tag{2}$$

$$\dot{y}_1 = v_1\sin(\theta_1) \tag{3}$$

Table 1. Definition of parameters

Variable	Meaning
θ_1	Global angle of truck
θ_2	Global angle of trailer
(x_1, y_1)	Global coordinates of the rear axle of the truck
(x_2, y_2)	Global coordinates of the rear axle of the trailer
L_1	Length between front and rear axles of the truck
L_2	Length between front and rear axles of the trailer
v_1	Velocity of the rear axle of the truck
v_2	Velocity of the rear axle of the trailer
M_1	Length between the rear axle of the truck and the connecting point between the truck and the trailer
β	Angle between truck and trailer, $\beta = 0$ meaning perfect alignment
α	Steering angle of the front wheels

$$\dot{\theta_2} = \frac{v_1 \sin(\theta_1 - \theta_2)}{L_2} - \frac{M_1 \cos(\theta_1 - \theta_2)\dot{\theta_1}}{L_2} = = v_1(\cos(\beta) + \frac{M_1 \sin(\beta)\tan(\alpha)}{L_1 L_2} \tag{4}$$

$$v_2 = v_1(\cos(\beta) + \frac{M_1 \sin(\beta)\tan(\alpha)}{L_1}) \tag{5}$$

$$\dot{x_2} = v_2\cos(\theta_2) = = v_1\cos(\theta_2)(\cos(\beta) + \frac{M_1 \sin(\beta)\tan(\alpha)}{L_1} \tag{6}$$

$$\dot{y_2} = v_2\sin(\theta_2) = = v_1\sin(\theta_2)(\cos(\beta) + \frac{M_1 \sin(\beta)\tan(\alpha)}{L_1} \tag{7}$$

$$\beta = \theta_1 - \theta_2 \tag{8}$$

$$\dot{\beta} = \dot{\theta_1} - \dot{\theta_2} = = \frac{v_1}{L_1}\tan(\alpha) - v_1(\frac{\sin(\beta)}{L_2} - \frac{M_1\cos(\beta)\tan(\alpha)}{L_1 L_2}) \tag{9}$$

$$\dot{\beta} = \frac{v_1}{L_1}\tan(\alpha) - v_1(\frac{\sin(\beta)}{L_2} - \frac{M_1\cos(\beta)\tan(\alpha)}{L_1 L_2}) \tag{10}$$

With linearization of the system (10) around $(\beta, \alpha) = (0{,}0)$ the linear system (11) is achieved, which corresponds to a linear state space model

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

where the input, x, to the process is the angle β and the output, u, is the front wheel angle, α.

$$y = \beta$$

$$\Delta\beta = \beta - \beta_0$$

$$\Delta u = u - u_0$$

$$\Delta y = y - y_0$$

$$\begin{cases} \Delta\dot{\beta} = -\frac{v_1}{L_1}\Delta\beta + \frac{v_1}{L_1}(1 + \frac{M_1}{L_2})\Delta u \\ \Delta y = \Delta\beta \end{cases} \tag{11}$$

The open transfer function of system (11) can be seen in (12) which is unstable for $v_1 < 0$. A negative velocity means that the vehicle is reversing which is intended for this project purpose. The aim is to always reverse with a constant speed of $v_1 = 0.1m/s$. The physical model is designed in such a way that M_1 is very small and will therefore be approximated to zero.

$$G_p(s) = \frac{v_1}{L_1}\frac{1}{(s + \frac{v_1}{L_1})}(1 + \frac{M_1}{L_2}) \tag{12}$$

3.2 PI Controller

The linearized system (11) has a transfer function (12) of relative degree 1 (no zeros and one pole), which means that the derivative part of the controller is unnecessary. The requirement of the controller is to be able to follow a step response well with small overshoot - thus a PI-controller is chosen. By introducing a PI-controller together with the open transfer function of the linearized system (12) the closed loop transfer function (13) and (14) is achieved. By placing the poles at $s = -1.1 \pm 0.3i$, the controller values $K = -6.33$ and $T_i = 1.3$ are obtained - an initial guess with requirement on stability.

$$PI(S) = \frac{KsT_i + K}{sT_i}$$

$$G_{cl}(s) = \frac{G_p(s)PI(S)}{1 + G_p(s)PI(S)} \tag{13}$$

$$s^2 - s(\frac{0.1}{L_2} + \frac{0.1K}{L_1}) - \frac{0.1K}{L_1 T_i} = 0 \tag{14}$$

3.3 Path planning

The trajectory planning is generated using an interpolation method and stays fixed throughout the process. This is in order to facilitate the control system implementation. The path is generated through a set of assumptions: at time zero the truck and trailer are almost fully aligned with respect to both themselves and to the parking space, and the parking is performed on the right hand side of the road, seen from the perspective of the vehicle. Both of these assumptions could be ignored with a more generalized parking program, but for this project time has been a deciding factor in some decisions. Starting out simple and getting it working was the first goal, and if time allowed it more ambitious solutions would be considered. The interpolation method chosen was Cubic Hermite interpolation as it has shown to be sufficient for this type of path generation in earlier projects [3].

The path planning is one of the first things that should be created in the implementation. To be able to create a spline from the vehicle into the parking space, the coordinates of both the trailer and the parking space are needed. The path is

a discrete valued spline created with Cubic Hermite interpolation where the values represent coordinates. The end points of the interpolation are the rear axle of the trailer and a point at the far end of the parking space. The assumption that the trailer is parallel to the parking space allows the interpolation to be performed with tangent values of zero at both end points, as this represents a parallel to the parking space.

The Cubic Hermite Spline has the general equation (15)

$$p(t) = h_{00}(t)p_k + h_{10}(t)(x_{k+1} - x_k)m_k + h_{01}(t)p_{k+1} + h_{11}(t)(x_{k+1} - x_k)m_{k+1} \tag{15}$$

where

$$t = (x - x_k)/(x_{k+1} - x_k)$$

$$h_{00} = 2t^3 - 3t^2 + 1$$

$$h_{10} = t^3 - 2t^2 + t$$

$$h_{01} = -2t^3 + 3t^2$$

$$h_{11} = t^3 - t^2$$

This equation creates a polynomial with the start and end points at the coordinates (x_k, p_k) and (x_{k+1}, p_{k+1}) respectively. The variables m_0 and m_1 are the respective tangent values, and h_{ij} are the Hermite basis functions.

Depending on the starting position, some problems may occur if the generated path is not considered, as it then could cross over the boundaries of the parking space. To prevent this the distance to the closest corner of the parking space on each path point is checked. If a point that is too close to the parking space boundaries is found, an extra point is generated. This point lies on a large enough distance on the line crossing the two points from the corner. The interpolation is then performed again with regards to this new point, giving an extra polynomial in the spline. Finally, a path is achieved which in theory should be possible to follow by the vehicle. An example of what a generated path could look like is seen in Fig. 4.

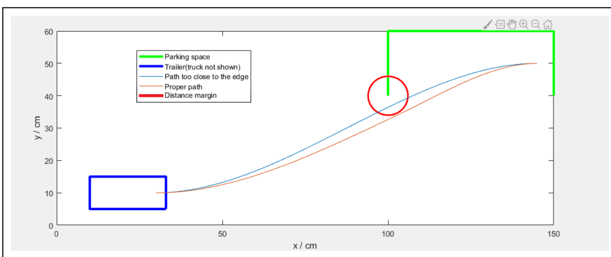


Figure 4. A simulated path

3.4 Pure pursuit controller

When the path is generated, an algorithm is needed in order to investigate how the trailer is supposed to move to be able to follow said path. The Pure Pursuit Controller was chosen as the algorithm for this task as it is a simple algorithm that yields good results.

Given a path to follow, a current point for the trailer, a chosen target distance T_d and the global angle of the trailer, θ_2, the Pure Pursuit Controller calculates a point on the path at a distance T_d from the trailer. The angle θ_e between the trailer and the target point, called the angle error, is then used to get a desired angle, β_{ref} between the truck and the trailer. To follow the path it is simply a matter of maintaining the β_{ref}, which is a job for the PI-controller. As the trailer moves closer to the point the Pure Pursuit Controller will eventually calculate a new point on the path and a new β_{ref} is calculated, giving a smooth following of the path, given that the path has enough points. The distance T_d to the target point that the Pure Pursuit Controller calculates is held constant and can be chosen to obtain a desired result. A larger distance gives a smoother ride but a larger error, while a shorter distance gives larger angles to turn to but a smaller error. This distance has not yet been chosen at the time of writing this report as the distance is supposed to be chosen through testing.

The formula for β_{ref} is seen in equation (16) and (17).

$$\beta_{ref} = arctan(\frac{2L_2 sin(\theta_e)}{R}) \tag{16}$$

$$\theta_e = arctan_2(Y_p - y_2, X_p - x_2) - \theta_2 \tag{17}$$

where (X_p, Y_p) is the target point coordinates. The full derivation for equation (16) and (17) can be seen in section 5 in [3]. The sign difference of β_{ref} is due to a different sign convention.

3.5 Resulting control system

The resulting control system is a fusion of both the PI controller and the Pure Pursuit Controller. The Pure Pursuit and the PI controller are serial connected, where the Pure Pursuit controller sends its output signal, β_{ref}, as a reference signal into the PI Controller, which will yield the angle α, the desired angle of the front wheels to obtain the angle β_{ref} of the trailer. A graphical representation of the control loop can be seen in Fig. 5.

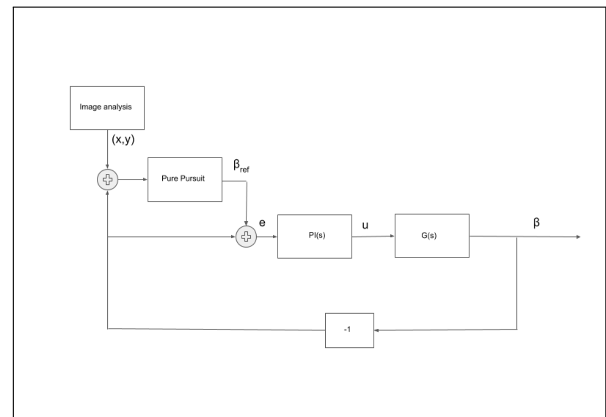


Figure 5. Control scheme

3.6 Implementation

Throughout the implementation of the control process in the chosen environment, all units in meters are converted into pixels, to facilitate the implementation with respect to the frame taken by the camera. This means that the lengths of the vehicle, L_1 and L_2 are represented in pixels, as well as the coordinates of the parking space, and the points generated in the spline.

Control system implementation To be able to implement the control system correctly in real-time, the PI controller is discretized. The Pure Pursuit controller only feeds information to the PI controller, and is already discrete, thus it is simply implemented in the program as an individual class. The PI controller, which ultimately controls the vehicle, can be described as

$$u(kh) = P(kh) + I(kh)$$

where h denotes the sample time and k denotes the sample. To discretize the controller, each part is discretized separately [8].

$$P(k) = K(\beta y_{sp} - y(k))$$

$$I(k+1) = I(k) - \frac{Kh}{T_i} e(k)$$

where βy_{sp} denotes a set point weighting, which is used to reduce the overshoot in the output, following step changes in the set point [9]. The I-part of the controller is discretized using forward difference, where K is the proportional gain, T_i the integral term and $e(k)$ is the error. In practice, there is a limitation on the magnitude of the control signal that can be realized by the actuator. The control can deteriorate drastically unless the controller detects when the control signal is saturated. The problem is that, when the control signal saturates, the integral part of the controller can continue to grow. This is called integrator windup, which can cause large overshoots [10]. To avoid this, constraints must be implemented for the control signal. One solution to this, called tracking, is when the control signal saturates, the integral is recomputed so that its new value gives a control signal at the saturation limit. The scheme in Fig.6 represents the implementation of tracking for a PID controller.

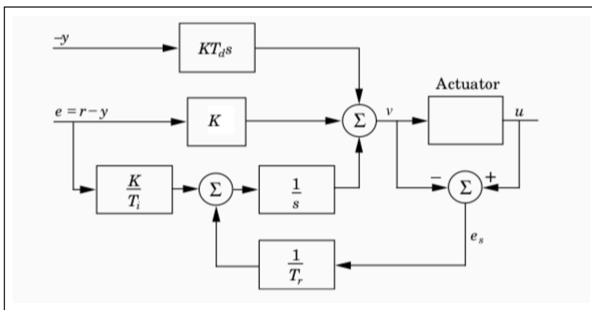


Figure 6. Tracking scheme of a PID controller in continuous time

Image processing To simplify the parallel parking process, image processing with color tracking is utilized. The camera is mounted above the vehicle in a fixed position, to overview the parking sequence. The colored ball fixed in the rear part of the trailer, is used to be able to trace the position of the rear axle of the trailer. The webcam continuously feeds the video into the computer, where color tracking is performed in each frame. The position of the ball is then used by the Pure Pursuit controller in order to find the desired angle between the trailer and the truck. Since the group members did not have any prior knowledge in image processing, the open source library *openCV* was used, together with a guide found online [7] that shows how to track a tennis ball. Slight alterations had to be made to this example in order to fit the application to the current project. The color tracking algorithm works as following pseudo code:

Following statements are executed periodically with a period of 33 milliseconds (which corresponds to 30FPS) {

- grabFramefromWebcam();
- void imageProcessing {
  - frame.convertRGBtoHSV();
  - HSVimage.blur(); //remove noise
  - for (all pixels in HSVimage): pixel.getHSVvalues();
  - boolean pixel.inCorrectHSVRange();
  - image.erode(); //exclude outliers
  - image.dilate(); //group same type of pixels
    }
- maskedImage.drawContours(); //Draw contours around grouped pixels
- contours.getArea(); //Calculate mass center of area within the countour.
- area.meanValue();
- return Point p; //mass center (x,y coordinates)

}

A picture of the Color Tracking GUI is shown in Fig. 7 below. The center of the ball is then found by retrieving the

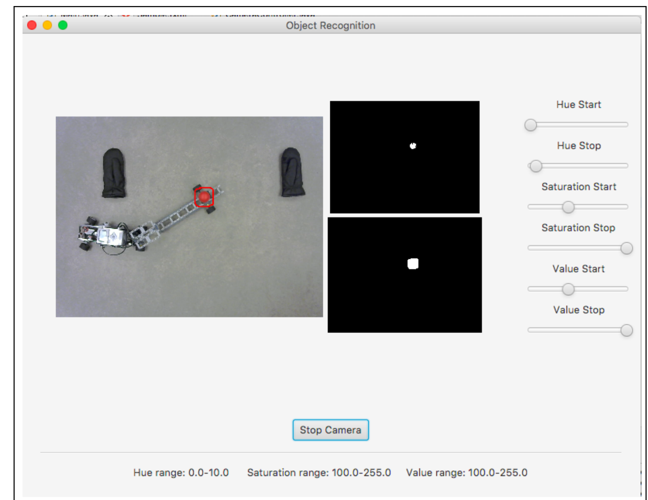


Figure 7. Color Tracking GUI

contours of the tracked object, and then the mass center of the area within the contours is calculated, using methods from the image processing library *openCV*.

Installation The programming language Java leJOS is used to implement the control process in the EV3 brick. The software is developed in Eclipse with a plugin for the leJOS firmware to allow for a smooth work flow and easy communication with the brick, since the code developed on the computer is also made in the Eclipse environment. All communication between the brick and the computer is done through

bluetooth network tethering, together with socket communication to enable sending and receiving messages between the brick and the computer.

The leJOS operating system used in this project has to be loaded onto an SD-card and inserted into the EV3 brick according to the instructions on their website [5]. The leJOS plugin to Eclipse has to be downloaded and installed from their website [6]. This plugin can be used to create leJOS projects and includes libraries for accessing the physical ports and buttons of the EV3 brick, as well as interfaces for working with the servo motors and sensor used in this project.

To start a program on the brick, the EV3 needs to be connected to the computer through a bluetooth network tether. After a bluetooth connection has been established, the EV3 can be connected as a PAN client through the network settings of the computer. Furthermore, the IP address of the computer and a commonly chosen port (in this case; '4444') are required to establish a socket connection between the two devices. The computer opens a server socket that the brick connects to upon starting the program. Input- and OutputStreams are used to send strings of data from the brick to the computer containing information about the process, and vice versa. The data sent from the brick contains the angle β given from the angle sensor, while the data sent from the computer contains the calculated angle α from the PI-controller. The information is represented as a one word string made up of digits.

Program structure The schematics of the classes and the communication implementation is visualized in Appendix A. Classes are represented by rectangles and their respective methods inside of these rectangles. The communication between them is shown by the drawn lines that connect the different rectangles. An arrow pointing at a rectangle means that that method is called from the class where the arrow origins from. In order to control the power to the motors directly, each motor has to be configured as an unregulated servo motor in leJOS. This allows control of power settings through a setPower method of the UnregulatedMotor class provided by leJOS. The program is built in such a way that both main classes (MainOSX and MainEV3) run continuously, in a sequential manner, sending and receiving data through the communication between the Client and Server classes. The CameraController class is run on a thread in order to keep the camera feed running continuously.

4. RESULTS

By bringing all components together in a simulation model and creating a theoretically realistic scaled parking lot - the control theory could be tested. By tuning the Pure pursuit parameters, PI-parameters and analyzing results from previous simulations, optimal parameters for the process were found. The Simulink schematics and corresponding Matlab code can be seen in Appendix B and C.

Simulation of the PI controller, yielded in following step response, see Fig. 8 where the overshoot and stationary error does not exceed 5%. Furthermore, Fig. 9 shows the simulation of β and β_{ref} angles of the resulting control system including both the Pure Pursuit controller and the PI controller, when this is simulated using the optimal simulation parameters. Fig. 10 shows the the parallel parking setup. The setup includes the initial path generated by the Cubic Hermite spline functions together with the simulated path taken by the trailer. The path taken by the vehicle is as a result of the Pure Pursuit controller together with the PI controller. The end position of the vehicle is also shown in the figure, represented by green and red crosses. The resulting controller parameters in the optimal simulated case are presented in Table 2 below.

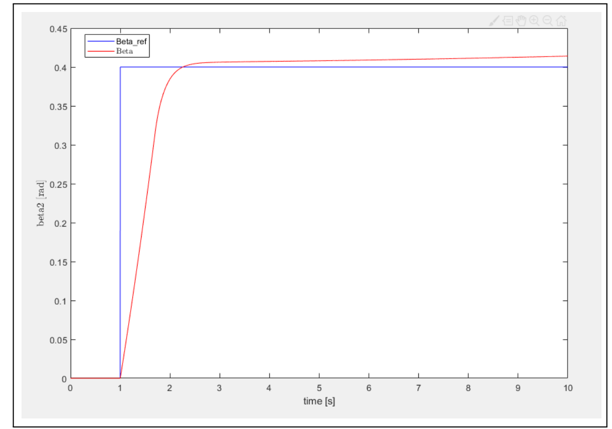


Figure 8. A simulated step response at time t=1

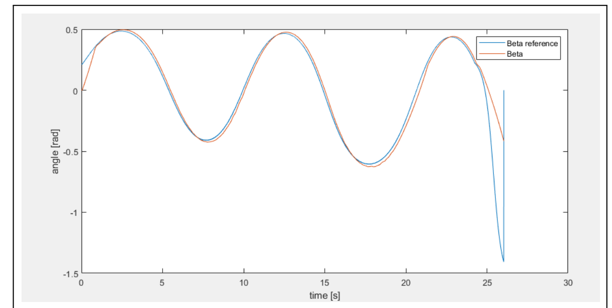


Figure 9. Beta reference and beta from a simulation with optimal simulation parameters

Table 2. Optimal simulation parameters

Parameter	Value
K	-13
T_i	1
T_s (sample time)	0.05 s
T_d (target distance)	0.2 m
L_1	0.26 m
L_2	0.356 m
v_1	0.1 m/s

The resulting controller parameters in the real application were found after some extensive tuning of the process in the implementation environment. These are represented in the table 3 below. The parameters L_1 and L_2 are the only parameters in this table that are not tuned, since these represent the constant lengths of the truck and trailer, respectively.

5. DISCUSSION

At the time of writing this report the project has not yet fully succeeded in parking the trailer. Because of this it is difficult

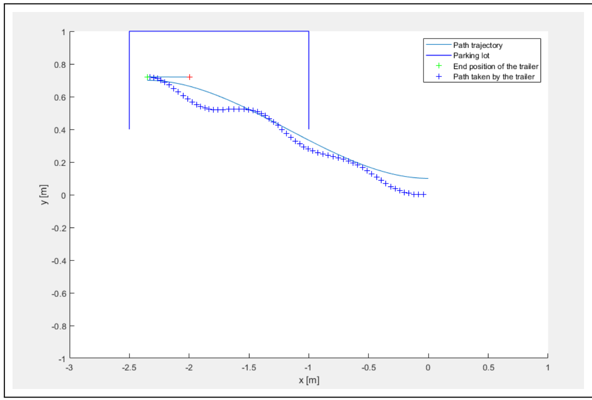


Figure 10. Parallel parking simulation with optimal control parameters

Table 3. Final parameters

Parameter	Value
K	-15
T_i	3
T_s (sample time)	0.1
T_r (tracking constant)	2
L_1	150 pixels
L_2	250 pixels
v_1	98 pixels/s

to draw proper conclusions about some of the components in the implementation. The group is confident that its designed control system is sufficient to succeed with the aim of parallel parking the vehicle. This is based on the performed simulations, where it is clearly seen that the control system behaves as desired. The step response from the PI controller shows that there is a minimum error, and that controller follows the step response smoothly, and is therefore a stable system. Furthermore, when the entire control system is simulated, it is seen that the β_{ref} follows the β angle closely without any major errors. It is also convincing that the simulation of the parking sequence looks as expected, where the vehicle follows the generated path and avoids any obstacles, in this case stays within the parking space boundaries.

At the current state in the project, the group believes that the issue is regarding the actual implementation of the process in the programming environment, and not in the control design. The vehicle is able to reach inside the parking space, but not as smoothly as desired. The vehicle finds its end of the path somewhere in the middle of the parking space, when the desired point of stopping would have been more to the end of the parking space. Also, the group has encountered an issue where the programming environment crashes after an uncertain amount of time.

5.1 Troubleshooting

In this section, the group has chosen to include some of the major problems that occurred during the project. The section also includes the time consumption when trying to solve the problems, which was a setback for the group's progress and planning.

Throughout the implementation, the group has systematically gone through every section of code in order to facilitate troubleshooting. Each section of code has been isolated and tested independently from the rest of the structure. This was done in order to find possible problems that affected the entire process.

The trajectory generation has been tested by simply moving the red ball by hand and checking the calculated coordinates given by the image analysis. When comparing the ball coordinates and the spline coordinates the numbers look proper. This is easily verified by moving the ball in roughly the same manner the spline would. Also, the generated spline was tested in Matlab, with the values given from the programming environment, and the resulting plot, Fig. 11 shows that the spline visually looks as expected.

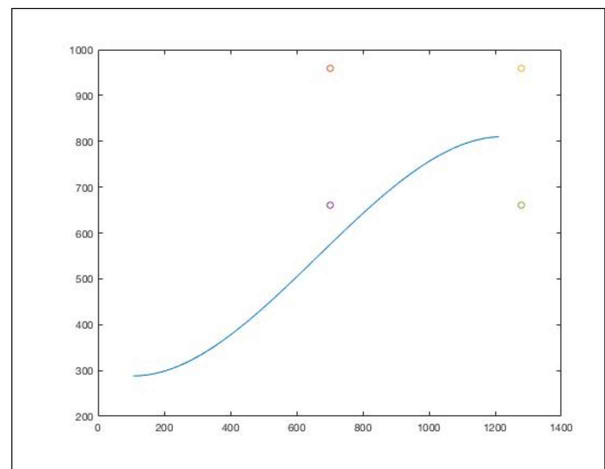


Figure 11. Spline generated from the programming environment

The PI-controller has been tested by setting constant β_{ref} values, essentially bypassing the Pure Pursuit controller and the image analysis. This has been working as expected, the vehicle is able to smoothly follow a constant reference signal for a longer time without generating any errors. Thus it can be concluded that the PI-controller works.

Given that the PI-controller can handle reference values it entails that the EV3 brick itself seems to perform as it should. With this it would be safe to assume that the fault lies in the Pure Pursuit controller. While it has undergone testing it is still not clear as to why it seems to not behave as intended.

However as is mentioned in the beginning of this chapter it is hard to draw any final conclusions about the components of the implementation. Something may still have been overlooked somewhere in the code and without a fully functioning end result nothing should be seen as fully functional.

OpenCV + leJOS At the beginning of the project, a lot of time was spent downloading the various software applications for Eclipse. This was rather difficult at first due to the group's lacking knowledge of the applications.

Communication The group have experienced some problems with the communication part of the project. Since the image analysis is implemented using OpenCV, all including classes must be ran through Java version 1.8. OpenCV is not

compatible with any other version. Meanwhile, all classes for controlling the robot are used with Java leJOS, which is only compatible with Java version 1.7. This caused a lot of trouble when first implementing the code. An enormous amount of time was spent on trying to figure out how to work around this problem. After re-arranging the code and classes, creating a better structure, the group came to the conclusion that a Client/Socket communication would be the best solution. An attempt to use this type of communication between all classes run on version 1.8 and 1.7 did not give any results. This was due to the fact that the communication was run from the computer to itself. The final solution then became to use the Client/Socket between the computer and the LEGO Mindstorms brick. The final program structure is shown in Appendix A.

Java crashes At the time of writing this report, the latest occurring problem was that the programming environment Eclipse was crashing when running the code. This was to the fact that the code includes a lot of different applications such as the image analysis and the leJOS plugin. After investigating the error message, the group believed that the main cause of the problem was the fact that the JavaFX plugin was running on an older version of Eclipse/Java. After testing this idea, the group could conclude that this was not the issue, which to this day remains unknown. Therefore, when Java crashed, the group simply executed the code again, hoping on a better result.

5.2 Potential improvements

After solving most of the major problems described above, the group was able to start tuning the parameters and narrow down the errors. Unfortunately, this happened rather late in the project, resulting in a very short time to debug small errors. If the opportunity to re-do the project was given, a few changes in the work flow would have been made. First of all, a better guide on how to implement all the required applications would have helped a lot. Also, more and deeper knowledge on how the communication should work would have made a huge difference.

A future goal of the project would have been to try to straighten up the truck and trailer after getting the trailer into the parking spot. The easiest way to achieve this would be to hard code minor commands to the LEGO Mindstorms brick. For an example, once the back of the trailer has made its way into the parking spot, the commands could be to, continuously, drive back and forth with a slight angle on the front wheels for a while. This would eventually result in the entire vehicle straightening up. In this project, however, the limitation was set to only having the trailer parallel to the parking spot at the end of the parking sequence.

The choice for what interpolation method to use was for the most part affected by the time limitation of the project. The idea was to start out with something that would work, and later perhaps look at other methods. Since [3] showed that Cubic Hermite interpolation was working, this was chosen as a starting point. Unfortunately, there was not enough time to try out other methods in the end.

Overall, the group are satisfied with the project, having learnt a lot from the different obstacles encountered over time. Also, having made such progress considering the limited amount of time that was left when the program finally started working, is something that the group are proud of. It would have been interesting to continue working on the project, for further improvements and a "flawless" parallel parking.

Bibliography

[1] Mathematical Model of LEGO EV3 Motor, `http://nxt-unroller.blogspot.com/2015/03/mathematical-model-of-LEGO-ev3-motor.html`

[2] Logitech HD webcam c270 Technical Specification, `https://support.logitech.com/en_us/article/17556`

[3] A. Ganslandt, A. Svensson and J. Ericson *LEGO Trailer report in Projects in Automatic Control*. Department of Automatic Control, Faculty of Engineering, Lund University, 2017.

[4] O. Ljungqvist, *Motion Planning and Stabilization for a Reversing Truck and Trailer System*. Department of Electrical Engineering, LinkÃűping University, 2015. `http://liu.diva-portal.org/smash/get/diva2:826978/FULLTEXT02.pdf`

[5] Windows Installation, `https://sourceforge.net/p/leJOS/wiki/Windows%20Installation/`

[6] Installing the Eclipse plugin, `https://sourceforge.net/p/leJOS/wiki/Installing%20the%20Eclipse%20plugin/`

[7] Object Detection, `https://opencv-java-tutorials.readthedocs.io/en/latest/08-object-detection.html`

[8] Approximation of Analog Controllers, PID Control, `http://archive.control.lth.se/media/Education/EngineeringProgram/FRTN01/2018/L08_slides6.pdf`

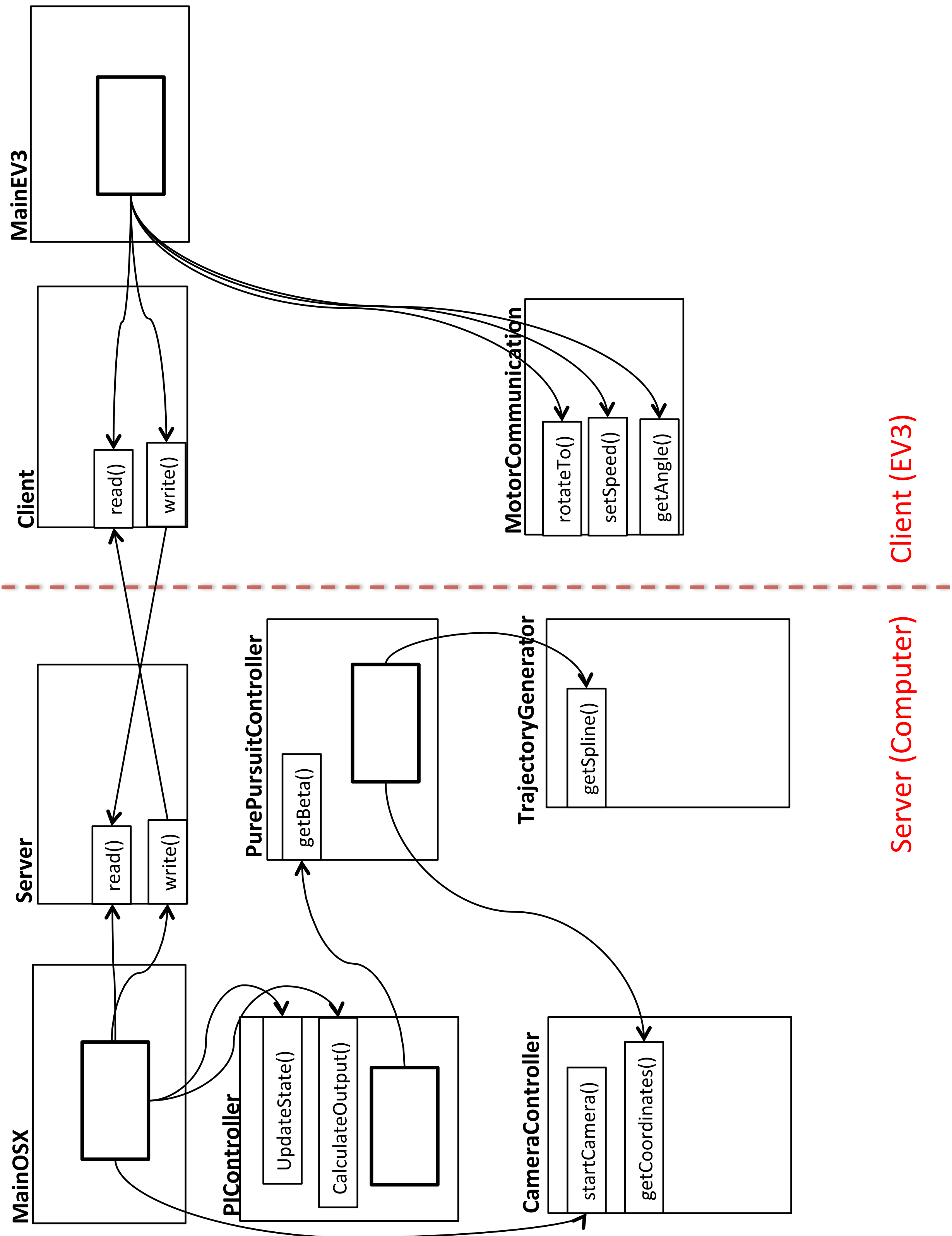
[9] Set-point weighting, `https://people.eng.unimelb.edu.au/mcgood/ctrl301/pid_old/pid-wsp.html`

[10] REPETITION (OCH LITE NYTT) AV REGLERTEKNIKEN, `http://www.it.uu.se/edu/course/homepage/h2orentek/WWT98/RepReg.pdf`

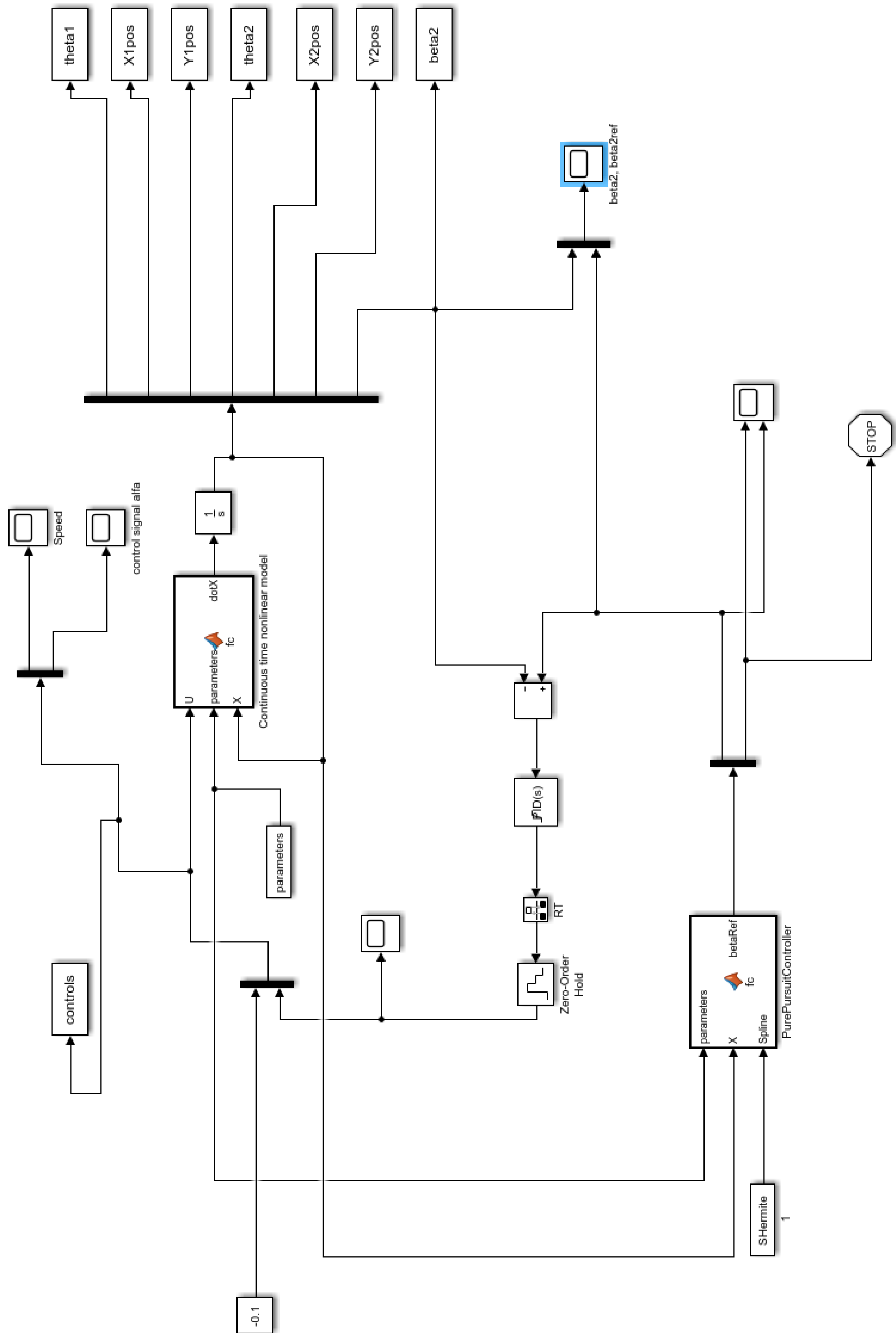
APPENDIX

Appendix A: Program Schematics
Appendix B: Simulink Diagram
Appendix C: Matlab Code

Appendix A



Appendix B



Appendix C

Trajectory generation (1/2)

```
%Trajectory generator for the trailer with plotting
%Currently has arbitrary numbers until we can get data from the image
%analysis
%distanceMargin minimal distance to corner i.e length from center to tire on
%trailer
distanceMargin = 6;

%Input:
%Parking lot corners
xpark1=100;
ypark1=40;
xpark2=250;
ypark2=100;

%trailer coordinates at t = 0
xtrailer0 =0;
ytrailer0 = 10;

%Plot parkinglot

 %[150, 150, 100, 100]
 %[40, 60, 60, 40]

xpark = [xpark2, xpark2, xpark1, xpark1];
ypark = [ypark1, ypark2, ypark2, ypark1];
xedge = xpark1; %Edges to be caseful with
yedge = ypark1;

plot(xpark, ypark, 'b-', 'LineWidth', 3);
% hold on;
% xlim([0, xpark2*1.2]);
% ylim([0, ypark2*1.7]);
% viscircles([xpark1, ypark1],distanceMargin); %Plot "danger zone"

%plot trailer
xtrailer1= xtrailer0-26;
xtrailer2= xtrailer0;
ytrailer1= ytrailer0;
ytrailer2= ytrailer0;
xtrailer = [xtrailer1, xtrailer2, xtrailer2, xtrailer1, xtrailer1];
ytrailer = [ytrailer1, ytrailer1, ytrailer2, ytrailer2, ytrailer1];
plot(xtrailer, ytrailer, 'b-', 'LineWidth', 3);
% hold on;

xend = xpark2 - 0.1*(xpark2 - xpark1);
yend = 0.5 * (ypark1 + ypark2);
xstart = xtrailer0;
ystart = ytrailer0;
xHermite = xstart:0.1:xend;%Adds evenly distributed x-values between the start
and end points with increments of 0.1
maxIndex = numel(xHermite);
sHermite = [];
for xx = 1:maxIndex
    t = (xHermite(xx) - xstart)/(xend - xstart);
    h0p = 2*t^3 - 3*t^2 + 1;
```

Trajectory generation (2/2)

```
        h1p = t^3 - 2*t^2 + t;
        h2p = -2*t^3 + 3*t^2;
        h3p = t^3 - t^2;
        sHermite = [sHermite, h0p * ystart + h2p * yend];
    end

    plot(xHermite,sHermite);
    % hold on;

    %Find closest point to corner of parking space
    SHermite = [xHermite;sHermite];

    % Compute the distance of each of those points from (xedge, yedge)
    distances = sqrt((SHermite(1 , :) - xedge) .^ 2 + (SHermite(2, :) - yedge) .^
    2);
    % Find the closest one.
    [minDistance, indexOfMin] = min(distances);

    %If closest point is too close we interpolate on an additional point
    %further from that corner
    if (minDistance < distanceMargin)
        k = (SHermite(2,indexOfMin) - yedge)/(SHermite(1,indexOfMin) - xedge);
        newPoint = [1.1*distanceMargin/sqrt(1 + k^2) + xedge,
    1.1*distanceMargin*k/sqrt(1 + k^2) + yedge];
        sHermite = [];
        midTangent = -1/k;
        for xx = 1:maxIndex
            if (xHermite(xx) < xHermite(indexOfMin))
                x0p = xHermite(1);
                y0p = ystart;
                x1p = newPoint(1);
                y1p = newPoint(2);
                deltax = newPoint(1) - xHermite(1);
                m0 = 0;
                m1 = midTangent;
            else
                x0p = newPoint(1);
                y0p = newPoint(2);
                x1p = xHermite(maxIndex);
                y1p = yend;
                deltax = xHermite(maxIndex) - newPoint(1);
                m0 = midTangent;
                m1 = 0;
            end
            t = (xHermite(xx) - x0p)/(x1p - x0p);

            h0p = 2*t^3 - 3*t^2 + 1;
            h1p = t^3 - 2*t^2 + t;
            h2p = -2*t^3 + 3*t^2;
            h3p = t^3 - t^2;
            sHermite = [sHermite, h0p*y0p + h1p*deltax*m0 + h2p*y1p +
    h3p*deltax*m1];
        end
    end

    SHermite = [xHermite/100;sHermite/100];
```

Initiation code

```
%% Parameters

L1 = 0.26; %Length of truck
L2 = 0.356; %Length of trailor
M1 = 0    ; %Hitch length

k = (-2.2-0.1/L2)*L1/0.1
Ti = -0.1*k/(L1*1.3)
parameters = [L1,L2, M1];

%% Initial conditions
h    = 0.01;  % Time step
theta1_0 = 0;
x1_0 = L2;
y1_0 = 0;
theta2_0 = 0;
x2_0 = 0;
y2_0 = 0;
beta2_0 = 0;

%% Run simulation
open('model2.slx')
sim('model2.slx')

%% Plot results
stateResponse = {X1pos,Y1pos,theta1,theta2,X2pos,Y2pos,beta2};
colors        = {'r-','b--','k-','g--'};
ylabels       = {'$x1(t)$', '$y1(t)$',
'$theta_1$','$theta_2(t)$','$x2(t)$','$y2(t)$','$beta2$'};

subplot(4,1,1); hold on;
plot(controls.Time, controls.Data(:,1), 'r', 'LineWidth',2)
plot(controls.Time, controls.Data(:,2), 'b', 'LineWidth',2)
legend({'Velocity, $v(t)$', 'Steering angle, $\delta_f(t)$'},
'Interpreter', 'latex', 'Location','NorthWest')

for ii = 1:7

    subplot(8,1,1+ii);
    hold on;
    states = stateResponse{ii};
    plot(states.Time, states.Data(:,1), colors{1}, 'LineWidth',2)
    legend({'Continuous time'}, 'Interpreter', 'latex',
'Location','NorthWest')
    ylabel(ylabels{ii}, 'Interpreter', 'latex')


end
x1 = X1pos.Data(:,1);
y1 = Y1pos.Data(:,1);
x2 = X2pos.Data(:,1);
y2 = Y2pos.Data(:,1);
betu = beta2.Data(:,1);
reff_ = beta_ref.Data(:,1);

%reff_tid = reff.Time(:,1);
betu_tid = beta2.Time(:,1);
```

Continuous model function block

```
function dotX = fc(U, parameters, X)
% Continuous time nonlinear model
v  = U(1);
alfa  = U(2);


theta1 =  X(1);
theta2 = X(4);
beta2 = X(7);

% Parameter extraction
L1_ = parameters(1);
L2_ = parameters(2);
M1_ = parameters(3);

dotTheta1 = v/L1_*tan(alfa);
dotX1 = v*cos(theta1);
dotY1 = v*sin(theta1);
dotTheta2 = v*sin(beta2)/L2_ - M1_*cos(theta1-theta2)*dotTheta1/L2_;
 v2 = v*cos(beta2)+M1_*sin(theta1-theta2)*dotTheta1;
dotX2 = v2*cos(theta2);
dotY2 = v2*sin(theta2);
dotBeta2 = v/L1_*tan(alfa) - v*(sin(beta2)/L2_-
(M1_*cos(beta2)*tan(alfa)/(L1_*L2_)));


 dotX = [dotTheta1; dotX1; dotY1; dotTheta2; dotX2; dotY2; dotBeta2];
```

Purepursuit function block

```
    function betaRef = fc(parameters, X, Spline)
% PureSuitController

targetDistance_ = 0.2  ;%Distance to target

RearPointX = -X(5); %everything else designed for growing X value
RearPointY = X(6)+0.1;
L2_ = parameters(2);
S_ = Spline;
rearPoint = [RearPointX, RearPointY];
S_ = S_(:,S_(1,:) >= rearPoint(1)); %Remove "old" coordinates
A = 0; %variabel to end simulation if A = 1

distance = 99; %this value will never be used - only to make simulink happy
BetaRef_ = 0;  %this value will never be used

if ( numel(S_(1,:)) == 0 )

    A = 1;
    disp('nume = 0')
else
    for indexTarget_ = 1:numel(S_(1,:))
            distance = sqrt((S_(1,indexTarget_) - rearPoint(1)).^2 +
(S_(2,indexTarget_) - rearPoint(2)).^2);
           % disp('forloop after distance')
         if (distance > targetDistance_)
             %disp('at break')
             break;
         end
    end

    if (indexTarget_ == 1)
    %We have reached our destination (indexTarget = 1), return or smth.
    disp('work done')
    A = 1;
    else

    deltax = S_(1,indexTarget_) - rearPoint(1);
    deltay = S_(2,indexTarget_) - rearPoint(2);

    angleError = atan2(deltay,deltax);

    BetaRef_ = atan(2*L2_*sin(angleError)/distance);
    end
end

betaRef = [BetaRef_, A];
```

Code for analysis

```
figure(2)
d = zeros(1000,1);
axis([-3 1 -1 1])
hold on
plot(- SHermite(1,2:2351),SHermite(2,2:2351));


xpark1=-1;
ypark1=0.4;
xpark2=-2.50;
ypark2=1;

xpark = [xpark2, xpark2, xpark1, xpark1];
ypark = [ypark1, ypark2, ypark2, ypark1];

plot(xpark, ypark, 'b-', 'LineWidth', 1);

plot(x2(numel(x2(:,1)),1), y2(numel(x2(:,1)),1),'g+');

for t=1:numel(x2(:,1))
   crank = line([x1(t,1) x2(t,1)], [y1(t,1) y2(t,1)]);
   hold on
   h1 = plot(x1(t,1), y1(t,1),'r+');

   h2 = plot(x2(t,1), y2(t,1),'g+');

   %pause(0.0001)

  d(t,1) = sqrt((x1(t,1)-x2(t,1))^2+(y1(t,1)-y2(t,1))^2);
if (rem(t,90) == 0)
    plot(x2(t,1), y2(t,1),'b+')
end
   if (t<numel(x2(:,1)))
        delete(crank)
        delete(h1)
        delete(h2)
   end
end

legend('Path trajectory','Parking lot','End position of the trailer','Path
taken by the trailer');
xlabel('x [m]')
ylabel('y [m]')
```

Ball Catching Robot

Daniel Johansson[1] **Dennis Dalenius**[2] **Nicklas Norborg Persson**[3] **Per Josefsson**[4]

[1]elt14djo@student.lu.se [2]elt14dda@student.lu.se [3]elt14npe@student.lu.se
[4]elt14pjo@student.lu.se

Abstract: The objective of the project has been to get a robotic arm to catch a ball being thrown to it. The main tasks have been to perform image analysis, calculate a ball trajectory, as well as to move a robot to the correct position. Different methods have been tested with varying results. The end result was satisfactory, the robot used was able to catch balls thrown within its reach and without dropping them. The project was however limited to 2D and is only able to catch balls in a plane. This was due to limited time as well as 3D increasing the difficulty substantially.

1. Introduction

The main goal of this project has been to get a robot arm to catch a ball that was thrown to it. This was done by recording the throw and through image analysis detecting the ball by color and movement. When the ball was detected, the computer would calculate a trajectory and thus determine where the ball would land. These coordinated were then used to calculate what angles the robotic arm needed to position itself in order to reach the coordinates and catch the ball.

This has been done modularly. Each large part of the project has been solved to work individually before being combined into one unit. Thereafter the project consisted largely of optimizing and tuning each part in order to allow for cooperation.

This project was done last year and that report was read as research[1]. There has also been a similar project done by Magnus Linderoth [1] which has been an inspiration to the project.

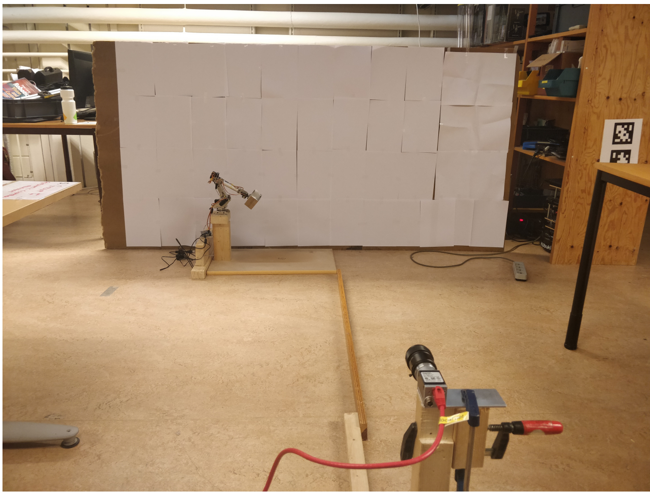


Figure 1. The full system setup. The camera is aimed at the robotic arm and the white background screen.

[1]http://portal.research.lu.se/portal/files/38414307/projektrapport.pdf

2. Modeling

2.1 Trajectory Calculations

The goal of the trajectory calculation was to predict a trajectory for the ball and to calculate the position where the robot should catch the ball. Due to some problems with the sampling, two methods for trajectory calculations were developed. The first one, using a model of the ball which did not work properly due to an issue with sampling. The second solution utilized polynomial regression.

Model of ball With the position vector $p = [X, Y]^T$ the differential equation (1) describes the ball in motion

$$\ddot{p} = -c\dot{p}||\dot{p}||_2 - \begin{bmatrix} 0 \\ g \end{bmatrix} + v_c \tag{1}$$

where c is a air drag constant, g is the gravitational constant and v_c is a load disturbance i.e disturbance caused by wind. By using the state vector $x = [X, Y, \dot{X}, \dot{Y}]^T$ the differential equation can be written in state space form seen in (2)

$$\dot{x} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ -c\dot{X}V \\ -c\dot{Y}V - g \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ v_c \end{bmatrix} \tag{2}$$

where $V = \sqrt{\dot{X}^2 + \dot{Y}^2}$ is the speed of the ball. The discretized system is given by (3)

$$x(k+1) = \begin{bmatrix} 1 & 0 & h & 0 \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} \frac{h^2}{2} \begin{bmatrix} -c\dot{X}(k)V(k) \\ -c\dot{Y}(k)V(k) - g \end{bmatrix} \\ h \begin{bmatrix} -c\dot{X}(k)V(k) \\ -c\dot{Y}(k)V(k) - g \end{bmatrix} \end{bmatrix} + v(k) \tag{3}$$

For simplicity the air drag was neglected ($c = 0$) giving a linear system (4).

$$x(k+1) = \begin{bmatrix} 1 & 0 & h & 0 \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ -gh^2/2 \\ 0 \\ -gh \end{bmatrix} + v(k)$$
$$= \Phi x(k) + \Gamma + v(k) \quad (4)$$

Since the air drag was neglected the decision of what ball to use was very important, as a light ball relative to its size would be heavily affected by drag.

With a given initial state $x(0)$ one can iterate the system in Equation 4 to get a prediction of the position at future time points. This model was used together with a Kalman filter described in Section 4.3.

Polynomial regression This method used polynomial regression to fit a second order polynomial to the gathered measurements. This polynomial would then be the estimated trajectory of the ball. The polynomial coefficients were calculated and a curve could then be fit to the trajectory.

Catching point Assuming a known trajectory of the ball and an interception boundary from which the interception point could be calculated. For this project two boundaries were tested.

The first one was a vertical limit. Whenever the trajectory intersects the boundary the "catch point" was determined. This can be seen at Figure 2 where the vertical line is the boundary.

The second one was a boundary consisting of the upper part of a ellipse and a vertical line, the red line in Figure 2.

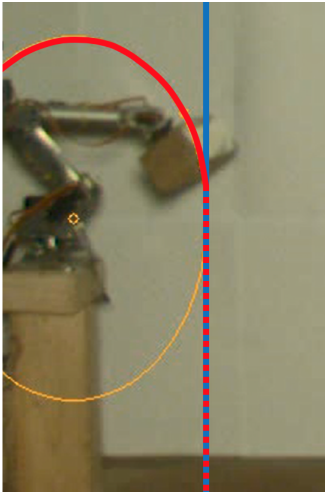


Figure 2.

2.2 Robot arm model

The model of the robot arm was used to describe the angle between the robot segments, e.g. the joint angles of the robot and the position of the servo motors used to control the arm. For the innermost and outermost servo motors, this was done by defining the angle of the servo motor to be equal to the angle of the joint it was controlling, as the servos had direct control of the joint

For the servo controlling the middle joint however, the problem was not as trivial, as the angle of the middle joint is determined by two servos, the servo used to control the joint angle and the servo used to control the innermost joint. Figure 3 shows the model of the middle joint and the components that were used in determining the servo angle. From the model, and with some slight simplifications, the relationship between the servo angle θ_2 and the joint angles θ_{j1}, corresponding to the innermost joint angle, and θ_{j2}, corresponding to the outermost joint angle, was calculated as

$$\theta_2 = cos^{-1}((l_1 - l_2 - l_3 sin(90 - \theta_{j2}))/l_4) - \theta_{j1}. \quad (5)$$

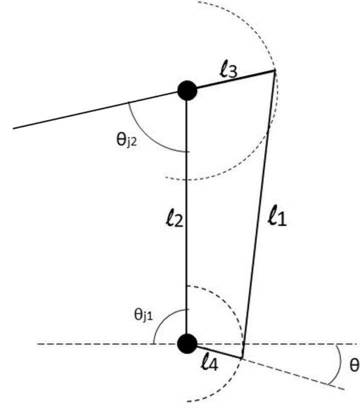


Figure 3. Model used to calculate the desired angle of the servo controlling the middle joint.

2.3 Robot arm movement

The first solution attempted was using an algorithm called FABRIK and the second one was modeling the arm segments as two circles and then finding the point where these intersect. However, initially the cup used to catch the ball needs to be aligned by position the end segment of the arm.

Aligning the cup In order to be able to catch the ball the alignment of the cup is important. If it is not correctly aligned with the incidence angle that the ball is approaching the robot with, then the ball will hit the edge of the cup and thus not land in it. A picture of this can be seen in Figure 4

This was solved in a way such that the incidence angle of the ball was a parameter for the algorithm. The last segment then used this angle in order to align the cup with the trajectory. This allowed for the algorithms explained in the following segments to only determine the position of two segments instead of three, which reduced the amount of calculations and thus also calculation time.

Kinematics Two operations were needed in order to perform FABRIK. These were inverse kinematics as well as forward kinematics. Inverse kinematics was central for calculating the movements of the robot arm as it was the way of finding the angles between each segment in order to reach a certain coordinate [2]. By knowing the measurements of the robot arm inverse kinematics would make this possible.

However, inverse kinematics is not enough. There was one additional method that was used in order to find the proper

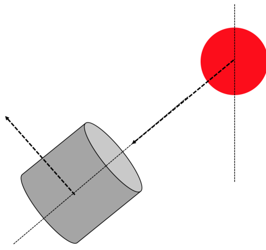


Figure 4. Showing how the cup was aligned with the trajectory.

movement of the arm, forward kinematics. This is a much simpler method. It calculates where the end-effectors are while knowing the measurements for the arm, as well as the angles between the between them[2].

FABRIK As stated in previous sections, forward and inverse kinematics are of great help when determining the movements of the arm in order to reach the ball. However, simply using one of these is not enough. The solution is using an algorithm called FABRIK (Forward And Backwards Reaching Inverse Kinematics). This method uses both inverse and forward kinematics in order to find the position of the end-effector as well as the angles between each segment that allows for this position.

The way that this algorithm works is that it does one inverse followed by one forward calculation and then evaluates the result. If the result is satisfactory, then the arm will move to the calculated angles. Otherwise the calculation will be done once more until the result is good enough. What this means is that FABRIK is a heuristic method, meaning that gives the guess. However good it may be, the result will hardly ever be perfect. Although, the achieved result will improve with every iteration, it is up to the developer to determine when it is good enough to use.

The way that each iteration of FABRIK is executed is by first placing the end-effector of the last segment at the target point. After this the remainngi segments are moved to fit this constellation of the robot arm. Following this the base segment is once again moved to the base and the last segment is corrected.

Circle intersection There is another method used to model the robot arm movements that proved to be more efficient than FABRIK. This was done by modeling the robot arm as two circles with the origin at the two points where the arm should end up. The first circle has its origin at the base of the robot arm and has a radius that is the length of the first segment of the robot. The second circle has its origin at the base of the third, already correctly positioned segment, and the radius of the circle is the length of the second segment of the arm. A picture of what this looks like can be seen in figure 5

As can be seen in Figure 5 there are two points of intersection for the two circles. These are the two possible solutions in order to position the arm and reach the ball. These are then evaluated based on which is possible to reach considering the limitations of the servos.

This solution also solves for when the ball is not within reach of the robot. This is the case where there are no intersections of the circles, as can be seen in figure 6.

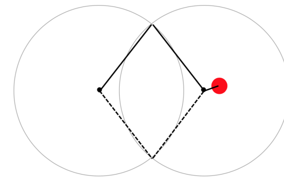


Figure 5. Second method of finding the position of the robot arm.

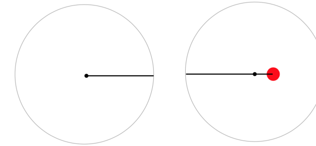


Figure 6. Second method of finding the position of the robot arm when the ball is out of reach for the robot.

GUI To be able to view the results from moving the robot a GUI that allowed showing the change of the robot arm position as well as where that ball was located was developed. Pictures from the GUI can be seen in Figure 7.

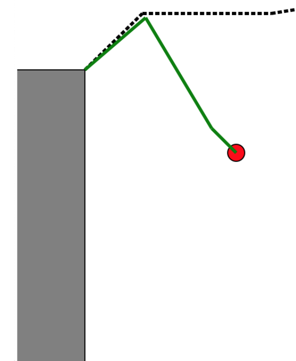


Figure 7. Graphics of the robot. The red circle symbolizes the ball and the grey box is the base on which the arm is placed. The dashed line is the arm before movement and the green line is the arm after movement

Figure 7 shows the robot arm before and after a solution has been found.

3. Electro-Mechanics

3.1 Robotic arm

The robot arm used in the project was already built when the project started. Figure 8 shows a picture of the robotic arm used in the setup. The arm is controlled by six servos, although only three were used in this project. Which three were used can be seen in Figure 9. The choice to only control three servos was made as a result of the problem being solved in two dimensions. All of the joints that were not used would only have allowed for control in three dimensions and were thus not used. To be able to power and control the servo motors an Arduino and top mounted servo shield was used. The servos take a pulse width modulated (PWM) signal as input, and the

duty cycle of the PWM signal determines the servo position, or angle. The servos also require a supply voltage of 4.8-6 V and the voltage determines the speed of the servos as well as their strength[2] [3]. For this reason, a voltage of 6 V was used as supply. The two innermost servos are of model MG996R, described as a high torque servo, and the outermost servo, which did not do any heavy lifting is of model MG90S. The speed of the servos at the chosen supply voltage is about 0.14 s/60°for the heavier servo and 0.08 s/60°for the lighter.

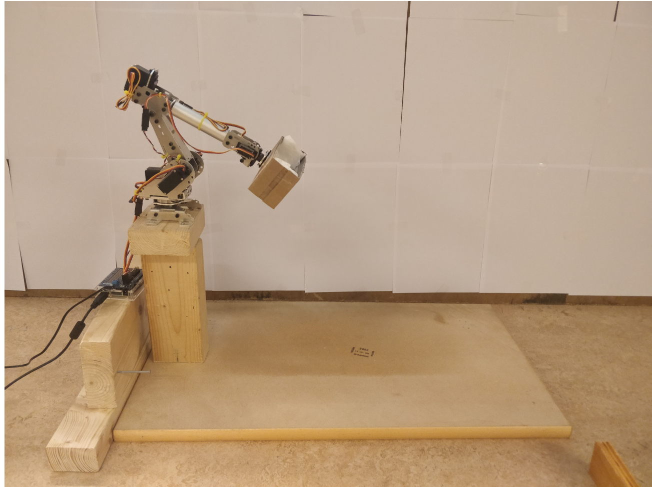


Figure 8. The robotic arm that is used to catch the ball. To the left of the ball it is possible to see the Arduino that controls it.

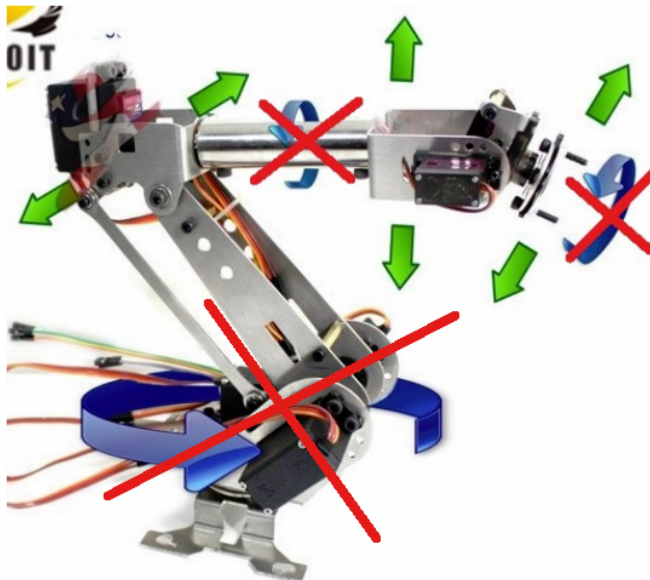


Figure 9. The ways the robot arm can move. Movements not used are crossed out. *Courtesy of group A, FRTN40 2017* [1].

3.2 Camera Setup

The camera used for the final setup was a *Basler ACA800-200GC*. This is a high speed camera that can reach 200 Frames Per Second, which is a step up from the initially used webcam that could reach around 30 FPS. The camera is connected to the main computer with an ethernet cable connected through a Gigabit switch. An ethernet-to-USBC adapter was also used to connect it to the main computer (a laptop) as it did not have any ethernet port. The camera and how it is mounted is shown in figure 10.

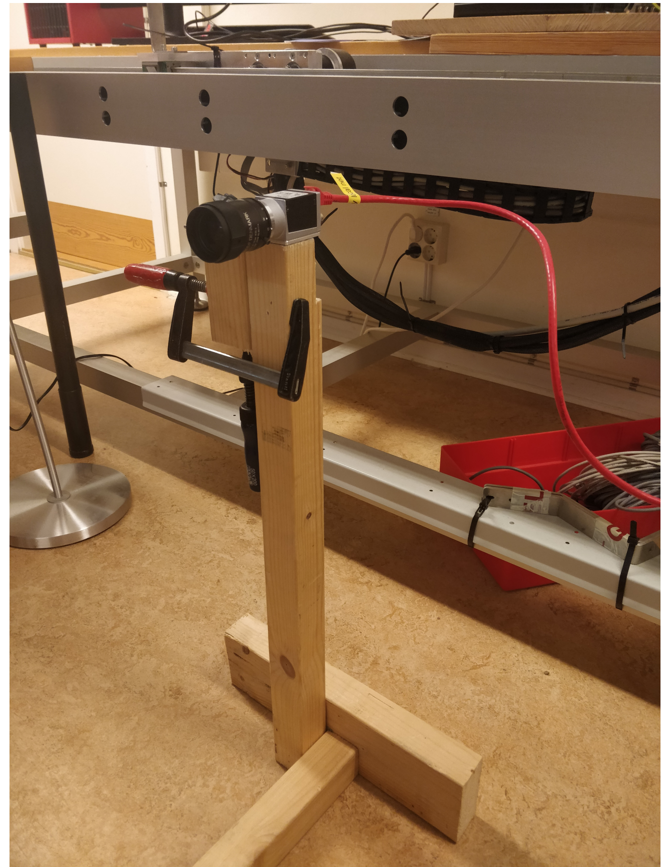


Figure 10. The high speed camera and its mounting that was used to capture the ball location.

4. Control

4.1 Real-time implementation

The control algorithm for the system was implemented on the main computer in the programming language Python 3.6. The reasoning behind the choice of this programming language comes from its ease of use, many and well documented libraries, as well as its good interface with the library *OpenCV* that was used for image analysis. It was implemented with respect to the necessary real-time programming aspects and was run on a single thread. There are four main parts of the control algorithm: the image analysis, the trajectory calculation, the robot kinematics calculations, and the transmission of the actuator signals to the robot. As these are quite separate tasks, they could have been implemented in four different threads, but due to the high speed requirements of the algorithm, the python threading libraries was to slow for this. The sample time was set to 20 Hz (sample every 50 ms) and the algorithm took around 25-35 ms; the algorithm was sufficiently fast. But as Python does not allow real multi-threading, the packages

[2] https://engineering.tamu.edu/media/4247823/ds-servo-mg90s.pdf

[3] https://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf

implementing these features do it virtually, adding a lot of overhead that draws a lot of computational power. This makes the control algorithm much slower and it will not finish within the required 50 ms. It would of course be possible to make the sample time slower, but 20 Hz has been tested to be sufficiently fast for the trajectory calculation and to make the robotic arm catch the ball. Furthermore, it is logical to have a sequential algorithm as the tasks should be executed in a specific order.

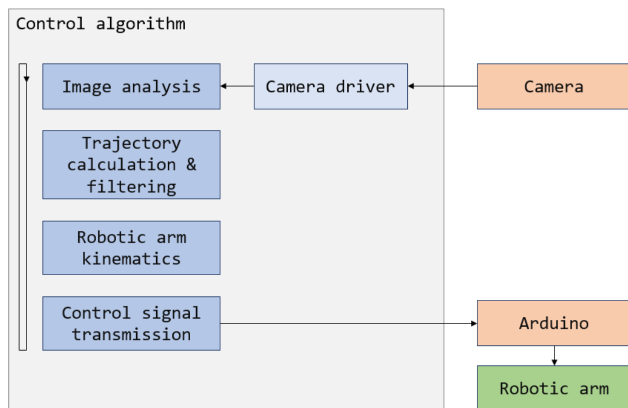


Figure 11. Control algorithm for the system. The different tasks are done sequentially every iteration. The input to the algorithm is an image from the camera, and the output is the servo angles for the robot sent to the Arduino through a serial connection.

Figure 11 shows a block diagram for the control algorithm. The control algorithm has access to the camera feed that is provided by the high speed camera and from this the latest frame is sampled at every iteration. The sampling time for the control algorithm has been set to 20 Hz so the algorithm reads an image every 50 ms. To be able to sample this camera stream, a camera driver module provides functions to make it easy to connect to, and get images from the camera using the *Pylon* interface provided by the camera manufacturer.

The image is then processed in the Image analysis module. This module searches for the ball characteristics in the image as it looks for the specific ball color and moving objects. This is described in further detail in Section 4.2. When the image analysis has been completed the ball coordinates have been determined and are passed to the next module, the trajectory calculation. The trajectory calculation then uses the measured values and the model of the ball trajectory to calculate the optimal point, where the robotic arm will catch the ball.

When the catching point coordinates have been determined, the joint angles has to be calculated for the robotic arm to move to the desired position. This is done by passing the catching point coordinate to the module for the robotic arms kinematics. The joint angle calculations are described in Section 4.4. When the angles for the robotic arm's servo motors has been calculated, they are transmitted over a serial link to the Arduino which applies the correct servo signals to move the robotic arm to the correct position. The Arduino runs a separate program that listens for commands sent over the serial link, and when a command has been received it is directly executed. The robotic arm's servo motors are powered through a servo shield mounted on top of the Arduino. To be able to use this shield a driver library from *Adafruit* was used[4].

4.2 Color and motion based object detection

In order to detect the ball, a high speed camera has been set up to deliver a video stream to the main computer. The main program is running on the main computer and analyses the video stream frame by frame to get the exact location of the ball. Currently there are two methods being used to detect the ball: *Color detection* and *Motion detection*. Both methods have different strengths and weaknesses, and are therefore best used in different situations. Motion detection is the most reliable of the two methods, however in the current implementation both of them are working together, complementing each other to get an almost perfect ball tracking. A white background screen was used to get better contrasts in the images, making them easier to analyze.

Color detection As the ball has a singular color and the background has another singular color a way to detect the location of the ball is to search for the color of the ball in every analyzed frame. This is done by setting a color gradient range that is possible for the ball to have and then say that every pixel in the frame that has a color within this color range is the ball. The color space used to set the colors that should be detected is not the traditional RGB representation but instead the HSV (Hue, Saturation, Value) representation. This representation is used because it is easier to change the color to be detected, as only the *Hue* value has to be changed to go from the color red to the color blue. See the HSV color cylinder in figure 12

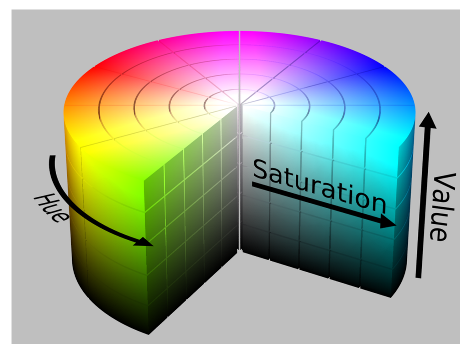


Figure 12. The HSV color cylinder. An alternative way to describe the color spectrum.

When this method is used it is important that only the ball has a color within the desired color range, such that no double readings will occur, otherwise it is hard to determine which one of the objects is the ball.

A problem when using color to detect the ball is to have a light source that gives the exact same light every experiment, as the perceived color of the ball can change with the light source. To counter this problem an extra light source with a constant bright light, has been acquired and is aimed toward the area of interest where the ball could be detected. Since the light from the light source is bright, there are some problems with glares on the ball. The glares can make the color of the ball go outside the color range and the tracking will be lost.

[4] https://www.adafruit.com/product/1411

This is fortunately not a big problem and can be avoided if the light sources are set up right. The largest problem with the color detection method is though to track the ball when it is moving fast. The camera has an electronic shutter that is slow. Therefore when the exposure time is long, fast-moving objects are displayed as a blur. The blur makes the color of the ball merge into the background, thus making it much harder to detect the ball.

Using color detection for locating the ball works fairly well but it can be unreliable if the light sources are not strong enough, or if the object that should be tracked is moving fast. Figure 13 and 14 shows an example of how the color detection method works. In this example the method is configured to find the blue ball, and as can be seen in Figure 13 a yellow circle has been added on top of it to mark a successful localization.

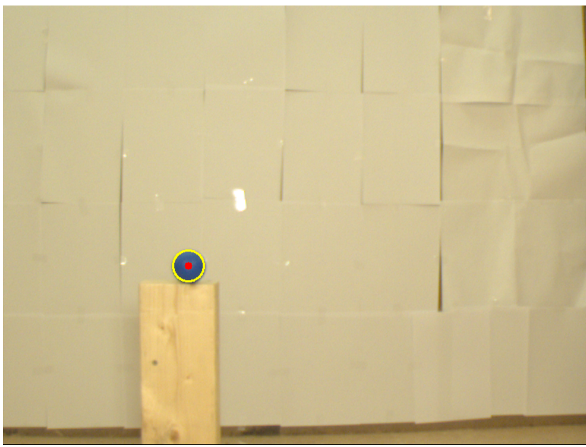


Figure 13. The frame to be analyzed for the color detection method. A circle has been added on top of the ball to see that the method has located it.

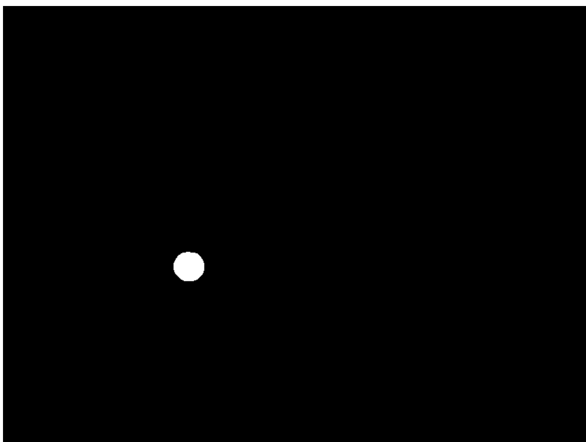


Figure 14. The mask matrix of the color detection matrix with the input frame from figure 13. As seen from this mask the ball has been located.

Motion detection As the ball will always move along the captured frame a motion detection method has been implemented to get the location of the ball. To detect the movement of the ball, two consecutive frames are analyzed to see what has changed between them. Since the only thing that should have changed between the two frames is the ball, it can easily be located. This is done by doing an absolute differential between the two images. In pixels where nothing has changed between the two frames the result will be zero and where the ball has moved it will be a non-zero value. By then passing this result though a threshold function a mask can be constructed to get a matrix where the pixels have a maximum value (255) where the something has moved and a minimum value (0) where nothing has moved. By then analyzing the mask it is possible to detect contours and groups of pixels with high values to get where something has moved to get the location of the moving ball.

An example of the motion detection is presented in Figure 15 and 16. In figure 15 a green square has been drawn on top of the moving ball to show that the motion detection method was able to locate the ball. Figure 16 shows which pixels has changed between two consecutive frames. The white color means that change has happened and black means that no change has happened. Since it is known that these changes are due to the moving ball it is possible to infer the ball location. However, there are two white contours in the mask, and there is only one moving ball. This is because one of these is the ball's location in the previous frame. But as the ball was located in the previous time-step it is possible to discard one of the read locations if it matches the previous ball location. Then only one of the readings is of interest which corresponds to the new location of the ball.

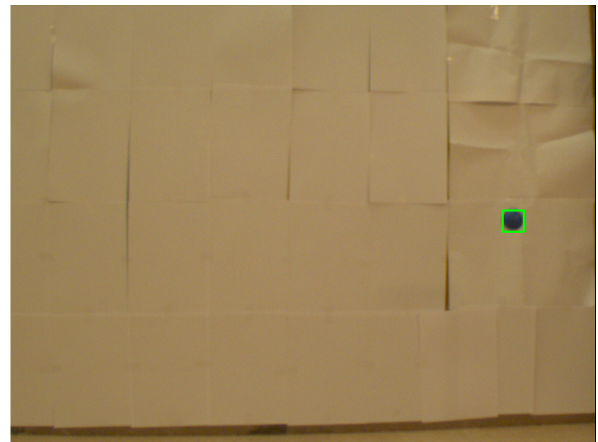


Figure 15. The frame to be analyzed for the motion detection method. This is the latest frame in a sequence of frames that is to be analyzed. A square has been added on top of the ball to see that the method has located it.

4.3 Kalman Filter

The Kalman filter used in this project is slightly different from the basic Kalman filter. The Kalman filter used in this project change over time compared to the simpler version which is stationary. The Kalman filter uses three matrices (R, P, Q) that can be modified to get the desired performance. If the measurements are known to be precise the diagonal elements of the R matrix should be low. If the system model is good with low amounts of disturbance the diagonal elements of the Q matrix should be chosen small. The P matrix should be initialized with low values on the diagonal elements if the

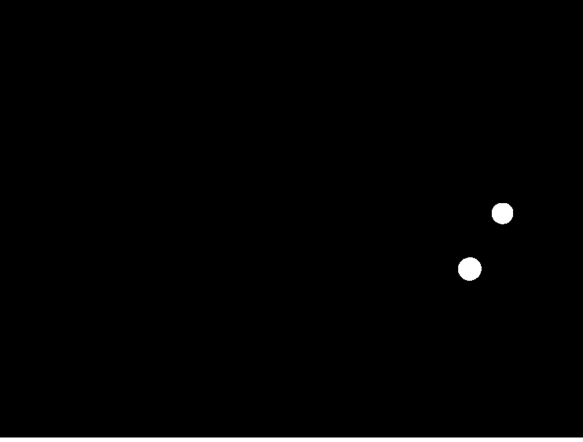


Figure 16. The mask matrix of the motion detection matrix with the input frame from figure 15. As seen from this mask, the method has two alternatives of the ball location to choose from. But as the computer knows the previous location of the ball, this location is discarded and one unique ball location is left.

initial states are precise. Each time-step these equations are executed:

$$P_k = \Phi P_{k-1} \Phi^T + Q \tag{6}$$
$$S_k = C P_k C^T + R \tag{7}$$
$$K_k = P_k C^T S_k^{-1} \tag{8}$$
$$\hat{x}_{k+1} = \Phi \hat{x}_k + \Gamma + K_k (y_k - C \hat{x}_k) \tag{9}$$
$$P_k = (I - K_k C) P_k \tag{10}$$

Equation (9) is the standard Kalman filter Equation for discrete time and Equations (6), (7), (8) and (10) is for updating the K matrix. During each iteration matrices K and P will be updated. The diagonal elements in the P matrix gets smaller due to number of measurement acquired. One important variable choice is the initial values of the states. Since the initial velocity and position differ a lot between throws it is set by the two first measurements where speed is calculated by $\dot{X}_{init} = (X_2 - X_1)/h$ and $\dot{Y}_{init} = (Y_2 - Y_1)/h$ where h is the sample time and $X_{init} = X_2$ and $Y_{init} = Y_2$.

4.4 Robot Arm Control

Feed forward control was implemented to control the robot arm. It was implemented as a mapping, from the point where the ball is caught, to the output signal to the servos. This mapping was divided into two parts, calculating the angles of the robot joints, such that the robot will catch the ball, and mapping these angles to the corresponding signal to send to the servos.

Servo Control The servos were controlled using an Adafruit servo shield for Arduino. In addition to this a servo control library from Adafruit was used to write to the servos. As input the servo library takes a frequency at which it sends pulses to the servos (60 Hz was used in the project) where the length of the pulse control the servos. The pulse length, i.e the input signal to the servos, was calculated from the desired servo position via linear regression. The linear regression takes the maximum and minimum value of the servo angle as well as the corresponding pulse lengths and outputs a pulse length for a desired servo angle.

Angle Calculations As mentioned in the Section 2.2 there are two methods of calculating the correct angles. For this project.we ch The to modelisoos is a ftwo hercle arims as a tsct and easy solution which generated two correct placements of the arm that achieve the same end result. As can be seen in Figure 5. In order to decide which of these intersection points to use, an evaluation is done where the angles for both solutions are compared to the servo limitations. If the initial solution is feasible, it will be used by the algorithm. If not, a flip is done where the second solution is tested. If the second intersection point is not reachable by the servos, there simply does not exist a solution where the robot can catch the ball. Finding the intersection points is done with Equations (11-15) where Figure 17 shows the scenario.

$$d = a + b \tag{11}$$
$$h = \sqrt{r_0^2 - a^2} \tag{12}$$
$$a = \frac{(r_0^2 - r_1^2 + d^2)}{2 \cdot d} \tag{13}$$
$$P2_{(x,y)} = P0_{(x,y)} + a \tag{14}$$
$$P3_{(x,y)} = P2_{(x,y)} + h \tag{15}$$

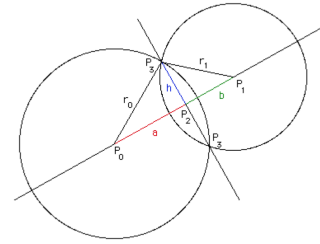


Figure 17. Intersection of the two circles.

This flip is done by calculating a line between the origins of the circles and mirroring the point of intersection to the other side of the line. The reason for doing it this way instead of just evaluating the second solution of the intersection problem is that it can be used for the FABRIK algorithm, it is also fast with regards to calculations.

The equation for doing this flip can be seen in equation (17), where *a* is the end point of the segment closest to the base, *v* is the line from the origin of the first circle to the end point of the second segment and *y* is the normalized vector along *v*.

$$y_{(x,y)} = \frac{a_{(x,y)} \cdot v_{(x,y)}}{\mid v_{(x,y)} \mid} \cdot v_{(x,y)} \tag{16}$$
$$(x,y)_{flipped} = 2y_{(x,y)} - (a_{(x,y)} + (x,y)_{base}) \tag{17}$$

Lastly, there needs to be a conversion from coordinates to angles as all calculations are done by considering the coordinates of each vector instead of angles. This was done by using Equation (18).

$$angle = arctan(x_1 \cdot x_2, x_1 \times x_2) \quad (18)$$

Where x_1 and x_2 are the two vectors between which the calculated angle is located.

5. Results

5.1 Image Analysis

The resulting method to locate the ball works very well. It detects the ball by both searching for the color of the ball and the movement of it. Most of the information comes from the motion detection as the color detection method looses track of the ball at higher speeds. In Figure 18 the current and the last three measurements of the ball location is plotted in the sample image. Hence, it is clear that the image analysis gives very accurate measurements of the ball location.

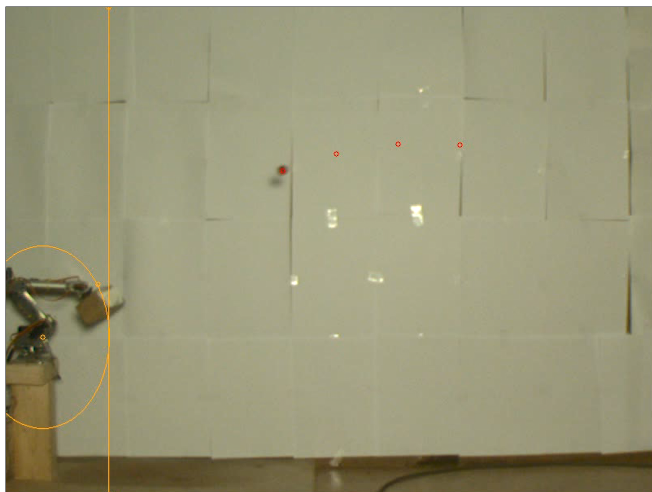


Figure 18. A sample image with the last four measured ball locations plotted with red dots.

5.2 Trajectory Planning

When the Kalman filter was used to predict the states, some problems occurred when calculating the trajectory. From time to time the estimates were very poor and resulted in an inaccurate catching point.

The method with polynomial regression worked very well and resulted in a good trajectory (which can be seen at Figure 19). The results can also be seen in the videos located in the documentation in the Git repository[5].

5.3 Robot Arm accuracy

The accuracy of the arm movement is determined by using a tape measure to find the position of the cup as well as drawing a line in the stream from the camera to be at the x value that the end effector should be at.

These measurements resulted in an maximal error of approximately 1 centimeter, which is good enough. It has never been an issue when testing as the cup is wider than the maximal error as well as the error being smaller than 1 centimeter for most cases.

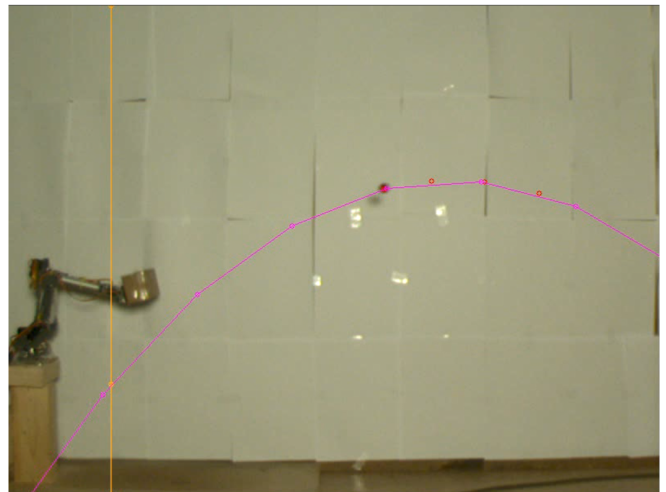


Figure 19. A sample image with the polynomial trajectory plotted.

5.4 Overall Results

In conclusion, the robot arm works very well and is able to catch balls within a large spectra. There is no problem with either speed or accuracy and the goal of the project has been fulfilled. However, there are always aspects that could be improved upon. Those are described in Section 6.3. Videos of the functioning system catching balls can be found in documentation of the Git repository for the project[5].

6. Discussion

6.1 Choice of camera

At the beginning of the project an ordinary webcam was used to track the ball. The resolution of this camera was good and it was easy to use in the system but it only delivered a video stream with 30 Frames per second. This did not seem to be a problem at first but when the Kalman filter and trajectory calculation was implemented a problem with frame skips was significant. As the sample time of the control system is 20 Hz and the video stream delivers a new frame with a frequency of 30 Hz, one frame can be skipped and the measured relative distance between two samples will be wrong. To minimize this frame skip effect the high speed camera with 200 FPS was used.

Figure 20 and 21 visualizes how a faster camera decreases the problem of frame skips and make the measurements more exact. In the figures the time is on the x-axis. The black vertical lines represents the time the program samples the video stream (20 Hz), and the red arrow shows when the camera is taking an image. In a sample the last image of the video stream is used. The black line show how the ideal sample time-line should look like, with equal time between the samples. The red line shows the resulting sample time-line. In Figure 20 with the 30 FPS camera it is clearly visible that the image frame used at the sample time is not actually the frame at that specific time, but rather the image from the last time the camera captured a frame. This will give an inaccurate location of the ball at the specific time as the image used might show the current location of the ball. It is also possible to see that if there are two measurements within the time of two samples one of them

[5] https://gitlab.control.lth.se/regler/FRTN40/2018/GroupF

is skipped which will make the ball look like it has traveled longer between these two samples.

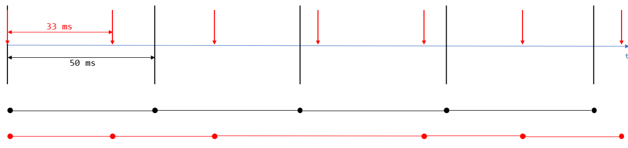


Figure 20. Figure of the the time-line of the camera feed samples (30 FPS) and the control system samples (30 Hz). The time is along the x-axis. The vertical black lines are the time of the control system samples and the red arrows shows when the camera takes a new image. The black line underneath shows what the ideal sample time-line should look like, and the red shows the resulting sample time-line as an image used to locate the ball might not be of the exact time of the sample.

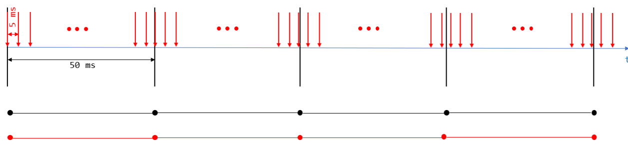


Figure 21. Figure of the the time-line of the camera feed samples (200 FPS) and the control system samples (30 Hz). The time is along the x-axis. The vertical black lines are the time of the control system samples and the red arrows shows when the camera takes a new image. The black line underneath shows what the ideal sample time-line should look like, and the red shows the resulting sample time-line as an image used to locate the ball might not be of the exact time of the sample.

Figure 21 shows how a high speed camera can minimize this problem. When there are many more frames delivered from the camera, the last image will still be used. As the time is now much smaller between the frames the effect of the frame skips will also be much smaller. As seen in the same figure, the black line is how the samples should look ideally, and the red line is the resulting sample time-line. As seen here, the red time-line is very close to the ideal, and the frames used at the sample time for image analysis is therefore very accurate and gives the location of the ball at the exact time. The effect of the delay is though not eliminated completely. The location of the second to last dot on the red time-line is not aligne with location of the black dot, but it is significantly closer than in figure 20.

6.2 Kalman vs polynomial regression

One large problem encountered during the project was the camera and the measurements which was explained in Section 6.1. Even though a high speed camera was used to decrease this problem, there were still some visible effects of this problem which caused a lot of troubles for the Kalman filter when estimating the states.

Since the measurements were performed indoors and the ball selection was made such that air drag would not affect the ball that much, the speed in the horizontal direction should almost be constant when airborne. This would mean that the distance between each sample in the horizontal direction would be the same. This caused troubles for the Kalman filter when estimating the velocity. This could be accounted for in the R-matrix but since the measurements were very precise this did not help that much. Due to this problem the results for the Kalman filter varied each run.

Since the measurements of the position were so precise, a new idea for the trajectory prediction was implemented. This idea was to use polynomial regression to fit a second degree polynomial with the measurement points. This gave a very precise trajectory of the ball from just 4-5 measurements.

6.3 Future work

Although the group is very satisfied with the result and the end result fulfills what the project specifications said it should there are certainly improvements and more work that could be done.

The main improvement that would have been interesting to implement is if the robot was able to catch balls in a 3D spectrum instead of only 2D. This would however require one more camera in the image analysis. Parts of the trajectory calculation would need to be modified for more than x and y coordinates. One large change is also when calculation the angle of the robot arm. Instead of using two circles as the solution, there would need to be two spheres with the intersection in the shape of a circle.

Another improvement that could be made is the introduction of feedback control to the system. With a colourful marker at the side of the robot arm cup, the position of the cup could be measured and controlled for. The feedback would allow for correction of any inaccuracies in the mapping from the catching point to the output to the servos. It would also introduce new challenges. More image analysis would have to be implemented. Where to place the control is also an issue to debate. Should the reference point be changed to accommodate for the error, or perhaps directly on the control signal. These problems would all require a lot of time, and therefore feedback control was put on hold as something to maybe work on if there was time and need.

There is also the issue of speed. As seen in the results the robot arm is not in a position to catch the ball well ahead of time, but often barely makes it to the right position. Feedback control when the ball is not yet in position would not be beneficial as the supply voltage, and thus the speed of the servos, cannot be controlled. This leaves a small window of time where feedback control would be beneficial and the robot might not have time to adjust for this control. Another reason for feedback not being implemented is the results of the robot arm accuracy. Because the error of the cup position is small enough to be a non-factor when catching the ball, feedback control has been deemed not worth implementing.

6.4 Concluding thoughts

Although a few things that could be improved upon, the final result of the project was satisfactory. The goal set at the beginning of the project was reached and the group is very satisfied with the result. For many of the challenges several solutions

were tested and as a result valuable lessons were learned and the goals of the project were achieved.

References

[1] M. Linderoth. *On Robotic Work-Space Sensing and Control*. PhD thesis. Department on Automatic Control, Lund University, 2013. ISBN: 978-91-7473-670-0.

[2] R. Paul. *Robot manipulators: mathematics, programming, and control : the computer control of robot manipulators.* MIT Press, Cambridge, MA., 1981. ISBN: 978-0-262-16082-7.

CSTR

Erik Bolin[1] **Jens Franzon**[2] **Peder Klint**[3] **Gustav Persson**[4]

[1]elt14ebo@student.lu.se [2]elt14jfr@student.lu.se [3]elt14pkl@student.lu.se
[4]bte13gpe@student.lu.se

Abstract: The project task was to run a continuous stirred tank reactor (CSTR) in continuous mode. This means that a liquid should be flowing through the tank without interruption. The main goal was to investigate and (if possible) realize a control structure for running an exothermic reaction process. For this to be possible the liquid has to have the correct temperature and concentration when flowing out from the tank. The main focus in this project was the design, implementation and realization of a controller and software and not the mechanical design of the process. The majority of the controller design were made on a simulated model in Simulink in Matlab and the fine tuning was made by experimenting on the real process. The final result was a process controlled by two PI controllers, one for the level and one for the temperature.

1. Introduction

A continuous stirred tank reactor (CSTR) is a batch reactor with a mixing device, such as an impeller, to achieve a homogeneous mix of reactants, see figure 1. In this project the lab process on which the experiments were carried out, was borrowed from the Department of Automatic Control, see figure 2. The equipment is used for laboratory sessions where non continuous batch processes are used, which means that there is no flow through the tank while mixing. The goal with this project and report was to investigate if it is possible to run the tank in continuous mode, i.e. having a constant flow through the tank while mixing. To achieve this it is necessary to control the level of fluid in the tank and the temperature of the fluid flowing out. This is because, when mixing two or more reactants in a batch reactor, an endothermic or exothermic process usually occurs. To simulate this, a heating element (immersion heater) and a cooling element (Peltier element) is attached to the process. In this project, an exothermic process was chosen to be modeled on the basis that it is more common in a chemical reaction. An exothermic process means that when two different liquids are mixed, heat will be released. If the mixture is not cooled, the temperature of the mixture will keep rising and the process will therefore become unstable. To keep the process stable, the Peltier element is used to cool the exothermic reaction while running the process. The process has one flow into the system that is possible to control with a DC-pump. The outflow can be controlled with either a DC-pump or by a mechanical valve. A temperature sensor is mounted at the bottom of the tank to be able to measure the temperature of the liquid. A DC-motor is attached to a mixer to be able to mix the content of the tank. All I/O is accessible through serial communication to a Linux-computer with Simulink.

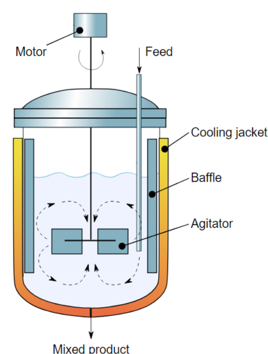


Figure 1: Example of a typical continuous stirred-tank reactor (CSTR) [1].

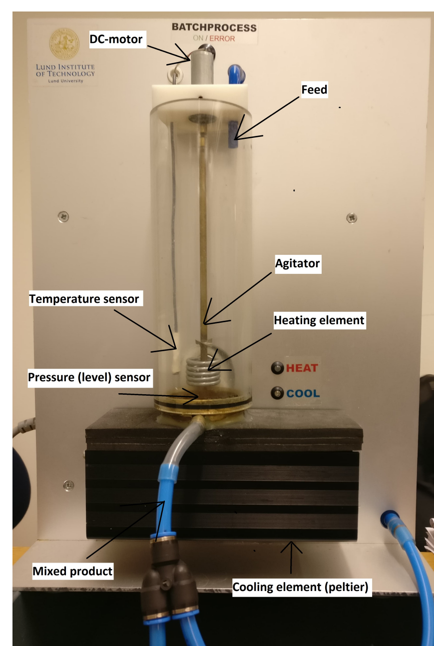


Figure 2: The lab process used for modeling and testing in this project.

2. Theory

In order to model the tank, the following assumptions were made:

1. The liquid in the tank is perfectly mixed. The temperature of the outflow equals the temperature of the liquid in the tank.
2. The inflow is the same as the outflow i.e. the volume in the tank is kept constant.
3. The density of the incoming liquid is equal to the density of the simulated mixed fluid in the tank. Because water is the liquid used on the real process, the density is chosen to be equal to the density of water and that it is constant.
4. The specific heat of the incoming liquid and the liquid in the tank are equal and they are constant and equal to the specific heat of water.
5. All other energies except for the heat of the inflow, heat of the outflow, heat added and taken from the heater and Peltier element are neglected in the energy balance model.

Volume model The volume is modeled based on mass balance calculation

$$\frac{dm}{dt} = m_{in} - m_{out}$$

where m_{in} is the mass flow rate in and m_{out} is the mass flow rate out. Rewritten in a form in terms of volume, density and volume flow rate

$$\frac{d(\rho V)}{dt} = q_{in} \cdot \rho_{in} - q_{out} \cdot \rho_{out}.$$

With a cylindrical tank with constant cross-section area the change in volume can be calculated with the change in height

$$\frac{d(\rho V)}{dt} = \frac{d(\rho A h)}{dt} \Leftrightarrow A\frac{dh}{dt} = q_{in} - q_{out}. \tag{1}$$

Temperature model The temperature is modeled in a similar way as the volume but with energy balance instead of mass balance

$$\frac{d}{dt}(mc_p\Delta T) = m_{in}c_{p_{in}}\Delta T_{in} - m_{out}c_{p_{out}}\Delta T_{out} + Q$$

where ΔT is temperature difference, c_p is the specific heat capacity and Q is the heat contributed from the cooler and heater. Rewriting the mass flows and expanding the temperature differences

$$\frac{d}{dt}(\rho A h c_p(T - T_{ref})) = q_{in}\rho_{in}c_{p_{in}}(T_{in} - T_{ref}) - q_{out}\rho_{out}c_{p_{out}}(T_{out} - T_{ref}) + Q.$$

Assumption 3 and 4 leads to the following equations

$$A\frac{d}{dt}(h(T - T_{ref})) = Ah\frac{dT}{dt} + A(T - T_{ref})\frac{dh}{dt} = q_{in}(T_{in} - T_{ref}) - q_{out}(T_{out} - T_{ref}) + \frac{Q}{\rho c_p}.$$

Assumption 1 and 2 results in the following equation

$$Ah\frac{dT}{dt} = q_{in}(T_{in} - T) + \frac{Q}{\rho c_p}. \tag{2}$$

Chemical reaction As mentioned in the last section the temperature is dependent on Q which can be expanded into a cooler and heater part

$$Q = Q_c + Q_r$$

where Q_c is the heat which the cooler transferring out of the system and Q_r is the heat contribution from the chemical reaction. The chemical process is modeled as an Arrhenius equation which is a common way to describe temperature dependence of reaction rates [2]. The Arrhenius equation is given as follows

$$Q_r = (-\Delta H_R)Vkc_A$$

where k is the reaction rate constant, $-\Delta H_R$ is the change in enthalpy and c_A is the molar concentration. k is described by the following equation

$$k = k_0 e^{-\frac{E}{RT}}$$

where k_0 is a frequency factor, E is the activation energy and R is the universal gas constant. The molar concentration also changes over time. Under the assumption that $q_{in} = q_{out}$, which we do in assumption 2, the change in molar concentration can be modeled as following

$$\frac{dc_A}{dt} = \frac{q_{in}}{Ah}(c_{A_{in}} - c_A) - kc_A. \tag{3}$$

Models From equation 1, 2 and 3 the model for the CSTR-system can be described by the following differential equations

$$\dot{h} = \frac{1}{A}(q_{in} - q_{out}) \tag{4}$$

$$\dot{T} = \frac{q_{in}}{Ah}(T_{in} - T) - \frac{Q_c}{\rho c_p A h} + \frac{(-\Delta H_R)k_0 c_A}{\rho c_p}e^{-\frac{E}{RT}} \tag{5}$$

$$\dot{c_A} = \frac{q_{in}}{Ah}(c_{A_{in}} - c_A) - k_0 c_A e^{-\frac{E}{RT}}. \tag{6}$$

The control signals are the inflow rate from the pump, q_{in}, and the heat generated from the cooler, Q_c. Limitations on the cooler leads to that Q_c is limited between 0-10 Watts and is desired to be close to 5. The measurement signals are the temperature in the tank, T, the height of the liquid in the tank, h, and the molar concentration c_A.

Stationary points The stationary points have to fulfill that equation 4, 5 and 6 all equal 0

$$0 = \frac{1}{A}(q_{in_0} - q_{out}) \tag{7}$$

$$0 = \frac{q_{in_0}}{Ah_0}(T_{in} - T_0) - \frac{Q_{c_0}}{\rho c_p A h_0} + \frac{(-\Delta H_R)k_0 c_{A_0}}{\rho c_p} e^{-\frac{E_0}{RT_0}} \tag{8}$$

$$0 = \frac{q_{in_0}}{Ah_0}(c_{A_{in}} - c_{A_0}) - k_0 c_{A_0} e^{-\frac{E_0}{RT_0}} \tag{9}$$

where h_0, T_0, q_{in_0}, Q_{c_0}, c_{A_0} and E_0 are the desired stationary process states. With these state conditions the other variables can be calculated

$$\begin{aligned} m_t &= -\left(\frac{q_{in_0}}{Ah_0}(T_{in} - T_0) - \frac{Q_{c_0}}{\rho c_p A h_0}\right) \\ m_c &= \frac{q_{in_0}}{Ah_0}(c_{A_{in}} - c_{A_0}) \\ c &= \frac{(-\Delta H_R)k_0 c_{A_0}}{\rho c_p} \\ t &= \frac{E_0}{R}. \end{aligned}$$

By introducing these new variables equation 8 and 9 can be rewritten

$$\begin{cases} m_t = c e^{-\frac{t}{T_0}} \\ m_c = \frac{\rho c_p}{(-\Delta H_R)} c e^{-\frac{t}{T_0}} \end{cases} \Leftrightarrow \begin{cases} \ln(c) = \frac{1}{T_0} t + \ln(m_t) \\ \ln(\frac{\rho c_p}{(-\Delta H_R)} c) = \frac{1}{T_0} t + \ln(m_c). \end{cases}$$

The remaining constants can be calculated as

$$\begin{aligned} k_0 &= \frac{e^{\frac{1}{T_0} t + \ln(m_c)}}{c_{A_0}} \\ (-\Delta H_R) &= \frac{\rho c_p}{k_0 c_{A_0}} e^{\frac{1}{T_0} t + \ln(m_t)}. \end{aligned} \tag{10}$$

Linearization Linearization of equation 5 and 6 around the stationary points h_0, T_0, q_{in_0}, Q_{c_0}, c_{A_0} and E_0 results in the following Jacobian matrices

$$\begin{aligned} A &= \begin{bmatrix} -\frac{q_{in_0}}{Ah_0} + \frac{(-\Delta H_R)k_0 c_{A_0}}{\rho c_p} \cdot \frac{E_0}{RT_0^2} e^{-\frac{E_0}{RT_0}} & \frac{(-\Delta H_R)k_0}{\rho c_p} e^{-\frac{E_0}{RT_0}} \\ -k_0 c_{A_0} \cdot \frac{E_0}{RT_0^2} e^{-\frac{E_0}{RT_0}} & -\frac{q_{in_0}}{Ah_0} - k_0 e^{-\frac{E_0}{RT_0}} \end{bmatrix} \\ B &= \begin{bmatrix} -\frac{1}{\rho c_p A h_0} \\ 0 \end{bmatrix}. \end{aligned} \tag{11}$$

With A and B a state space model for the linearized system can be found on the form

$$\begin{aligned} \dot{z} &= Az + Bu, \quad z = \begin{bmatrix} T - T_0 \\ c_A - c_{A_0} \end{bmatrix}, \quad u = Q_c \\ y &= Cz, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \end{aligned} \tag{12}$$

Stability Plotting the real part of the eigenvalues of the A matrix, in the linearized process model in equation 11, over t yields the relationship shown in figure 3. A positive real part of an eigenvalue corresponds to an unstable system which in this case causes thermal runaway. We observe that the system is stable for $t < 50$.

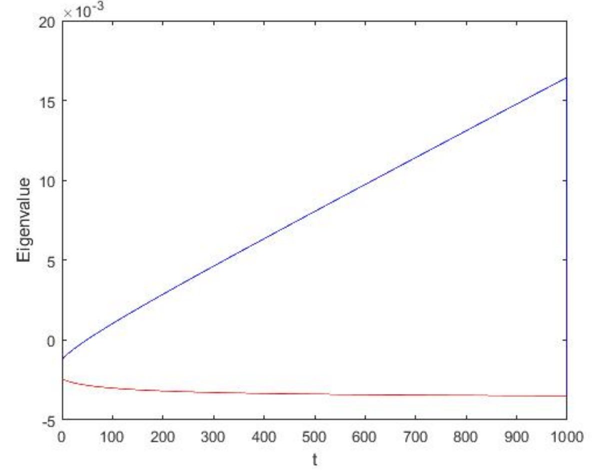


Figure 3: Real part of the eigenvalues of the process over t.

Below in table 1 all variables have been gathered. The values of the variables with predefined values have been added and the variables without values are to be determined.

Table 1: Variable table

Description (symbol)	*Value*	*Units*
Mass flow rate (m)		$\frac{g}{s}$
Volume flow rate (q)		$\frac{mm^3}{s}$
Density (ρ)	0.001	$\frac{g}{mm^3}$
Volume (V)		m^3
Cross-section area of the tank (A)	2827.4	mm^2
Liquid height in the tank (h)		mm
Temperature (T)		°C
Specific heat capacity (c_p)	4.186	$\frac{J}{g \cdot °C}$
Heat (Q)		J
Change in enthalpy (H_R)		$\frac{J}{mol}$
Substance concentration (c_A)		$\frac{mol}{mm^3}$
Frequency factor (k_0)		$\frac{1}{s}$
Activation Energy (E)		$\frac{J}{mol}$
Universal gas constant (R)	8.314	$\frac{J}{mol \cdot °C}$

3. Control Design

Different controllers were designed and compared to yield a result as good as possible. In the first iteration of the simulated model, the level was regulated with a PI controller and the temperature with different types of linear-quadratic regulators, LQR in short. In the second iteration of the simulated model, the level was regulated with a PI controller and the temperature with a P controller.

3.1 LQR

The temperature state can be measured and therefore a state-feedback controller was a natural choice to examine. The different types of LQRs calculated was LQR and LQI for the complete linearized model and LQR for a model reduced with balanced model reduction. To calculate the state-feedback gain for the LQR a state space model on the form as in equation 11 and 12 had to be calculated. The LQR for the complete linearized model resulted in control signals way over the limits and was excluded. The LQI and LQR for the reduced model resulted in state-feedback $L_i = [0.1766, -0.0277]$ and $L = 0.1852$ respectively. The results of responses with decreasing steps from the simulated LQR and LQI models can be seen in figure 4 and figure 5.

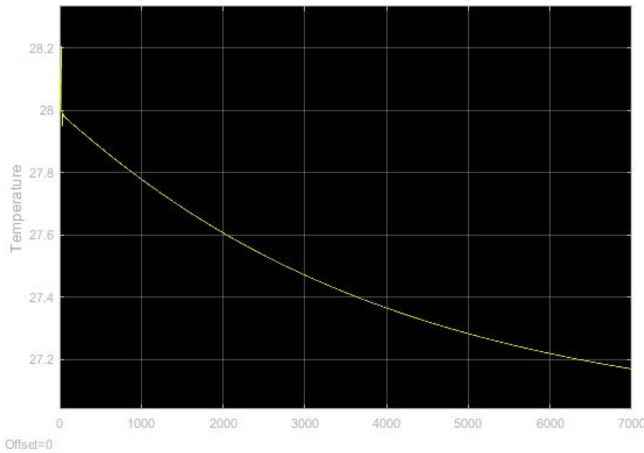


Figure 4: Impulse response LQR.

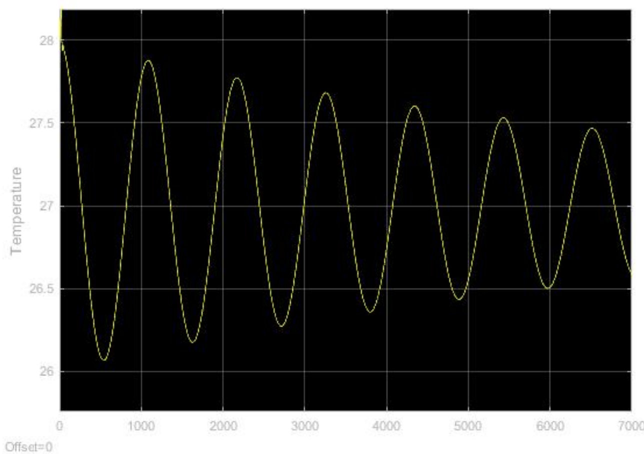


Figure 5: Impulse response LQI.

The control signal, Q_c, was kept within the desired region, close to 5, but the control signal of the LQI had an oscillatory behaviour which exceeded even the limitations of Q_c.

3.2 PID

Level For level control, a PI controller with gain 0,3 and T_i = 0,001 gave the best results, see figure 11

Temperature To control the temperature, a simple P controller with gain 0,3 gave much better results than the LQR and LQI controllers, see impulse response in figure 6.

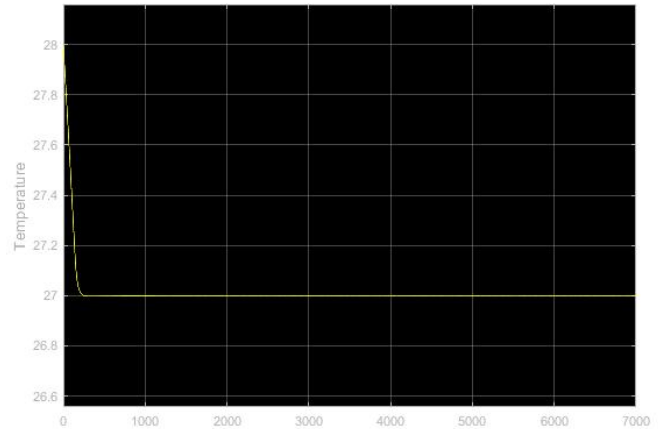


Figure 6: Impulse response P controller.

Stability Plotting the real part of the eigenvalues of the closed loop process over t yields the relationship shown in figure 7. It can be seen that the closed-loop system is stable for $t < 166$. This enables the process to run continuously with a t that is roughly tree times larger than the open-loop system.

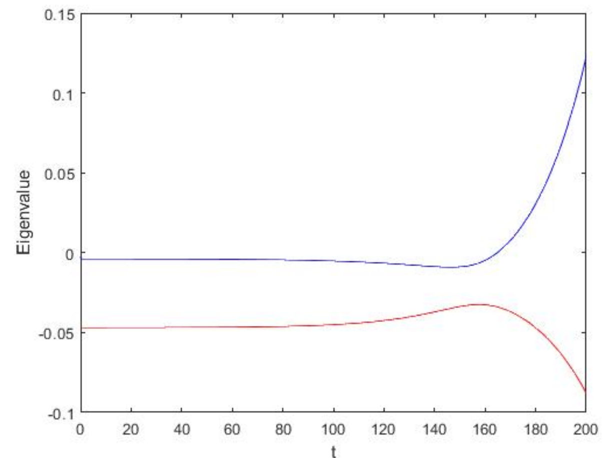


Figure 7: Real part of eigenvalues for the closed-loop system over t.

4. Implementation

In order to design controllers for the process, all of its components had to be identified and modelled. The process had several restrictions and non-linearities such as cooling power and voltage thresholds for the pumps. These variables were identified through different tests on the process. To find different equilibrium temperatures, the cooling element was kept at half of its effect, as the cooling was the controlling parameter, while the heating element was iterated through different values.

Further, a complete model of the system was set up in Simulink in Matlab. This enabled a faster design of the systems controllers.

Fitting terms in the chemical reaction was chosen from figure 8 and figure 9. In the real model, the exothermic chemical reaction was simulated with a 150 W heating element whilst the cooling in the system was carried out by a 40 W Peltier element. Due to the low power of the Peltier element, the cooling power was the process' restricting factor. To minimize the impact of the restriction, a flow through the system and a mixing volume as low as possible was coveted. To get the flow through the system in the desired span, a head of a syringe, which restricts water flow, was connected to the inlet and outlet tubes, see figure 10. To solve the problem with the inlet pump yielding no flow until a certain voltage was supplied, an offset voltage had to be added to the control signal. This effectively means that the pump always operates in its linear region.

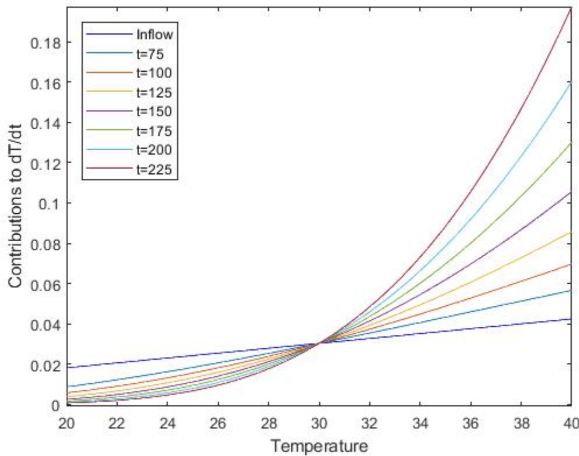


Figure 8: Different chemical processes at constant height.

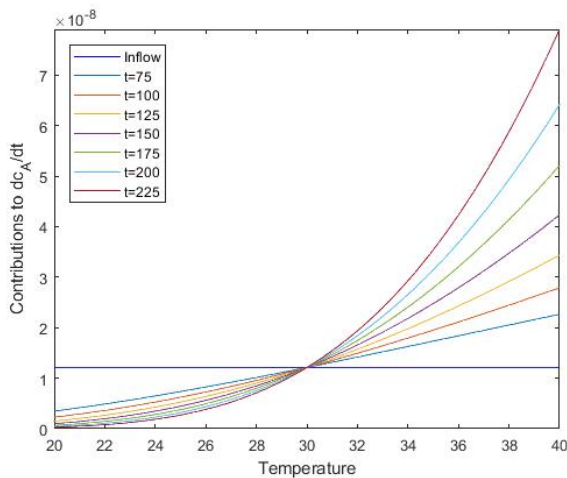


Figure 9: Different molar concentration processes at constant height.

Figure 10: Head of a syringe connected to the outlet tube.

5. Result

The system's threshold values were identified through simple experiments on the real process, see table 2.

Table 2: Threshold values

Process	*Value*
Pump_in_start	2.65 V
Pump_in_stop	2.40 V
Pump_out_start	1.8 V
Pump_out_stop	1.3 V
Mixer	3 V
Level	60 mm

The pumps have a threshold value where they will start if they are turned off and another threshold value where they will stop if they are already turned on.

Stationary points The stationary points had to be chosen such that they fitted the limitations on the real process and that the system calculated from equation 11 and 12 was still stable. With the stationary points in table 3 the stationary temperature T_0 could be chosen freely in the range where the real process could operate and where the closed-loop system was stable.

Table 3: Stationary values

Desired stationary points that can be matched on the real process that has resulted in stable closed-loop systems.

Description	*Desired stationary point*
Inlet flow rate (q_{in_0})	240
Liquid height in the tank (h_0)	70
Cooler effect (Q_c)	5
Inflow concentration ($c_{A_{in}}$)	20
Mixed concentration (c_{A_0})	10
Change in enthalpy (H_R)	0.0042
Frequency factor (k_0)	0.0162
Activation Energy (E_0)	581.98

Level control Unlike the simulated model, the level of the real process' tank is regulated with a PI controller with a set

offset. The offset was chosen to be +2.4 V as this was just under the voltage required to turn on the pump in its linear region. Due to the throttled inlet, a buffer of inlet fluid arose at a certain pump voltage. When using a PI controller with an aggressive integral part, this buffer caused a oscillating behaviour. On the other hand, the buffer had a positive impact when using a PI controller with a less aggressive integral part, as it eliminated the occurrence of overshoot. This yielded a good result on the level control. The PI controller generating the best result had a gain of 0.3 and a T_i of 0.001.

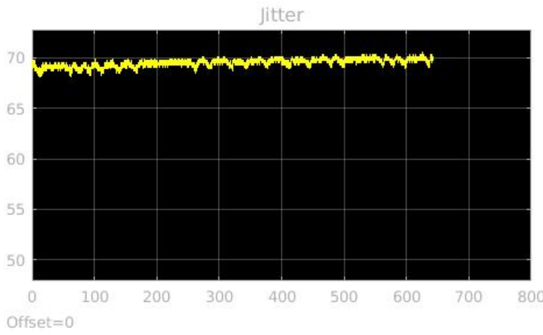


Figure 11: Level control on the real process with a PI controller with reference value 70. X-axis and y-axis shows time [s] and temperature [°C], respectively.

Temperature control For temperature control of an exothermic process, the simulated model showed acceptable results with LQR of the reduced system. All simulations and calculations were based on the stationary points in table 3. Although the simulated model yielded the best results with a simple P controller and the simulated LQR was very slow, it was decided to apply and compare both the LQR controller and the P controller on the real process. In simulation, all the controllers were performing well at stationary values but when applying the LQR controller on the real process the controller had, as in simulation, a very slow step response as can be seen in figure 12. The control signal is shown in figure 13 was too low but did not have aggressive oscillations.

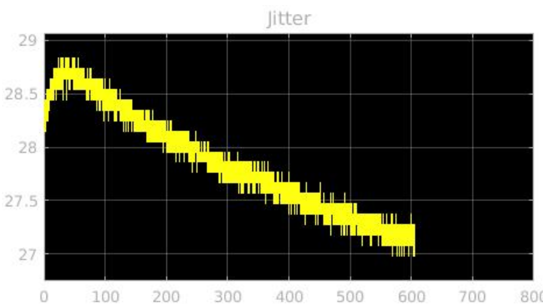


Figure 12: Temperature response with an decreasing step using LQR. X-axis and y-axis shows time [s] and temperature [°C], respectively.

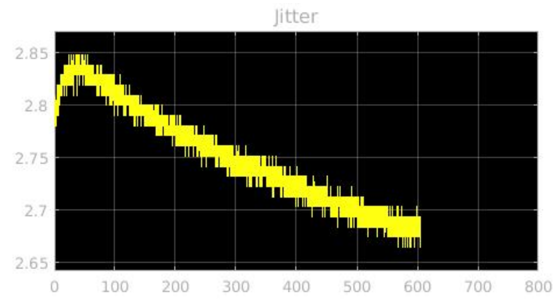


Figure 13: Control signal during step response using LQR. X-axis and y-axis shows time [s] and cooler power [W], respectively.

Instead the P regulator was applied with improved results; it was more stable and much faster. Further tuning on the real process resulted in a PI controller. The controller were then tested with different values of E_0, see figure 14, 15, 16 and 17.

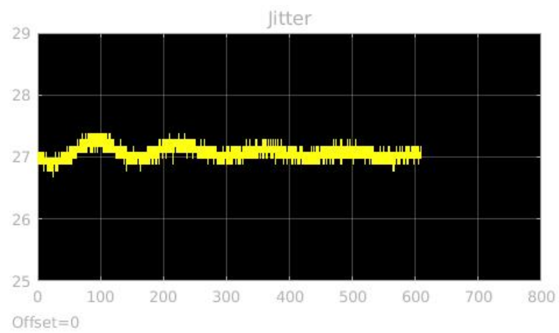


Figure 14: Temperature with PI controller and E_0 = 581.98. X-axis and y-axis shows time [s] and temperature [°C], respectively

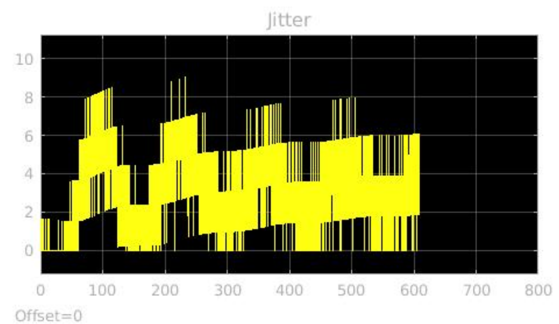


Figure 15: Control signal, Q_c, when E_0 = 581.98. X-axis and y-axis shows time [s] and temperature [°C], respectively

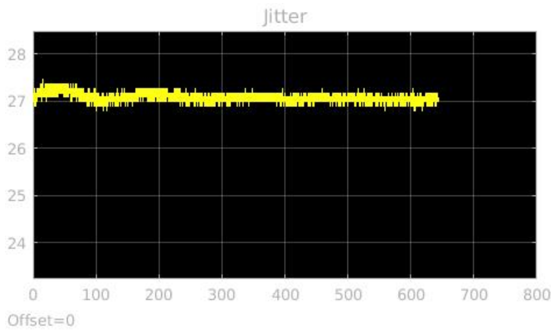


Figure 16: Temperature with PI controller and E_0 = 8314. X-axis and y-axis shows time [s] and temperature [°C], respectively

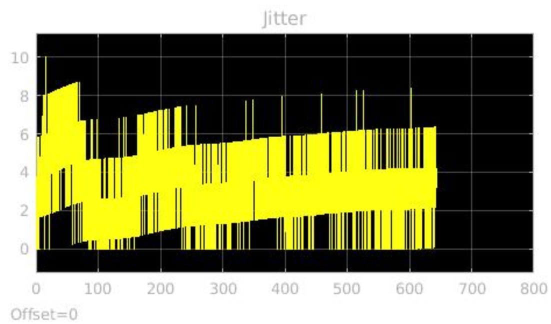


Figure 17: Control signal, Q_c, when E_0 = 8314. X-axis and y-axis shows time [s] and cooler control signal [Q_c], respectively

As can be seen in figure 18, 19, 20 and 21, the step responses for both positive and negative steps in temperature are stable. Even though oscillations arise from a decreasing step in temperature, they are kept within the limits. The control signal is quite aggressive but its average value is within the desired region.

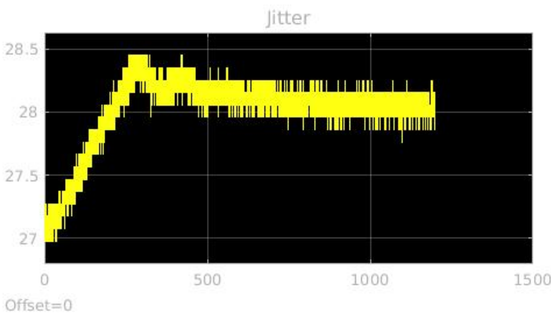


Figure 18: Response with an increasing step using PI controller. X-axis and y-axis shows time [s] and temperature [°C], respectively

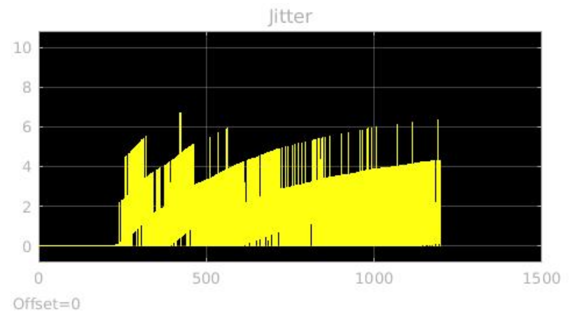


Figure 19: Control signal, Q_c, from an increasing step response using PI controller. X-axis and y-axis shows time [s] and cooler control signal [Q_c], respectively

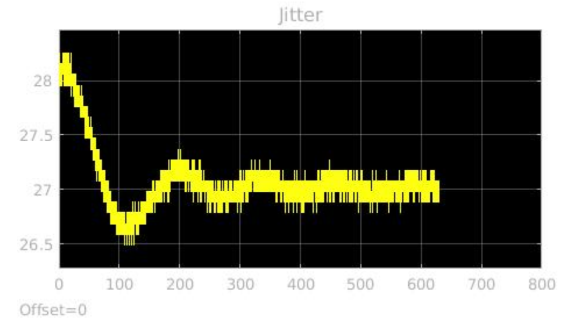


Figure 20: Response with a decreasing step using PI controller. Response with an increasing step using PI controller. X-axis and y-axis shows time [s] and temperature [°C], respectively

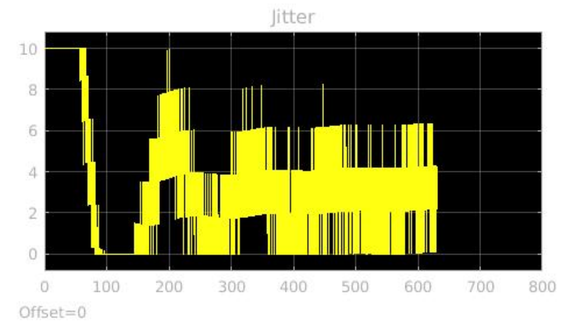


Figure 21: Control signal , Q_c, from a decreasing step response using PI controller. X-axis and y-axis shows time [s] and cooler control signal [Q_c], respectively

6. Discussion

The restricting factor of the system is without doubts the Peltier element. Even though the Peltier element has a power of 40 W its efficiency is only around 47 %, resulting in an effective power of 18.75 W. With a more efficient cooling element, the process could have been run with a bigger flow and at a higher stationary temperature. With the current cooling element, the flow of water into the system stands for 80 % of the cooling at 40 $C°$. At higher temperatures, the room temperature also comes into calculation. These contributing factors are why the process was chosen to run at 27 $C°$. When the process started at a stationary temperature, increasing E_0 did not result in a temperature increase. When having a high E_0 and

applying a step in temperature, the temperature increase exploded. It can be concluded that the PI controller can control highly unstable processes when starting at stationary , which was unexpected. This might be because of the cooling from the process' surroundings can increase its tolerance of more aggressive chemical reactions. However, as expected, a small increase in temperature resulted in an uncontrollable state.

As the real process is very slow, every iteration of tests on the system took very long time. Most of the controller tuning were therefore done in Simulink, which had both pros and cons. The pros are the time effectiveness and the ease of plotting results. The cons are that the simulated model doesn't take all the surrounding factors of the real process into count i.g. room temperature and changing temperature of flow into the system. Since there was no validation of the simulated model done to the real process its actual accuracy was unknown. This was noticeable when designing the controllers; a controller that appeared to work with good results in the simulation often did not work as well on the real process. Apart from the disturbances of room temperature and the changing temperature of the flow, the measurement signal for height and temperature also had a quite low resolution which naturally had an impact on how well the controller could perform. The apparent quantization visible in the control signal Qc stems from the controller amplifying the quantized measurement signal as well as a heavily compressed x-axis. To achieve a more accurate model for future work, several things have to be implemented, such as disturbances and a more thorough validation of process parameters.

Bibliography

[1] Daniele Pugliesi [GFDL] https://commons.wikimedia.org/wiki/File:Agitated_vessel.svg (Accessed 2018-12-5)

[2] Seborg, Dale E.; Edgar, Thomas F.; Mellichamp, Duncan A. *Process Dynamics and Control*. 2004.

Ball and Plate

Mathias Artursson[1] **Hampus Farmängen**[2] **Felix Lundbladh**[3]
Arantxa Juárez Pérez[4]

[1]tfy14mar@student.lu.se [2]hfarmangen@gmail.com [3]felixlundbladh@gmail.com
[4]arijp.94@gmail.com

Abstract: The main objective of the *Ball and Plate* project was to balance a ball on a resistive touch plate. An Arduino-based solution was used to enable easy controlling and reading of the hardware. Two Arduino-compatible servo-motors were provided, among other things, and a set up was designed and constructed. To enable the plate to tilt without swivelling; a rather expensive universal joint was provided and a MDF-model (Medium Density Fibreboard-model) was designed and laser cut to fixate the moving parts. The software design consisted of servo control, Kalman filtering, and PD-regulation with support from theoretical calculations. This way, the main objective was fulfilled, which was to get the ball to balance in the middle of the plate.

1. Introduction

The main objective of the *Ball and Plate* project was to balance a ball on a resistive touch plate, as can be seen in Figure 1. This has been done by obtaining the position of the ball from the touch plate. Arduino calculates the needed angles for the servo motors to maintain the set-point of the ball by calculating the position of the ball in terms of coordinates.

The touch plate was supported on an universal joint and a MDF-model was designed in CAD and laser cut. The software design consisted of servo control, Kalman filtering, and PD-regulation designed by theoretical calculations. The PD that was designed to control the servo motors was calculated mathematically, however, the values had to be modified for suitable performance.

Figure 1. Real system setup.

2. Modeling

The control system for the ball and plate process in 1D is illustrated by Figure 2.

The control system shown in Figure 2 is represented as a block diagram in Figure 3.

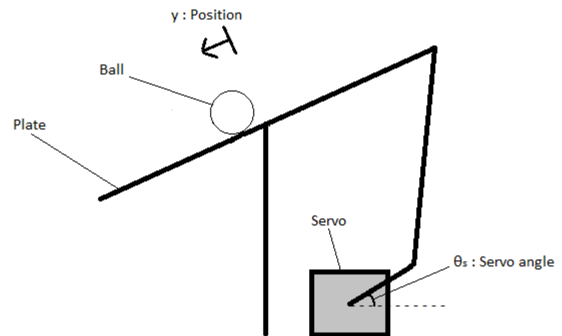


Figure 2. Control system for controlling the position y of the ball with the servo angle θ_s.

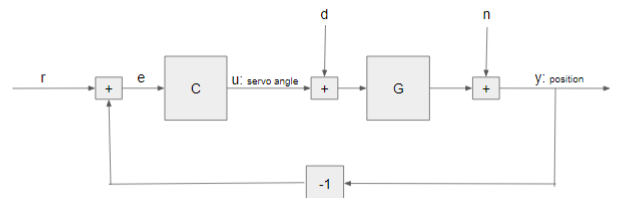


Figure 3. Assumed model for the control system, where r is the reference, e the error, C the controller, u the control signal (servo angle), d is a disturbance, G is the process (from servo angle to ball position), n is noise, and y is the output (ball position).

2.1 Process

The aim was to find a model for the process G in Figure 3. The process identification presented here was inspired by [3].

First, it was needed to relate the plate angle θ_p to the forces acting upon the ball, with this input it was possible to position the ball. This is illustrated in Figure 4.

Assuming no friction, from Newton's second law, (1) can be derived

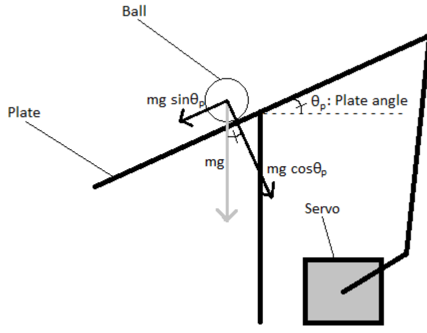


Figure 4. Illustration of how the plate angle θ_p will affect the forces on the ball. θ_p is the plate angle, m the mass of the ball, and g the gravity constant.

$$m\ddot{x} = mg\sin\theta_p. \tag{1}$$

Assuming small plate angles, (2) will hold

$$\sin\theta_p \approx \theta_p. \tag{2}$$

Combining (1) and (2), one could relate the plate angle θ_p to the position of the ball x according to (3)

$$\ddot{x} = g\theta_p. \tag{3}$$

Figure 5 illustrates how the servo angle θ_s affects the plate deviation $h(\theta_s)$.

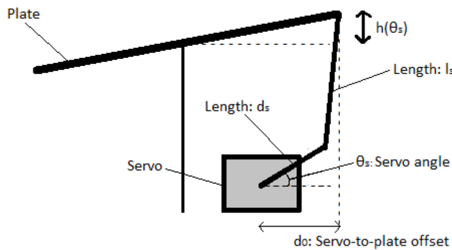


Figure 5. Illustration of how the servo angle θ_s affects the plate deviation $h(\theta_s)$. The two lengths of the servo arm are denoted as d_s and l_s, and the servo-to-plate deviation is denoted as d_0.

Figure 6 shows the result of applying trigonometry on Figure 5.

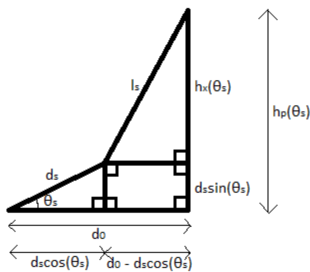


Figure 6. Trigonometric interpretation of Figure 5.

Using the Pythagorean theorem, it was possible to identify the length $h_x(\theta_s)$ as seen in (4)

$$h_x(\theta_s) = \sqrt{l_s^2 - (d_0 - d_s\cos\theta_s)^2}. \tag{4}$$

The distance between the servo and the plate, $h_p(\theta_s)$, will therefore be as (5)

$$h_p(\theta_s) = h_x(\theta_s) + d_s\sin\theta_s. \tag{5}$$

If the plate deviation was zero at a servo angle offset u_0, then the plate deviation could be described as (6)

$$\begin{aligned} h(\theta_s) = h_p(\theta_s) - h_p(u_0) = \sqrt{l_s^2 - (d_0 - d_s\cos\theta_s)^2} + \\ d_s\sin\theta_s - (\sqrt{l_s^2 - (d_0 - d_s\cos u_0)^2} + d_s\sin u_0). \end{aligned} \tag{6}$$

Figure 7 relates the plate deviation $h(\theta_s)$ to the plate angle θ_p.

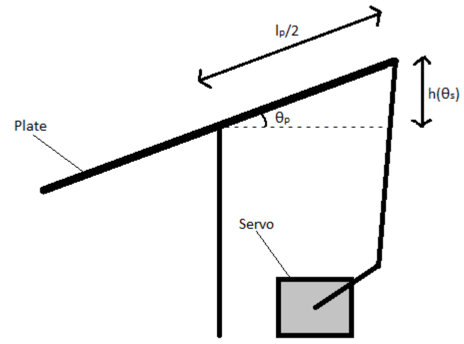


Figure 7. Illustration of how the plate deviation $h(\theta_s)$ relates to the plate angle θ_p.

Looking at Figure 7, it was clear that the relation between the plate deviation $h(\theta_s)$ and the plate angle θ_p will be as in (7)

$$\sin\theta_p = \frac{2h(\theta_s)}{l_p}. \tag{7}$$

If one once again assumes small plate angles according to (2), one will instead have the relation according to (8)

$$\theta_p = \frac{2h(\theta_s)}{l_p}. \tag{8}$$

Finally, combining (3), (6) and (8), an equation which relates the servo angle θ_s to the position of the ball was obtained. This relation will be as in (9)

$$\begin{aligned} \ddot{x} = \frac{2g}{l_p}(\sqrt{l_s^2 - (d_0 - d_s\cos\theta_s)^2} + d_s\sin\theta_s \\ -(\sqrt{l_s^2 - (d_0 - d_s\cos u_0)^2} + d_s\sin u_0)). \end{aligned} \tag{9}$$

Now, to transform this into a state space form, one could set the following:

$$x_1 = x, x_2 = \dot{x}, u = \theta_s, y = x,$$

which it results in (10)

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{2g}{l_p}(\sqrt{l_s^2 - (d_0 - d_s\cos\theta_s)^2} + d_s\sin\theta_s \\ &\quad -(\sqrt{l_s^2 - (d_0 - d_s\cos u_0)^2} + d_s\sin u_0)). \\ y &= x_1.\end{aligned} \tag{10}$$

Since, the state space (10) was nonlinear, it needed to be linearized. Linearizing around $x_1 = 0$, $x_2 = 0$, $u = u_0$ and $y = 0$, it results in the linearized state space given by (11)

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{2g}{l_p}(\frac{(d_s\cos u_0 - d_0)d_s\sin u_0}{\sqrt{l_s^2 - (d_0 - d_s\cos u_0)^2}} + d_s\cos u_0)\Delta u \\ y &= x,\end{aligned} \tag{11}$$

where $\Delta u = u - u_0$.

If the state space (11) was written in matrix form, it results in (12)

$$\begin{bmatrix}\dot{x}_1 \\ \dot{x}_2\end{bmatrix} = \overbrace{\begin{bmatrix}0 & 1 \\ 0 & 0\end{bmatrix}}^{A}\begin{bmatrix}x_1 \\ x_2\end{bmatrix} + \overbrace{\begin{bmatrix}0 \\ K1\end{bmatrix}}^{B}\Delta u \qquad y = \underbrace{\begin{bmatrix}1 & 0\end{bmatrix}}_{C}\begin{bmatrix}x_1 \\ x_2\end{bmatrix}, \tag{12}$$

where K_1 was according to (13)

$$K_1 = \frac{2g}{l_p}(\frac{(d_s\cos u_0 - d_0)d_s\sin u_0}{\sqrt{l_s^2 - (d_0 - d_s\cos u_0)^2}} + d_s\cos u_0). \tag{13}$$

Rearranging (12) into the transfer function G from a servo angle to the position of an axis, it results in (14)

$$G(s) = \frac{K_1}{s^2}. \tag{14}$$

From (14), it is concluded that the process could be approximated by a double integrator.

2.2 Controller

The process G was going to be controlled by a PD controller of the form given by (15)

$$C(s) = K + sK_d. \tag{15}$$

The design for the controller was inspired by [4]. The design philosophy was as follows. First, pole placement was used on the closed loop system to get a hint of suitable magnitudes for the PD parameters K and K_d. A value for K was set within this proximity, and K_d was chosen for simulated step responses have an overshoot of ~14%. The pole configuration of the resulting PD parameters was then applied to the real process with the Kalman filter. The distance to the origin in the s-plane for the pole configuration was then tuned, resulting in new PD parameters with the same properties as the ones before, but resulting in a different speed for the controller. A suitable speed of the controller could be evaluated through the time constant of the closed loop system and through testing. The time constant τ for a closed loop system could be defined as the time it takes for the system to reach 63% of a new reference [7].

Combining (14) for the process with (15) for the controller, the closed loop system transfer function G_{cl} will be as in (16)

$$G_{cl} = \frac{K_1(K + sK_d)}{s^2 + K_1(K + sK_d)}, \tag{16}$$

where K_1 is given by (13).

If desired poles of the closed loop system were a and b, where the real part of these were assumed to be negative, one could then match these with the PD parameters by identifying coefficients in the characerastic equation of the closed loop system. The characteristic equation of the closed loop system was given by the denominator of (16). Matching the poles with the PD parameters results in (17).

$$K = \frac{ab}{K_1}, K_d = -\frac{(a + b)}{K_1}. \tag{17}$$

The design philosophy could be applied for the y-axis as an example below, using MATLAB.

First, parameters for the y-axis were set:

```
ptom = 430*2/0.2112; %number of pixels
per meter
g = 9.82*ptom;
ds = 0.0285*ptom; %length of small
servo arm
ls = 0.12*ptom; %length of large
servo arm
lp = 0.2112*ptom; %length of plate
u0 = (pi/180)*15; %servo angle offset
d0 = 0.0373*ptom; %servo-to-plate offset
```

Note that the position output had pixels as unit length. Therefore, all model parameters with length meter were redefined in lengths of pixels instead.

Setting $a = b = -10$ and using (17), it results in the PD parameters being as follows:

$$K = 0.0098, K_d = 0.0020$$

The speed of this system could be evaluated by simulating a step response of the closed loop system for these values of K and K_d. The closed-loop system was created with $K = 0.0098$ and $K_d = 0.0020$ according to (16). This system was then discretized using the MATLAB function *c2d* with the sample time of 50 ms. *c2d* transformed a continuous-time transfer function to a discrete-time one. Step responses were then simulated using the MATLAB function *step*. Doing the above, it results in the step response, according to Figure 8.

When tuning for a suitable overshoot, it may then be suitable to choose $K = 0.01$ and vary K_d around 0.0020. Further on, the closed loop system according to equation (16) was created with $K = 0.01$ and $K_d = [0.0016 : 0.0022]$. This system was then discretized using the MATLAB function *c2d* with the sample time of 50 ms. Step responses were then simulated

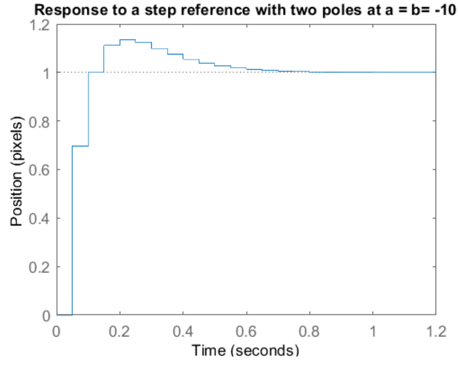


Figure 8. Simulated step responses using discretized closed loop transfer functions with $K = 0.0098$ and $K_d = 0.0020$. It could roughly be seen that $\tau \approx 50 - 100$ ms. This might be too fast because the system was about as fast as the sampling time. It will need be slowed down when changing the distance to the origin for the poles.

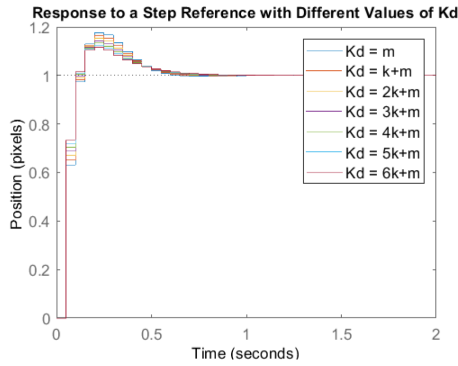


Figure 9. Simulated step responses using discretized closed loop transfer function with $K = 0.01$ and $K_d = [0.0016 : 0.0022]$ ($m = 0.0016, k = 0.0001$).

using the MATLAB function *step*. Doing the above, it results in the step responses according to Figure 9.

Looking at Figure 9, then for $K_d = 0.0019$ ($K_d = 3k + m$) it results in the simulated overshoot being ~14%.

Choosing $K = 0.01$ and $K_d = 0.0019$, it results in the poles a_y and b_y being as in (18)

$$\begin{aligned} a_y &= -9.6862 + 2.8527i \\ b_y &= -9.6862 - 2.8527i \end{aligned} \tag{18}$$

When using $K = 0.01$ and $K_d = 0.0019$ on the real system with a Kalman filter, the controller seemed way too fast.

The poles could be slowed down by defining the poles as (19)

$$\begin{aligned} a_{ys} &= a_y y_s \\ b_{ys} &= b_y y_s, \end{aligned} \tag{19}$$

where y_s was a design parameter.

Slower values for K and K_d could be received by inserting poles (18) into (19) with $y_s = (0 : 1)$, and calculating new K and K_d values using (17). Doing this with $y_s = 0.32$ it resulted in the new PD values K_y and K_{dy} which seemed much better. K_y and K_{yd} can be seen in (20)

$$\begin{aligned} K_y &= 0.0010 \\ K_{yd} &= 6.0800 \cdot 10^{-4}. \end{aligned} \tag{20}$$

A simulation of the step response and the step disturbance using PD parameters K_y and K_{yd} can be seen in Figures 10 and 11.

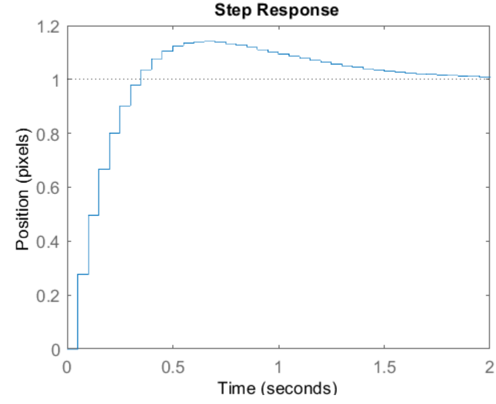


Figure 10. Step response using PD parameters K_y and K_{yd} according to (20) for the discretized closed loop system. It could roughly be seen that $\tau \approx 150 - 200$ ms, which seems like a more reasonable time constant compared to $50 - 100$ ms previously.

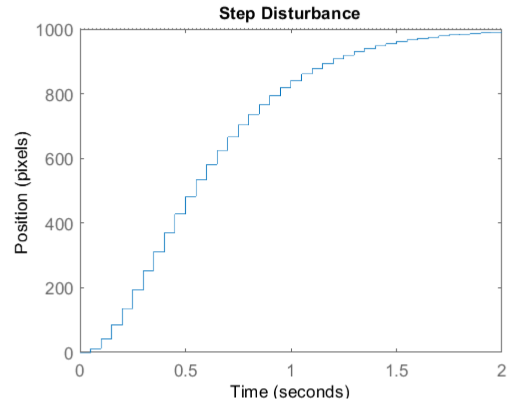


Figure 11. Step disturbance using PD parameters K_y and K_{yd} according to (20) for the discretized closed loop system. A step disturbance may be, for example, someone putting their hand on the plate so that the servo motor angle initially gets reduced by 1 rad.

Bode plot for the closed loop system $(GC/(1 + GC))$ using the new PD parameters can be seen in Figure 12.

3. Electro-Mechanics

As previously mentioned, the hardware played a key role in the project. This is what the control theory was based on and what the software was controlling. Most of the hardware was provided, including the resistive touch plate, servo motors, rods and rubber bands. Remaining components and materials were easily acquired. The first step of this project was to create a model to assemble the motors and plate on. MDF (Medium

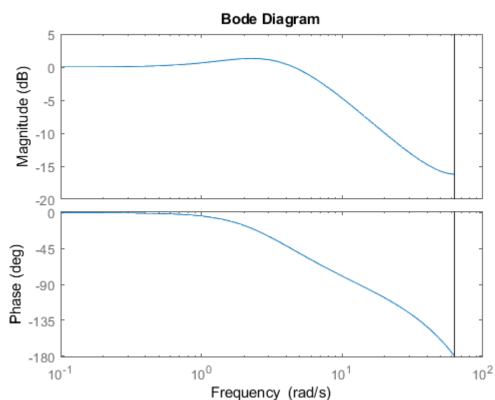


Figure 12. Bode plot using PD parameters K_y and K_{yd} according to (20) for the discretized closed loop system. It looks like a typical double integrator where the gain starts to decrease at the two poles for K_y and K_{yd}.

Figure 13. Universal joint

Density Fibreboard) was chosen for its low cost and its ability to be laser cut. A model was created with CAD (Computer Aided Design) and was designed with flat MDF-plates to be assembled into a 3D structure and can be seen in Figure 1.

Resistive touch plate and fence To acquire a measurement of the ball position on the plate a resistive touch plate was used. The control and reading of its values is presented in Section 4.3. The plate was mounted on a simple MDF base and secured by four corner pieces. On top of this was a frame of MDF mounted that functioned as a fence when testing and tuning the balancing. This turned out to be a key element for the efficiency of PD-tuning.

Universal joint To balance the plate, a universal joint was used which can be seen in Figure 13. This works much like a ball joint except that it hinders swivelling. This simplifies the control task, since any force from the servos that was not completely perpendicular to the universal joint will turn the table and the axis of which the forces were acting upon will be offset. This was probably the most expensive part of the whole design since it was completely made of metal and very complex. This was a part that could be worked around with an alternative design, such as adding another servo to over-constrain the system (this would add complexity in the terms of inverse kinematics) or a simpler joint (probably less reliable).

Servo Motors The servos were standard MG-995 and a CAD-Model can be seen in Figure 14. As the specifications below describe, they were powerful enough to exert the forces required for efficient control. The voltage, however, required an external power supply since the Arduino was not powerful enough to supply two servos with current.

- Axle type: Futaba
- Size: 40.7 x 19.7 x 42.9mm
- Weight: 55 g
- Speed: 0.2s / 60° (4.8V)
- Stall: 10Kg/cm (4.8V)
- Working temperature: 0 – 55°
- Deadband: 10us
- Voltage: 4.8 – 7.2

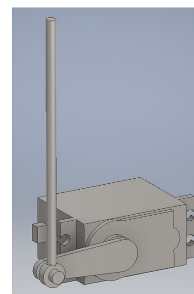


Figure 14. Motor. Tower Pro MG995 DIGI HI-SPEED.

At the end of each servo, a rod was attached that controls one axis of the tilt of the plate. These were supplied from the start and the origin was not known, they were length-adjusted to fit our model and as some play at the plate attachment.

Backlash rubber bands The rubber bands were placed on the servo arm to reduce or even eliminate backlash, as there was some space in the attachment between the rods and the plate. Control was very hard with backlash since precision in the plate angle was impossible in small movements. Using rubber bands was a common way of eliminating backlash, and they were adjusted to exert moderate force on the plate in order to not burden the servos too much and still reduce the backlash efficiency.

4. Control

The system was implemented using an Arduino and the coding language was the Arduino version of C++. A few external libraries were used in the code in order to simplify handling hardware, filtering and control. The touch plate was a bit of a niche product and was hence handled manually using input and output to and from the analog pins.

4.1 Libraries

Some libraries were imported from GitHub. These are:

- Arduino PID Library [2]
- Servo Library [5]
- Simple Kalman Filter [6]

4.2 Global variables, setup and loop

In C++ you may declare global variables which were available to all functions. This concept was used in order to clean up the code and gather all information which was commonly changed at the top of the source code. There were also two functions which were native to the Arduino version of C++, `setup` and `loop`. The `setup` was run once on start up and the `loop` was thereafter run continuously.

Global variables The global variables contained both constant and dynamic values initialized by the keyword *#define*. These were optimized during compile time and was not to be changed during run time. Other variables could be changed. Examples of declares are the pins (constant) and the position variables (dynamic).

Setup In the setup the servos and the PD-controllers were started.

Loop The loop function was the most important part of the code since this was where most of the code executes from. It was run continuously (apart from when interrupted, which was not used here) as long as the Arduino has power. Here both the control calculations and the movement of the servos were handled.

4.3 Resistive touch plate

The touch plate was one of the most important components, both hardware- and software-wise. In order to get the position of the ball, a function called `updateCoord` was implemented. The measurement worked with inputs and outputs from and to the analog pins using voltage division. Below follows a brief explanation of the code and the actual code for getting the X-coordinate. In order to find the X-position we actually measured the resistivity on the Y-pins. Since we were sharing the same space when measuring X- and Y-coordinates we needed to set the Y2 (and X2) to tri-state[1].

After setting the X2 to tri-state, a voltage divider was formed between the X1 and X2. The voltage was then read at the Y1-pin and scaled in order to fit the wanted resolution and place (0,0) in the centre[1]. Since there were a lot of measuring noise and disturbances the input values need to be filtered. The filter of choice was a Kalman-filter. The actual value of the coordinate was fed into the filter and an estimation was returned and stored for future usage.

```
void updateCoord(){
    pinMode(Y1, INPUT);
    pinMode(Y2, INPUT);
    digitalWrite(Y2, LOW);
    pinMode(X1, OUTPUT);
    digitalWrite(X1, HIGH);
    pinMode(X2, OUTPUT);
    digitalWrite(X2, LOW);
    actual_x = ((analogRead(Y1)) / (1024
        / Xresolution) - 500);
    est_x = (double)
        x_kFilter.updateEstimate
        (actual_x);
    ...
}
```

4.4 Kalman filtering

As previously mentioned, the measurement from the resistive touch plate was contaminated with high levels of noise. It was therefore required to filter this noise and the tool we use for this was a Kalman filter. This LQE (Linear Quadratic Estimator) was a two-step process where an estimate was predicted from measured state variables and a predetermined noise level, which were then compared to the actual measurement. The weighting within the algorithm was then redistributed. This recursive algorithm was at first very unstable, but once it has gone through enough iterations the estimated measurement was very stable and reliable. This filtering came with a price, however. Depending on the level of noise specified in the setup process the estimate either could be either slowed down, eliminating all noise, but resulting in very slow estimates which quickly become inaccurate in our system, or sped up which does not eliminate enough noise. The complications became prominent when the filter was paired with the PD controller. Our estimated values for the controller do not take the filtering effects into account which means that the filtering of the system needs to be manually tweaked to reach the optimal level.

We have used a library to implement the filtering which has worked well. An example of the code is shown where the Kalman filter was created by initiating the `SimpleKalmanFilter` with the setup (Measurement Uncertainty, Estimation Uncertainty, Process Noise). The filtering was then done by the function `UpdateEstimate` where the actual measurement was in the data. The function then returns the estimated value as a double which was fed to the PD controller.

```
SimpleKalmanFilter
    y_kFilter(10, 10, 0.01);
...
est_x = (double)
    x_kFilter.updateEstimate
    (actual_x);
```

The Kalman filter was tuned quite aggressively and an example of the obtained measurement data compared to the filtered is shown Section 5.

4.5 PD-control

When implementing the actual controller, a PD library was used for convenience. The library takes doubles, hence floating numbers were used instead of fixed-point arithmetic. The system was capable of running using floating numbers with an update frequency of up to 500 Hz, which was more than enough for the application. The library made the usage of the PD very simple and intuitive. The code below demonstrates the usage of the library and handling the control. One could simply create an object and use its specified functions. Worth to mention is that references were passed into the constructor, making it possible for the past references value to be altered meaning the functions rarely returns a value, but alters the value stored at the referenced address directly. In C++ this is common practice since it is much cheaper than caching and returning values as is commonly done in Java.

Below follows an example of the implementation of the X-axis. This code may be duplicated and slightly modified in order to implement stabilization for the Y-axis as well. The variables needed throughout the program were declared and the K-values and loop_period were initialized since these will never change. A PD-object was instantiated passing the variables as references using the &-sign. Since the K-values were not to be changed, these were passed by value.

```
double actual_x, setpoint_x, est_x;
double Kp_x = .00060025, Ki_x = .0,
    Kd_x = .000392;
PID pid_x = PID(&est_x, &radians_x,
&setpoint_x, Kp_x, Ki_x, Kd_x, DIRECT);
static unsigned short loop_period = 50;
unsigned long time_stamp;
Servo servo_x;
```

Assumptions of small angles of the servos were made when developing the control model, hence output limits were set quite narrowly. Another benefit of the output limits was that the system will limit itself slightly, making the testing a bit of an easier process. The loop period of the system was fed to the PD and it was turned on by setting it to automatic.

The set-point was set to zero since the ball should (at least for now) stay in the middle. The servo was initialized to pin number ten and a time stamp was read in order to know when the setup was finished, and the main loop could begin.

```
void setup(){
    pid_x.SetOutputLimits(−40, 40);
    pid_x.SetSampleTime(loop_period);
    pid_x.SetMode(AUTOMATIC);
    setpoint_x = 0;
    servo_x.attach(10);
    time_stamp = millis();
}
```

The main loop was where the actual control happens and thanks to the earlier implemented and imported functions and libraries this was a very smooth process. The first thing to do was to update the coordinates by calling the function `updateCoord`. This would update the estimate used in the PD-computations.

In order to keep the measurements as fresh as possible when calculating the function `Compute` was called on the PD object. This will take the estimate updated in `updateCoord` and change the value of the variable called *radians_x*. The radians were then converted to degrees and pressed together with a manual calibration to the servo motor. The servo motor will then change its angle quickly, albeit with a slight delay.

In order to calculate the integral part correctly the loop period needed to be constant. This was handled by a busy wait where some work may be done in the future. After the loop period has been reached, a new time stamp was taken and the control loop restarts.

```
void loop(){
    updateCoord();
    pid_x.Compute();
    servo_x.write(radians_x*
        rad_to_deg + 87);

    while ((time_stamp +
        loop_period) > millis()){}
    time_stamp = millis();
}
```

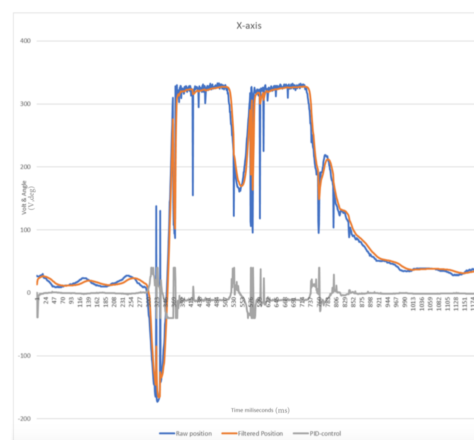


Figure 15. Measurements from failing X-axis

5. Results

One of the most complicated parts of the project has been to achieve adequate values in the PD. Mathematically, the values that were obtained had to be slightly modified in order to achieve the desired stability. The final values used in the control are shown below:

$$K_x = 0.0035, K_{xd} = 0.002 \qquad (21)$$
$$K_y = 0.0010, K_{yd} = 0.0005.$$

It is worth mentioning that the control of the X-axis has been problematic. Every now and then (very irregularly) the servo shuts down for half a second, resulting in the ball rolling to one side. Therefore it was not possible to obtain a correct measurement on the X axis since the motor did not work correctly. This made it impossible to obtain coherent results. The raw signal did not continuously follow the filtered signal, as can be seen in Figure 15. The large spikes in the position measurements mean that they were not entirely filtered out by the Kalman filter and this affects the control in a negative way. The servo has been switched for a new one, but with the same result. Another power supply has been added to feed both servos with enough current but with no effect. Finally, the servo that controlled the X-axis was changed. Three different servos were tried and the best one was chosen.

In Figure 16 shown below, it can be seen how the Kalman filter was tuned out to be quite aggressive, as well as an example of the obtained measurement data compared to the filtered. However, it can be observed how the filtered signal follows the raw signal correctly.

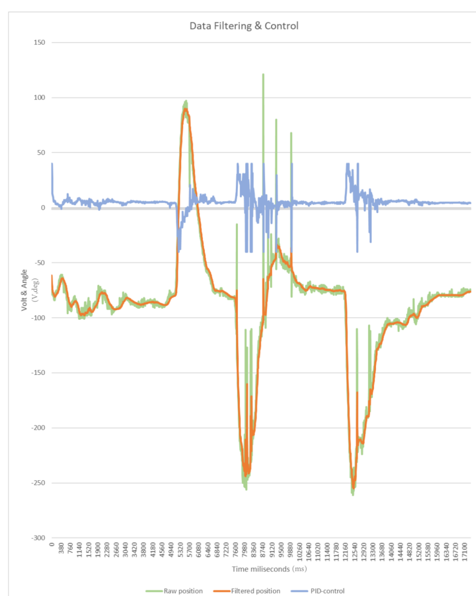


Figure 16. Position filtering and PD-control example on y-axis

6. Discussion

There are many things that could be improved if the project was redone or was taken by students repeating the same project in the following years. For example a new version could be made out of more rigid material and therefore allow a better control. Now, with the current version, when the servos were functioning, the base moved and that destabilized the ball control. A possible solution would be to make a stiffer base, either by putting weight on it or making it thicker (different layers of wood superimposed).

The 3D model could be improved allowing a better grip of the servos; better holes to place the Arduino and the joystick; incorporating all necessary features for a sturdier and portable design. Also, the ball joint, one of the most expensive parts of the project, could be replaced by using three servos instead of two. This way we would achieve the same goal for less money. Logically, it would be necessary to reprogram the data collection, its interpretation and the movement of the servos. Finally, the length of the rod that was attached to the servos of the new version of the model could be changed for a shorter length allowing us to achieve greater precision. Also decreasing the radius of the servo lever allows for a better resolution in controlling angles. The new version would be designed based on its control, leading to a more accurate mathematical description of the hardware.

In addition, there are different things that could be changed to improve the global operation of the project. For example, implementing movement patterns, which has been done, but it could be improved and elaborated if the control was fine-tuned further. Also, it would be better to process the signal differently to get rid of the twitching, by using something other than a Kalman filter. It would be ideal to remove outliers, for example, using a median-filter. We could have used fixed-point arithmetic instead of floating numbers. This way the control would work much faster and more efficiently. Another thing that could be modified in the programming was the busy wait. In the current program, a certain time was expected in the loop, preventing the controller from doing anything else. This was not efficient, and another system could be used, such as interruptions or a timer. In addition, machine learning could also be used to calibrate the parameters that were used in the PD. It would be interesting to elaborate different programs to make the ball move, not just keeping it stabilized in the centre of the plate, but to do it for example move continuously in circles or to be able to draw a pattern that the ball will later follow.

Overall, we want to say that we are satisfied with the results. We have learnt a lot throughout the project due to all the problems that kept arising constantly, and which we were able to find satisfactory solutions for.

References

[1] *4-wire touch screen interfacing with arduino*. URL: https://www.instructables.com/id/4-Wire-Touch-Screen-Interfacing-with-Arduino/ (visited on 2018-12-18).

[2] *Arduino-pid-library*. URL: https://github.com/br3ttb/Arduino-PID-Library (visited on 2018-12-18).

[3] S. Awtar and K. C. Craig. *Mechatronic Design of a Ball on Plate Balancing System*. Department of Mechanical Engineering, Aeronautical Engineering and Mechanics, Troy, NY12180, USA, Unknown year.

[4] *Control tutorials for matlab & simulink*. URL: http://ctms.engin.umich.edu/CTMS/index.php?example=MotorPosition§ion=ControlPID (visited on 2018-12-04).

[5] *Servo library*. URL: https://www.arduino.cc/en/Reference/Servo (visited on 2018-12-18).

[6] *Simplekalmanfilter*. URL: https://github.com/denyssene/SimpleKalmanFilter (visited on 2018-12-18).

[7] *Time constant*. URL: https://www.controlglobal.com/blogs/controltalkblog/key-misunderstood-terms-for-control-system-dynamics-tips/ (visited on 2019-01-09).