Projects in Automatic Control 2017

Soltesz, Kristian; Cervin, Anton

2018

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

Projects in Automatic Control 2017

Kristian Soltesz
Anton Cervin
Editors

LUND
UNIVERSITY

Department of Automatic Control

Technical Report TFRT-7653
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 2018

Preface

This report contains the student papers describing the projects in the 2017 course *Projects in Automatic Control* (FRTN40). The course is given annually by the Department of Automatic Control, Lund University, during the second half of the fall semester, with the possibility of projects starting at other times upon agreement. It is an advanced level course, giving 7.5 ECTS credits. The main purpose of the course is to consolidate and develop the students' knowledge through a practical project. Each project contains several of the elements typical for an automatic control project, such as modelling, identification, analysis, synthesis and computer implementation. For further information, see the course home page `https://www.control.lth.se/FRTN40/` or the course syllabus (Swedish) `https://kurser.lth.se/kursplaner/17_18/FRTN40.html`. The course receives an annual participant evaluation, which can be found at `https://www.ceq.lth.se`.

Each project is completed by a student group, typically comprising three to four individuals, under the supervision of a PhD student or teacher from the department. As an alternative to projects proposed by the department, it is possible for individual groups to propose their own projects. Apart from completing the project itself, the course provides training in technical writing, through the writing of the papers compiled into this report. Participants are also provided opportunities to strengthen their oral presentation skills through two feedback seminars and a final presentation. They also serve as peer reviewers for each other's work.

During 2017, student prizes were handed out within the categories "Best control engineering" and "Best paper, documentation and presentation". The "Best control engineering" prize went to Andreas Abramsson, Emil Wåraeus, Rijad Alisic and Sififfo Sonko for *Quadcopter with Artificial Intelligence*, and the "Best paper, documentation and presentation" prize went to Henrik Fryklund, Lucas Lindén, Johan Lindqvist and Martin Sollenberg for *Balancing Suitcase*.

We would like to thank all participants and supervisors of the 2017 edition of the course for their enthusiasm and hard work leading up to this report. It is our hope and belief that the course has provided new knowledge and skills, which we hope will be of use in the future careers of the participants.

Lund, January 2017
Kristian Soltesz
Anton Cervin

Contents

Ball-Catching Robot Arm

August Bergöö*, Simon Jönsson†, Joakim Magnusson‡ and Nils Vreman§
Lund University
*tfy13abe@student.lu.se, †tfy13sjo@student.lu.se, ‡elt12jma@student.lu.se, §tfy13nvr@student.lu.se
https://gitlab.control.lth.se/regler/FRTN40/group-A

***Abstract*—Robot control with visual sensor inputs can be used in a wide variety of applications. This project aims to get a small robot arm to catch a thrown ball. The ball position is found using a colour based object detection algorithm and a computer vision system maps the position from 2D to 3D. The ball trajectory is estimated using a discrete Kalman filter, and the arm-trajectory is calculated using the FABRIK method, which gives good servo angles as long as the impact point is in reach of the robot. The object detection is fast enough for a good update rate and works good for nice looking scenes. However the ball, which is orange, is often confused with beige skin colours which gives unpredictable ball positions. The robot system has tight real-time constraints which have influenced the software design greatly. The system is written with low coupling and each module is running on separate threads, speeding up the computations considerably. The ball catching robot system will need further development in order to be functional. Right now the ball catching robot arm is not connected to the image analysis. A more distinctive coloured ball should be used and some kind of outlier rejection should be implemented in the trajectory estimation. There are also some strange behaviours in the FABRIK algorithm when it tries to set a position that is out of reach which will have to be handled.**

I. Introduction

Object catching using image analysis has applications within many different fields. Everything from automating apple picking to advanced industrial assembly line robots and real-time tracking of objects. This project was aimed at catching a ball thrown by a human using real-time robot control and image analysis. Since this project is multidisciplinary it leads to innovation and knowledge development within automatic control as well as image analysis, computer vision, mathematics, real-time programming and electronics. Fields covered in this project are amongst others object recognition using image analysis, spatial positioning using computer vision, projectile trajectory modelling using Kalman filters, real-time programming using monitors and robot control using the FABRIK method.

The idea behind catching a ball with a robot arm was inspired by a Ph.D-thesis [1] made by Magnus Linderoth.

II. Modeling

A. Ball trajectory

An important design idea behind this project is the need to estimate the ball trajectory from just a single measurement point. By doing this efficiently the time-margins for moving the robot arm and estimating the ball position might be increased (giving the robot arm something to aim for very early in the ball's actual trajectory). First of all, a mathematical model of the ball position is needed.

Given the position vector $p = [X,\ Y,\ Z]$, the position of the ball is acquired through the differential equation:

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

In (1) g is the gravitational constant, v_c is a disturbance in the position (for example caused by wind) and c is the air drag coefficient (affected by the mass and the size of the projectile).

After discretisation (assuming $\ddot{p}(t) \approx \ddot{p}(kh),\ \forall\ kh \leq t \leq (k+1)h$) and by using the state vector $x = [X,\ Y,\ Z,\ \dot{X},\ \dot{Y},\ \dot{Z}]$, (1) can be represented by the

Fig. 1: The rig we used. Robot arm and connections can be seen in the figure.

state-space seen below:

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 1 & 0 & 0 & h & 0 & 0 \\ 0 & 1 & 0 & 0 & h & 0 \\ 0 & 0 & 1 & 0 & 0 & h \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x(k) \\ &+ \begin{bmatrix} \frac{h^2}{2} \begin{bmatrix} -c\dot{X}(k)V(k) \\ -c\dot{Y}(k)V(k) - g \\ -c\dot{Z}(k)V(k) \end{bmatrix} \\ h \begin{bmatrix} -c\dot{X}(k)V(k) \\ -c\dot{Y}(k)V(k) - g \\ -c\dot{Z}(k)V(k) \end{bmatrix} \end{bmatrix} + v_c(k) \\ &= \Phi x(k) + \Gamma + v_c(k). \end{aligned} \tag{2}$$

In (2) $V = \sqrt{\dot{X}^2 + \dot{Y}^2 + \dot{Z}^2}$ is the speed of the ball and the Γ matrix can be seen as some sort of time-dependent part of the ball trajectory estimation where the states depend on the air-drag and gravitation. Note that $\dot{X}(k)$, $\dot{Y}(k)$ and $\dot{Z}(k)$ are the three last states in the state vector $x(k)$.

Using (2), a Kalman filter can be developed to estimate the next position in each step, $x(k+1)$. This does not give the entire trajectory. However, when a new ball position is acquired (from the computer vision) and the velocity for this position has been calculated, the next ball position is estimated using the Kalman filter. From this state estimation we then estimate the next ball position and so on and so forth until we reach a point that lies as close to the robot arms neutral position as possible (the impact point).

The Kalman filter is explained in more detail in Section IV-B.

III. Electro-Mechanics

The mechanical parts of the robot arm were bought as a kit and assembled by hand. Six servos were used, four heavier ones (Towerpro MG996R) to control the major joints and two lighter ones (Turnigy MG90S) to control the smaller joints. To send PWM signals to the servo motors an Arduino Uno R3 was used with an Arduino servo shield. The servo shield was powered by a 5V DC adapter with a maximum current of 7.5 A. A 12V DC adapter with a maximum current of 1 A powered the Arduino. Two PS3 EYE cameras were used for the image analysis and the computer vision. Since the cameras needed to be kept at a fixed distance from the robot arm and from each other, a wooden frame was built as well. All of this can be seen in Figure 1 (excluding the cameras that were used for testing during the photo).

IV. Control

A. Real-time implementation and Monitor

Since the entire project consists of four major tasks (ball trajectory estimation, image analysis, computer vision and robot-arm trajectory generation) it is essential that there is a solid core structure which will execute the tasks in parallel. Since our code is implemented in Java the tasks are very easy to parallelize using the **synchronized** keyword.

It is of utmost importance that our variables are mutually exclusive, since some of them are shared between threads (e.g the ball position is shared between the computer vision part and the ball trajectory estimator). The way this is solved is by introducing a monitor to keep the common variables locked. Another aspect we introduced to our code was an Observer-Observable connection on the monitor. The way this works is that we make the monitor class extend an Observable class and the classes that will use the monitor are implementing the Observer interface. When something in the monitor happens it notifies its observers that it has changed. This, in turn, triggers a specific method in the observer threads to wake up and collect the desired variables. This way the threads that depend on the monitor will not have to wait for something to happen since when it happens they will be notified.

B. Kalman filter

The Kalman filter is based on basic control theory:

$$\begin{aligned} \hat{x}(k+1) &= \Phi\hat{x}(k) + \Gamma + K\left(y(k) - C\hat{x}(k)\right) \\ &= \left(\Phi - KC\right)\hat{x}(k) + \Gamma + Ky(k). \end{aligned} \tag{3}$$

The discretized Kalman filter matrix, K, is found using MATLAB's built-in function **dlqe.m**. For this function to calculate K we need Φ, C and three matrices Q, R and G. The Q matrix is a weight matrix which says how much trust is put into the model in comparison to the measurements. The R matrix is in some sense the complete opposite. It puts weights on how much trust the measurements gets in comparison to the model. Finally the G matrix describes how the load disturbance is distributed amongst the states (since the disturbances are assumed to be independent in each direction, $G = I$).

By using (3) in a conditional loop, the optimal impact point can be found. With the ball positioned in $p = (X, Y, Z)$ and initial velocity $v = (\dot{X}, \dot{Y}, \dot{Z})$ the next position is estimated as (in accordance with (2)) $\hat{x}(1) = \Phi\hat{x}(0) + \Gamma(0) + v_c(0)$, where $\hat{x}(0) = [X(0), Y(0), Z(0), \dot{X}(0), \dot{Y}(0), \dot{Z}(0)]$ is the initial state vector created from the input position and velocity. In the next loop iteration $\hat{x}(2)$ is estimated from $\hat{x}(1)$. If the distance from the estimation $\hat{x}(2)$ to the robot arm is smaller than that of $\hat{x}(1)$ to the robot arm, the loop continues. If this is false $\hat{x}(1)$ is returned as the optimal impact point since it will have had the minimum distance to the robot arm.

Note that the algorithm above finds the optimal impact point for any choice of $\hat{x}(k)$ and $\hat{x}(k+1)$.

C. Colour based object detection

In this project, colour based object detection for finding the position of the ball is used. The problem with colour based detection is that the perceived colour of an object may vary a lot in different lightning conditions, e.g. sunlight or luminescent light. This makes simple colour detection techniques, such as simple thresholding in the RGB-space, sensitive to outer conditions. A more robust technique for colour based

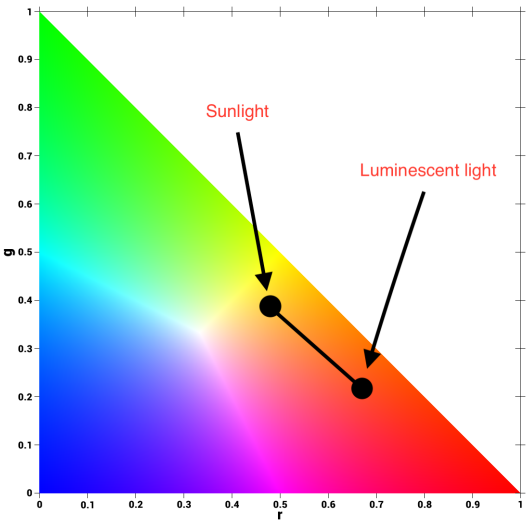


Fig. 2: The normalized RGB-space represented by the red and green channels.

object detection was suggested by Magnus Linderoth in his Ph.D-thesis [1], which will be used here.

A colour can be described by its chromaticity and luminance, where the chromaticity represents the "actual colour" and the luminance describes the intensity of the colour. The luminance is not used in the detector, which only uses the chromaticity which makes it "invariant" to light intensity (the colour channels might still be saturated or dominated by noise). The chromaticity can be represented by two channels in the normalized RGB-space $\{r, g, b : r+g+b = 1,\ r, g, b \geq 0\}$. The detector described here uses the red and green channels. The chromaticity is described by:

$$r = \frac{R}{R+G+B}, \qquad g = \frac{G}{R+G+B}. \tag{4}$$

How r and g in (4) map to actual colours can be seen in Figure 2. Let an object illuminated by sunlight have the colour (R_1, G_1, B_1) with chromaticity (r_1, g_1), and let the same object illuminated by luminescent light have the colour (R_2, G_2, B_2) with chromaticity (r_2, g_2). If the two light sources illuminate the object simultaneously, the resulting colour will be:

$$(R_3, G_3, B_3) = \lambda_1(R_1, G_1, B_1) + \lambda_2(R_2, G_2, B_2), \tag{5}$$

where $\lambda_1, \lambda_2 \geq 0$. The corresponding chromaticity for the colour in (5) will be:

$$(r_3, g_3) = \theta_1 \cdot (r_1, g_1) + \theta_2 \cdot (r_2, g_2), \tag{6}$$

where $\theta_i = \frac{\lambda_i(R_i+G_i+B_i)}{R_3+G_3+B_3}$. It can be shown that $\theta_1, \theta_2 \geq 0$ and $\theta_1 + \theta_2 = 1$, which implies that (r_3, g_3) is a convex combination of (r_1, g_1) and (r_2, g_2), which means that (r_3, g_3) will be somewhere on the straight line between (r_1, g_1) and (r_2, g_2) in the space seen in Figure 2. This property is the back bone of the colour based object detector.

The method for object detection can be divided into a number of steps. The following steps are performed offline:

1) A lot of images of the ball illuminated by two different light sources (sun and luminescent) are collected and the mean image is formed for the different light sources. This is to reduce the effect of noise in the training data.
2) The sensor bias is subtracted from the mean images. Sensor bias is the recorded intensity in completely darkness and may vary for different colour channels. Therefore it can introduce dependencies between the luminance and chromaticity, which is why it has to be subtracted.
3) A histogram over the chromaticities for the pixels in the training images that belong to the ball are created. This will create two peaks in the histogram for the different light sources. The bins in the histogram that lie between the peaks are increased according to the principle described by (6). If the histogram is normalized, it can be interpreted as the probability density for the chromaticity of the ball.
4) The "probability density" of the background is chosen. This is a tuning parameter which has to be found experimentally.
5) For each colour in the table, the probability of belonging to the foreground, P_{fg}, is calculated.
6) A look-up table with RGB values as its input is created. The output is $P_{fg} - 0.5$ (denoted as probability score) which means that a positive output implies $P_{fg} > P_{bg}$ and vice versa. The elements in the table are shifted according to the sensor bias, e.g. a sensor bias of $(1, 1, 1)$ and RGB-entry $(3, 3, 3)$ will get the probability for the "true" RGB-value $(2, 2, 2)$. This look-up table is stored for online use.

The probability mentioned in point 5 is calculated as:

$$P_{fg} = \frac{p_{fg}}{p_{fg} + p_{bg}}, \tag{7}$$

where p_{bg} is the tuning parameter from point 4. Note that since p_{bg} is chosen, not calculated, (7) is not a "real" probability. p_{bg} can be thought of as a threshold on how large the probability density should be before the algorithm decides that a pixel belongs to a ball. The steps taken online will be:

1) An image of the scene is taken and for each pixel, the probability score is retrieved from the stored look-up table. A "probability image" is created as the result.
2) The square that maximizes the sum of the scores enclosed by the square is calculated. The centre of this square will be interpreted as the position of the ball.

As mentioned in point 2, the ball is approximated by a square. Point 2 can be implemented efficiently with a method called *integral images* which is a lot easier if we look for squares compared to circles. The centre of the obtained squares also seems to approximate the true ball centre fairly good, as can be seen in Figure 15.

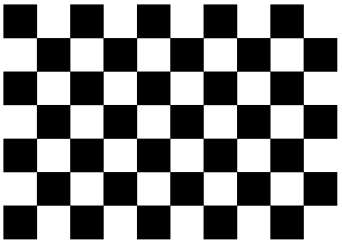


Fig. 3: The pattern used to calibrate a camera's intrinsic and extrinsic parameters.

D. Computer vision

From the image analysis one will achieve two images with the 2D-coordinates of the ball on each image. Using computer vision combined with parameters defined by the cameras, one can calculate the 3D-coordinates of the ball. There are two different types of parameters that are of interest, extrinsic and intrinsic parameters. Extrinsic parameters are the orientation and position of the camera in space and are not unique for a specific camera. Intrinsic parameters on the other hand, are the fundamental properties of the camera and they describe the camera's focal length, skew and aspect ratios. Both the extrinsic and intrinsic parameters can be calculated through a calibration process by using a calibration pattern, see Figure 3.

The calibration process consists of three steps; calibrating the first camera's intrinsic parameters, calibrating the second camera's intrinsic parameters and then calibrating the extrinsic parameters for both cameras simultaneously. All three steps use a function that detects chessboard corners and therefore one needs a printout of the calibration pattern seen above, see Figure 3. In order to get a decent calibration of the cameras, a large number of images (more than 50) have to be taken of the calibration pattern from multiple angles and distances by the cameras. The calibration is performed offline, which means that no real time constraints apply, and the parameters are saved for future use.

The extrinsic parameters are simply described by $\begin{bmatrix} R & t \end{bmatrix}$, where R is a rotation matrix and t is a translation vector. The intrinsic parameters, however, is achieved by forming a matrix K, as seen below:

$$K = \begin{bmatrix} f & sf & x_0 \\ 0 & \gamma f & y_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{8}$$

f is the camera's focal length, γ is the aspect ratio, (x_0, y_0) is the principal point and s is the skew. The extrinsic and intrinsic parameters can be stored in a camera matrix P:

$$P = K \begin{bmatrix} R & t \end{bmatrix}. \tag{9}$$

Having the two 2D-coordinates of the ball and the two camera matrices one can calculate the 3D-coordinates. If u_1 and u_2 define the 2D-coordinates of the ball then these coordinates can be written in homogeneous coordinates:

$$\begin{aligned} u_1 = (x_1, y_1) \rightarrow u_1 = (x_1, y_1, 1) \\ u_2 = (x_2, y_2) \rightarrow u_2 = (x_2, y_2, 1) \end{aligned}. \tag{10}$$

According to computer vision theory, these can be transformed to the following:

$$\begin{aligned} u_{1x} = \begin{bmatrix} 0 & 1 & -y_1 \\ -1 & 0 & x_1 \end{bmatrix} \\ u_{2x} = \begin{bmatrix} 0 & 1 & -y_2 \\ -1 & 0 & x_2 \end{bmatrix} \end{aligned}. \tag{11}$$

These coordinates can then be used to get the ball's 3D-coordinates in homogeneous coordinates U:

$$\begin{aligned} u_{1x} P_1 U = 0 \\ u_{2x} P_2 U = 0 \\ U = (x, y, z, 1) \end{aligned}. \tag{12}$$

From homogeneous coordinates and the camera matrices a matrix M can be constructed:

$$M = \begin{bmatrix} u_{1x} P_1 \\ u_{2x} P_2 \end{bmatrix}. \tag{13}$$

The null space of M can be calculated using singular value decomposition (SVD) and one could easily achieve the 3D-coordinates of the ball, since one of matrices resulting from the SVD contains the information of the homogeneous 3D-coordinates U. Both the image analysis and the computer vision were implemented using OpenCV for java (version 3.2.0).

E. Arm trajectory

1) FABRIK algorithm in 2D: FABRIK stands for forward and backwards reaching inverse kinematics. The algorithm consists of two parts, each consisting of a number of steps which now will be explained. The following steps just happen in the code and not on the actual robot. The picture illustrates a robot arm in 3D, with three segments connected with joints. The steps in the first part of the algorithm are illustrated in Figure4 to Figure9.

Figure 4 shows the robot arm in a random position and the red cross is the goal and the desired position for the end of the arm to reach. The first step in the first part of the algorithm is shown in Figures 5 and 6.

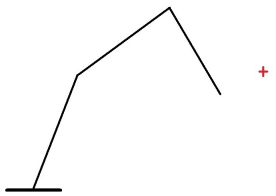


Fig. 4: Robot arm illustrated in 2D and a red cross which is the position to reach.

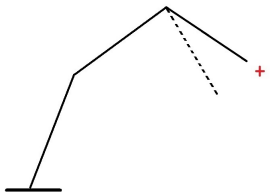


Fig. 5: The end segment has been rotated towards the goal.

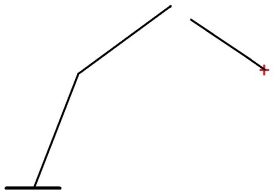


Fig. 6: The end segment has been moved so that it is positioned at the goal.

The next step is the same as the first, but this time the middle segment will be rotated towards the base of the end segment. This is illustrated in Figures 7 and 8.

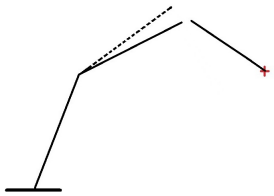


Fig. 7: The middle segment has been rotated towards the base of the end segment.

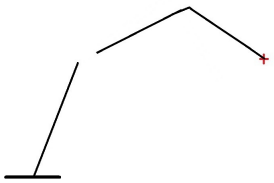


Fig. 8: The middle segment has been moved so that its end is positioned at the base of the end segment.

The last step in the first part of the algorithm can be seen in Figure 9. This time the segment is just rotated but not moved to another position as in the other steps.

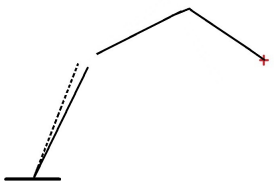


Fig. 9: The base segment has been rotated towards the base of the middle segment.

The first step in the second part of the algorithm is to rotate and move the middle segment back so it is connected with the base segment again. This is illustrated in Figures10 and 11

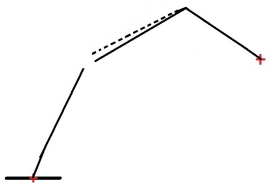


Fig. 10: The middle segment has been rotated towards the end of the base segment.

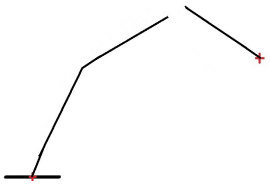


Fig. 11: The middle segment has been moved so that its base is positioned at the end of the base segment.

Last the end segment is connected with the middle segment again done by moving it as in Figures12 and 13

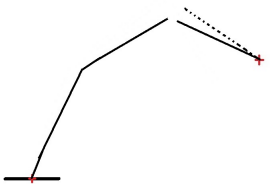


Fig. 12: The last segment has been rotated towards the end of the middle segment.

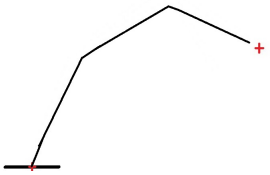


Fig. 13: The last segment has been moved so that its base is positioned at the end of the middle segment.

This is all calculated with vectors and the actual robot is not moved like in each of these steps. After part one and two of the algorithm is done the angles are sent to the robot and it is positioned like in Figure13. As can be seen in Figure13 the end segment has not reached the goal after one iteration, but it has moved closer. Iterating these steps and the end segment will eventually reach the goal. Since the computations are not very heavy and it does not take to many iterations to come close to the goal, this is done relatively fast in comparison with the rest of the code.

2) FABRIK in this project: The robot arm for this project and how the joints can be rotated can be seen in Figure 14. To simplify we will not change the joints making the rotations crossed out in Figure 14, they will be kept constant.

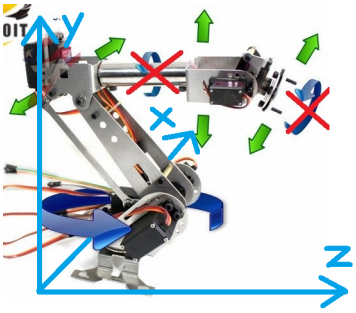


Fig. 14: Movements not used are crossed out.

Given the goal position (x,y,z) and rotate this around the y-axis to the yz-plane we only need to solve a 2D problem and thus the FABRIK algorithm can be implemented easier leaving out the angle of the base joint of the robot arm. The base joint angle is instead given by $\theta = tan^{-1}(x/z)$. This 2D algorithm was first implemented and simulated yielding promising, stable behaviour. In our project the computation time for the FABRIK algorithm is not a bottle neck because of the limited speed of the servos controlling the robot. The FABRIK method can be implemented taking account of angle constraints. This needs to be researched further and we will try using FABRIK without constraints in the meantime.

F. Controlling the servos

In order to send PWM signals to the servos an Arduino with a servo shield is used. The Arduino receives the desired angles for the servos through a USB cable from the PC running the main code. These angles are then mapped using linear regression for some known PWM signals and corresponding angles.

V. Results

A. Ball trajectory and Kalman filter

When constructing the Kalman filter, the following matrices were chosen as the covariance matrices needed to compute the Kalman gain matrix, K (as described in Section IV-B).

- G was chosen as the identity matrix, $I_{[6x6]}$.
- Q was chosen as the identity matrix, $I_{[6x6]}$.
- R was chosen as the identity matrix, $I_{[3x3]}$.

B. Robot-arm trajectory

The method used to measure the position of the robot arm end point was not accurate since no sensors existed on the servos, but an accuracy on the position could be estimated up to +-20mm from the actual position. If the position of the robot arm was set to a position out of reach, an oscillating motion occurred.

C. Image analysis

As can be seen in Figure 15, the object detection works well on the test image. Even though the ball is approximated by a square, the centre of the square almost coincides with the real centre of the ball.

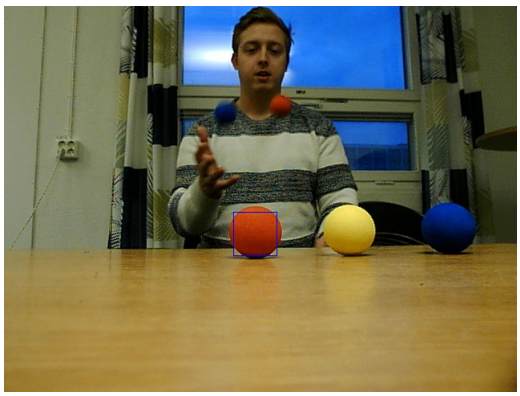


Fig. 15: The square that maximizes the sum of enclosed probability scores can be seen around the nearest red ball. The image is taken in a mixture of sunlight and luminescent light.

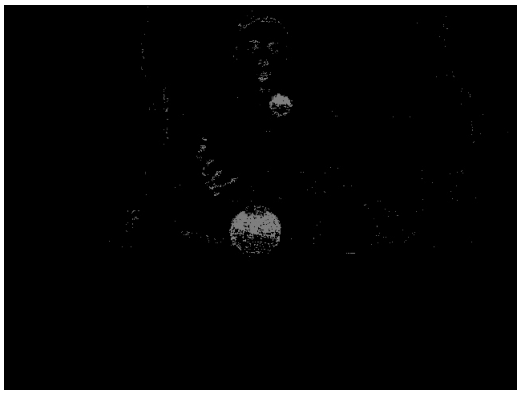


Fig. 16: The pixels from Figure 15 where $P_{fg} > 0.5$ are seen in gray.

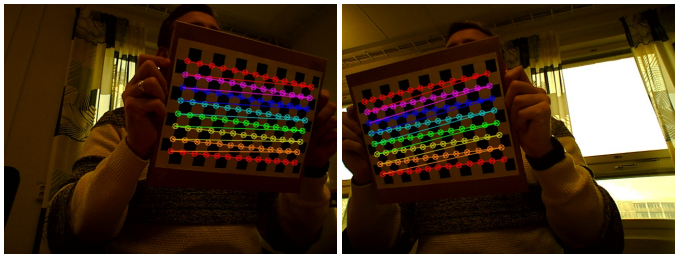
Figure 16 shows where in Figure 15 $P_{fg} > 0.5$, i.e. the pixels that are classified as the ball.

As can be seen in Figure 16, the probability is high for the two red balls in the image and there are also some noise around the hand, lips and eyebrows of the person in the image. The algorithm managed to classify the largest red ball as the most probable, and the approximated center (of the square) coincides well with the real center of the ball. However, the noise in Figure 16 is problematic when the ball is far away, and the noise has as large areas in Figure 16 as the true ball. Experiments showed that the algorithm found the ball in a majority of the images up to distances around three meters. At further distances, the noise dominates the probability image.

The PS3 EYE cameras take images with the resolution 640x480 and if they are scaled down to 160x120, the ball finding algorithm takes approximately 20 ms on average on a 2.5 GHz processor (the first couple of times takes almost the double due to cache misses and such).

D. Computer vision

Using the calibration method described earlier (see Section IV-D, page 4), the result on two sample images can be seen



(a) The view from one of the two cameras. (b) The view from the other camera.

Fig. 17: Detecting the calibration pattern with both cameras.

in Figures 17a and 17b.

The calibration pattern is recognized in both images by the software and the points are marked in the image. The software used these points, matched them accordingly with each other between the two images and calculated the cameras' parameters.

VI. Discussion

A. Kalman filter

All of our matrices were chosen as the identity matrix due to the fact that we were never able to test the Kalman estimator on the real system. However, since we needed to test the Kalman estimator the identity matrices were the initial, and final, choice.

B. Real-time programming

A monitor was used to keep the commonly used variables mutually exclusive. Since we also added Observer-Observable behaviour we might have done things twice. We wanted to keep the threads as independent as possible and therefore only notify them when something happened in the monitor but doing so we might have introduced unnecessary run time. When a thread calls a function in the monitor (which will notify its observers) it gets paused until the observers have completed their *update* functions. This might introduce some extra wait time for important threads if the *update* functions are computationally heavy. We went around this problem by making the *update* functions very nimble (finishing in under 1 ms) but it is important to notice that this might introduce problems if you are not aware of this fact. This problem might be solved by using *await* and *notifyAll* functions (that all classes in Java share) alongside conditional variables.

C. Robot arm

One of the problems we faced with the robot arm was the connection from the Arduino to different computers. Since most of our code (e.g the image analysis library OpenCV) was preferably run on a Linux based operating system we needed to configure the Arduino to connect with a Linux computer. Problems arose as soon as the two devices were connected as it seemed as the Linux computer sent signals to the Arduino (maybe asking for a virtual handshake) making the Arduino freak out. This was solved by resetting the Arduino every time the devices were connected and then running the code with root access (giving the serial connection root access as well).

As mentioned in V-B the accuracy of the robot arm is approximately +-20mm which we definitely think can be improved. The angles calculated by the FABRIK algorithm are mapped to PWM signals. Doing this mapping includes measuring the angles manually with a contractor and can definitely be a source of error.

The strange behaviour described in V-B when trying to set the robot arm to a position out of reach should be possible to prevent. It is probably an error in the code implementing the FABRIK algorithm even though we have not been able to localise the error. We believe this because this error did not occur when simulating the robot arm in 2D, mentioned in IV-E2. After the simulations were done the code was improved and a the handling of a third coordinate was added which may have introduced the error.

D. Image analysis

As mentioned in Section V-C, finding the ball in an image worked well when the ball was reasonable large in the image. The algorithm failed to find the ball at large distances, i.e. when the ball is depicted in few pixels in the image. There will always be some noise in probability image (Figure 16), and the probability that these pixels will be classified as the ball increases as the true ball gets smaller and smaller in the image. A solution to this may be to increase the resolution of the images, but this will also increase the computation times considerably. Black background colours are also a major problem. Black is a non-colour and is the result of low light intensity, which means that black objects can have "true" colours anywhere in Figure 2. The RGB-value $(1, 1, 1)$ with a maximum possible value of 255 will be perceived as black by our eyes, but will produce white colour in the classifier. The classifier is trained on a red ball, and as can be seen in Figure 16, skin tone and specifically lips are sometimes confused with the ball, which produces noise in the probability image. A better choice of ball colour would for example be a distinctive green colour, which is unusual in indoor environments (which is why green screens are used).

Slow computation times are a known issue for computer vision. Images are usually fairly large and looping through them takes a lot of time. Many computer vision applications also have real-time constraints which enforce a trade-off of computation time and precision. The object detection algorithm proved to be quite fast though, with execution times of tens of milliseconds for images with approximately 20,000 pixels (scales as $\mathcal{O}(n)$ with increased number of pixels). Every image is only looped through twice; the first is to translate the pixel values to probability scores and build an integral image, and the second is to search for the maximizing square in the integral image. This is to be compared with, for example, feeding the images through a deep convolutional neural network which often makes multiple convolutions (each requiring one image loop) at each layer.

E. Computer vision

The computer vision faced a lot of difficulties in this project. One of the biggest difficulties was the requirement of high accuracy on the two 2D-coordinates of the ball, which were acquired through colour based object detection from two images taken by the cameras. These 2D-coordinates had a tendency to be fluctuating from one image to another, which made it problematic to calculate reliable 3D-coordinates of the ball since the fluctuations got magnified in the conversion process. However, when the detection of the ball was functioning it seemed as if the 3D-coordinates had the proper behaviour, but it would still require a mapping or translation of the cameras' coordinate system to the robot's coordinate system. This was never done and therefore it is difficult to know if these coordinate systems would be difficult to align and if there were any other major issues with the computer vision.

Other difficulties were figuring out how the calibration process works in OpenCV, especially since OpenCV is not written in Java and the data types, classes and methods could be completely different compared to the original ones used in C++. In addition, the documentation for OpenCV in Java was not well documented and since some methods could require over thirteen parameters of different types, it is difficult to know if all of the inputs and outputs were done properly. However, since the behaviour seems reasonable when the detection of the ball is working, it is fair to assume that the calibration was working to some extent.

References

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

Self-Balancing Lego Robot

Mattias Josefsson*, Jonathan Persson†, Alexander Skafte‡ and Jakob Warlin§
Supervisor: Tommi Nylander
Lund University
*tpi13mjo@student.lu.se, †ast13jpe@student.lu.se, ‡tfy13ask@student.lu.se,
§tfy13jwa@student.lu.se, https://gitlab.control.lth.se/regler/FRTN40/2017/group-B

***Abstract*—The aim of this project was to design and implement different control strategies for balancing a robot on two wheels, and to investigate how measurement noise affects the performance. The problem can be formulated as balancing an inverted pendulum, which is a standard nonlinear control problem. PID and LQR controllers were implemented and compared to each other. In order to measure the angle of the robot a gyroscope and an accelerometer were used, and measurement noise was filtered using two different methods, which we compared: Kalman filtering and complementary filtering. The control algorithms were implemented first in Simulink and then in leJOS on a LEGO Mindstorms EV3 microprocessor. Our findings suggest that a LQR manages to balance the robot better than a PID controller does. As for filtering, our findings suggest that a Kalman filter is the best filter to use in order to reduce the drift from the gyroscope and the noise from the accelerometer.**

I. Introduction

This project aimed at balancing a two-wheeled LEGO-structure, shown in figure 1, similar to a Segway by keeping it in the upright position. The control problem is that of an inverted pendulum. Different control strategies for keeping the robot upright and steering it have been implemented and tested. Three common control methods have been compared: optimal control through a *linear-quadratic regulator* (LQR), control through *feedback linearisation* (FBL), and *proportional-integral-derivative* control (PID).

II. Equipment and materials

The robot was built using LEGO Mindstorms. The main components of the robot are two wheels, a LEGO EV3 processor, two NXT motors, a gyroscope and an accelerometer. Other LEGO pieces were used to construct the frame. The regulator for the robot was implemented in Java using the Eclipse IDE. The program was compiled on a computer connected to the robot with an USB-cable. The robot structure is partly based on blueprints found online [4].

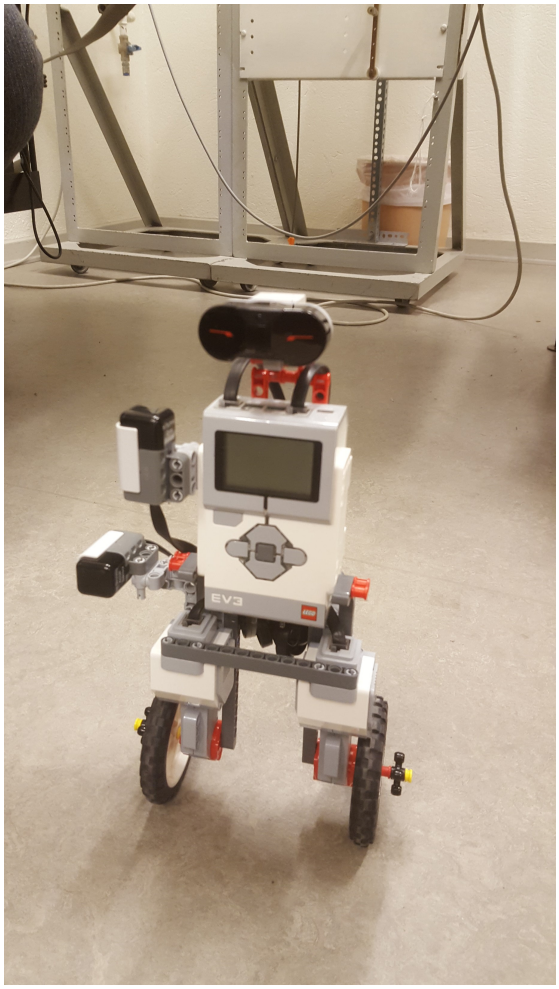


Fig. 1. Lego model

III. Theory

In this section theoretical backgrounds for the different control theories are described.

A. Complementary filter

When dealing with multiple sensors with various flaws, a complementary filter can be used to minimize these flaws. The idea of a complementary filter is to combine filtered signals from different sensors in such a way that the desirable properties of each sensor are utilized. An example is to combine an accelerometer and a gyroscope to measure an angle: a gyroscope typically is inaccurate low frequencies, which causes measurement drift, and an accelerometer typically has undesirable high frequency components, which causes measurement noise. By numerically integrating the gyroscope's signal and passing it through a high-pass filter and combining this signal with a low-pass filtered accelerometer signal, an estimate of the angle can be retrieved. See figure 2. This can easily be implemented using the update formula in equation

1.

$$angle = a \cdot (angle + gyro \cdot dt) + (1 - a) \cdot accel \quad (1)$$

where

$$a = \frac{\tau}{\tau + dt}$$

where τ is the desired time constant and $dt = \frac{1}{fs}$ where fs is the sampling frequency.

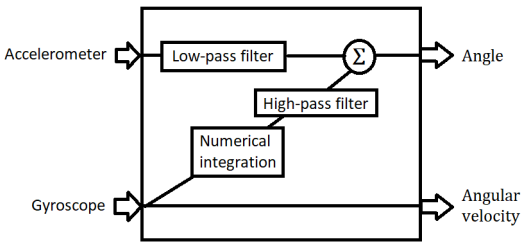


Fig. 2. Complementary filter

B. Kalman filter

A Kalman filter is an algorithm that can be used in order to estimate unknown variables and reduce noise from measurements. For a discrete-time state-space model on the form

$$x_{k+1} = Ax_k + Bu_k + v_k \quad (2)$$
$$y_k = Cx_k + e_k \quad (3)$$

where $E\{v_k v_j^T\} = Q_v \delta_{kj}$, $E\{e_k e_j^T\} = Q_e \delta_{kj}$, and $E\{v_k e_j^T\} = Q_{ve}\delta_{kj}$. The Kalman filter is constructed according to equations 4-8: [2]

$$\hat{y}_k = C\hat{x}_k \quad (4)$$
$$R_k = Q_e + CP_kC^T \quad (5)$$
$$P_{k+1} = AP_kA^T + Q_v - K_kR_kK_k^T \quad (6)$$
$$K_k = (AP_kC^T + Q_{ve})(Q_e + CP_kC^T)^{-1} \quad (7)$$
$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + K_k(y_k - \hat{y}_k) \quad (8)$$

In the case of estimating the angle with measurements from an accelerometer and a gyroscope, a simpler Kalman filter can be implemented according to the following equations [3]:

$$\begin{aligned}
\hat{\mathbf{x}}_{k|k-1} &= \mathbf{F}\mathbf{x}_{k-1\hat{|}k-1} + \mathbf{B}\dot{\theta}_k \\
\mathbf{P}_{k|k-1} &= \mathbf{F}\mathbf{P}_{k|k-1}\mathbf{F}^T + \mathbf{Q}_k \\
\hat{\mathbf{y}}_k &= \mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1} \\
\mathbf{S}_k &= \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R} \\
\mathbf{K}_t &= \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1} \\
\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\hat{\mathbf{y}}_k \\
\mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_{k|k-1}
\end{aligned} \quad (9)$$

With

$$\hat{\mathbf{x}}_k = \begin{bmatrix}\theta \\ \dot{\theta}_b\end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix}1 & -h \\ 0 & 1\end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix}h \\ 0\end{bmatrix}$$
$$\mathbf{Q}_k = \begin{bmatrix}Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b}\end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix}1 & 0\end{bmatrix}, \quad \mathbf{R} = var(e_k) \quad (10)$$

where θ is the angle, $\dot{\theta}_b$ the bias of the measurements from the gyro, h the sample time, Q is process noise, and R measurement noise. The noises are the parameters that can be tuned. The initial P-matrix can be set to a 2x2 matrix with zeros.

C. Linear-quadratic regulator

The LQR was used as the primary regulator. It is concerned with operating a dynamic system at minimum cost and can be described by a set of equations:

$$u(t) = -L\hat{x}(t) \quad (11)$$
$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + K(y(t) - C\hat{x}(t)) \quad (12)$$
$$\begin{aligned}0 = {} & AP + PA^T + NR_1N^T \\ & - (PC^T + NR_{12})R_2^{-1}(PC^T + NR_{12})^T\end{aligned} \quad (13)$$
$$K = (PC^T + NR_{12})R_2^{-1} \quad (14)$$
$$L = Q_2^{-1}B^TS \quad (15)$$
$$0 = A^TS + SA + M^TQ_1M - SBQ_2^{-1}B^TS \quad (16)$$

The equations eventually yield a state feedback controller.

D. PID control

A PID controller finds the difference between the desired reference signal and the actual system output signal and thereafter calculates a control signal. Denoting the error as $e(t)$ the PID-controller on standard form can be written as:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(t)\,\mathrm{d}t + T_d\frac{\mathrm{d}e(t)}{\mathrm{d}t}\right)$$

or, as a transfer function in the Laplace-domain (s-domain)

$$G(s) = K\left(1 + \frac{1}{T_is} + T_ds\right)$$

where K, T_i and T_d are tuning parameters for the controller. Since the controller was implemented digitally a discrete-time representation was needed. Using the backward Euler method with sampling time T_s the following transfer function was acquired:

$$H(z) = K\left(1 + \frac{T_s}{T_i}\frac{1}{1 - z^{-1}} + \frac{T_d}{T_s}(1 - z^{-1})\right)$$

Letting $K_1 = K$, $K_2 = T_s/T_i$ and $K_3 = T_d/T_s$ the equation can be rearranged into:

$$\begin{aligned}H(z) &= \frac{K_1(1 - z^{-1}) + K_2 + K_3(1 - z^{-1})^2}{1 - z^{-1}} \\ &= \frac{(K_1 + K_2 + K_3) + (-K_1 - 2K_3)z^{-1} + K_3z^{-2}}{1 - z^{-1}}\end{aligned}$$

Let $K_a = K_1+K_2+K_3$, $K_b = -K_1-2K_3$ and $K_c = K_3$, yielding:

$$u[k] = u[k-1] + K_a e[k] + K_b e[k-1] + K_c e[k-2]$$

which is a causal assignment that can be implemented in computer code.

IV. Modelling

The dynamics of a Segway, which the robot was based on, can be described as follows:

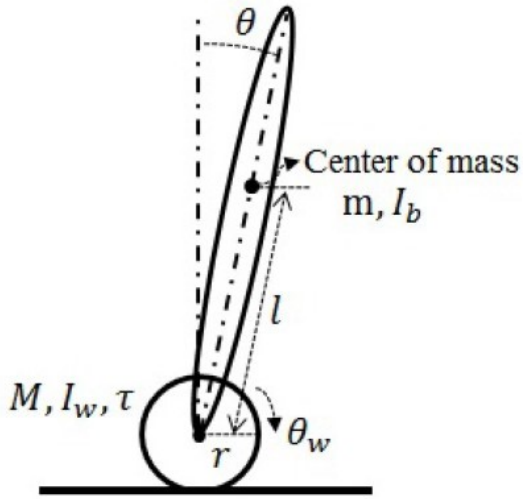


Fig. 3. Segway dynamics [1]

$$\begin{cases} m_{11}\ddot{\theta}_w + m_{12}\ddot{\theta}\cos\theta = \tau + m_{12}\dot{\theta}^2\sin\theta \\ m_{12}\ddot{\theta}_w\cos\theta + m_{22}\ddot{\theta} = -\tau + G_b\sin\theta \end{cases} \tag{17}$$

with:

$$\begin{aligned} m_{11} &= (m+M)r^2 + I_w \\ m_{12} &= mlr \\ m_{22} &= ml^2 + I_b \\ G_b &= mgl \end{aligned} \tag{18}$$

where M and m are the masses of the wheel and the body, r the radius of the wheel, l the length from the wheel to the centre of gravity, θ_w the rotational angle of the wheel, θ the angle of the body, I_w and I_b the moment of inertia for the wheel and the body respectively, and τ the applied torque on the wheel. [1]

A. First model

Introducing the states $x_1 = \theta$ and $x_2 = \dot{\theta}$ yields:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = f(x_1, x_2, \tau) \end{cases} \tag{19}$$

where:

$$f(x_1,x_2,\tau) = \frac{m_{11}G_b\sin(x_1) - m_{12}^2\cos(x_1)\sin(x_1)x_2^2 - m_{11}\tau}{m_{11}m_{22} - m_{12}^2\cos^2(x_1)} \tag{20}$$

The system in equation 19:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \frac{m_{11}G_b}{m_{11}m_{22}-m_{12}^2}x_1 - \frac{m_{11}}{m_{11}m_{22}-m_{12}^2}\tau \end{cases} \tag{21}$$

with $N = \frac{m_{11}}{m_{11}m_{22}-m_{12}^2}$ the system can be written in matrix form:

$$\begin{cases} \dot{x} = Ax + B\tau \\ y = Cx \end{cases} \tag{22}$$

with:

$$A = \begin{bmatrix} 0 & 1 \\ NG_b & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ -N \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{23}$$

B. Second model

$$\ddot{\theta}_w = \frac{(m_{12}\cos\theta + m_{22})\tau + m_{22}m_{12}\dot{\theta}^2\sin\theta - m_{12}G_b\sin\theta\cos\theta}{m_{11}m_{22} - m_{12}^2\cos^2\theta} \tag{24}$$

$$\ddot{\theta}_w = \frac{-m_{12}G_b}{m_{11}m_{22}-m_{12}^2}\theta + \frac{m_{12}+m_{22}}{m_{11}m_{22}-m_{12}^2}\tau \tag{25}$$

Using the states $\mathbf{x} = \begin{bmatrix}\theta & \theta_w & \dot{\theta} & \dot{\theta}_w\end{bmatrix}^T$ and introducing $K = m_{11}m_{22} - m_{12}^2$, the system become.

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ NG_b & 0 & 0 & 0 \\ \frac{-m_{12}G_b}{K} & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ -N \\ \frac{m_{12}+m_{22}}{K} \end{bmatrix}\tau \\ y &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x \end{aligned} \tag{26}$$

V. Control

In order to implement this on the real process, the system has to be discretized. This can be done with zero-order hold sampling. For the first model, the system becomes

$$\begin{aligned} x(t_k + h) &= \Phi_h x(t_k) + \Gamma_h u(t_k) \\ y(t_k) &= Cx(t_k) \end{aligned} \tag{27}$$

where

$$\Phi_h = \begin{bmatrix} \cosh(\sqrt{NG_b}h) & \frac{\sinh(\sqrt{NG_b}h)}{\sqrt{NG_b}} \\ \sqrt{NG_b}\sinh(\sqrt{NG_b}h) & \cosh(\sqrt{NG_b}h) \end{bmatrix} \tag{28}$$

$$\Gamma_h = \begin{bmatrix} -\frac{\cosh(\sqrt{NG_b}h)-1}{G_b} & -\frac{\sqrt{N}\sinh(\sqrt{NG_b}h)}{\sqrt{G_b}} \end{bmatrix}^T \tag{29}$$

and $h = 0.02$ is the sample time. For the second model, the matlab function *c2d* was used.

A. Control strategies

The linear systems can now be controlled using a LQR (described above) or a PID controller.

VI. Implementation

A. Simulation

In order to test the control strategies and the different filters before moving on to the real process, Simulink was used. Two different models were created, one with a complimentary filter, figure 5, and one with a Kalman filter, figure 5. Both used LQR to control the system.

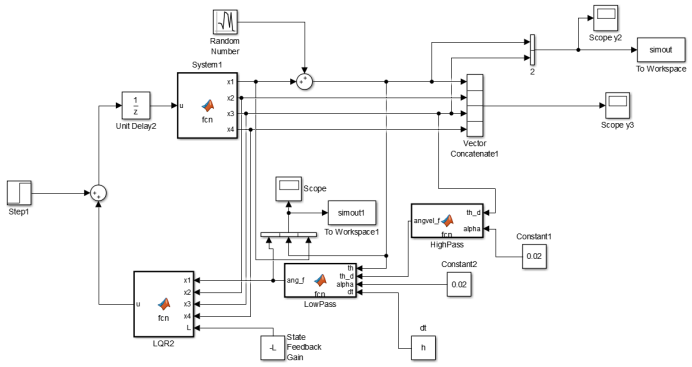


Fig. 4. Simulink model of the linearised and discrete system (with 4 states) with a LQ controller and a complementary filter.

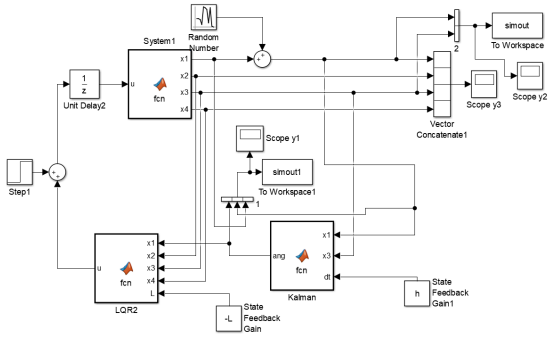


Fig. 5. Simulink model of the linearised and discrete system (with 4 states) with a LQ controller and a Kalman filter to estimate the angle.

In figure 4, the angle is filtered using a complementary filter with both the angle and the angular velocity as inputs. The filter is based on figure 2, where both the integration and the low-pass filter is included in the *LowPass* block. The system is controlled using a LQR. There is also a step disturbance on the control signal and noise on the measured angle to represent a bad sensor.

The model in figure 5 is very similar, but instead of a complementary filter, there is a Kalman filter described by 9 to estimate the angle.

B. Java

The program for the robot was written in Java and implemented with the operating system leJOS, which is a Java based operating system. leJOS VM supports most of the functionality from the standard library such as real-time threads, synchronization mechanisms, and it also includes some libraries for the LEGO Mindstorms hardware. In figure 6 the structure for the classes used for controlling the robot and communicating with the computer are visualized. A conceptual description of the blocks (Java classes) are described below:

- Main: Main method, starts all the required threads.
- ControlRobot: Depending on chosen control strategy, this controller regulates the dynamics of the robot.
- Wifi: Used for communication between the robot and a computer on the same LAN.
- LQR and PID classes: one class for each control strategy. Stores the variables needed for the control strategy.

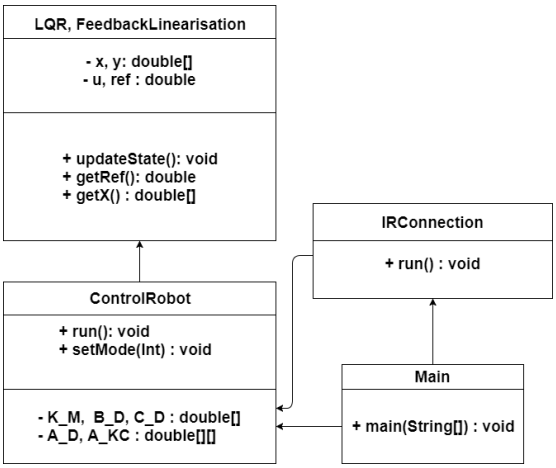


Fig. 6. UML diagram of the program structure

VII. Method

Initially, a feedback linearisation regulator was implemented but was put on hold due to not yielding any desirable simulation results. Instead an LQR controller with feedback was tried with the states:

$$x = (\theta, \dot{\theta}) \tag{30}$$

where θ is the angle of the robot body and $\dot{\theta}$ is the angular velocity of the body. However, after trial and error the robot still did not manage to balance, so another model were made with 4 states:

$$x = (\theta, \psi, \dot{\theta}, \dot{\psi}) \tag{31}$$

where the added states ψ and $\dot{\psi}$ are the wheel angle and the angular velocity of the wheel. Following a trial and error approach by simulating the system in Simulink, new model parameters were found that managed to balance the robot to a sufficient degree. With the robot now successfully balancing, using only a gyroscope to measure the angular velocity and the angle by numerical integration, and the wheels to measure the

current wheel angle and wheel angular velocity, an accelerometer was added to calculate the angle using sensor fusion. A complementary filter was implemented that mostly trusted the value from the gyroscope due to the accelerometer value being corrupted by noise. Secondly, to improve the measurements from the gyroscope and accelerometer for the angle, a Kalman filter was implemented according to 9. Simulations were done in Simulink with the different filters and then tried on the real robot. As comparison, a simple PID controller with two states was implemented as well.

VIII. Results

A. Simulink

In figures 7 and 8, the angle and the angular velocity, and the filtered angle is shown using LQR with a complementary filter. In figures 9 and 10 the same is shown but for a Kalman filter. The LQR parameters are: $\begin{bmatrix} -18.3341 & -0.0459 & -2.8688 & -0.0619 \end{bmatrix}$.

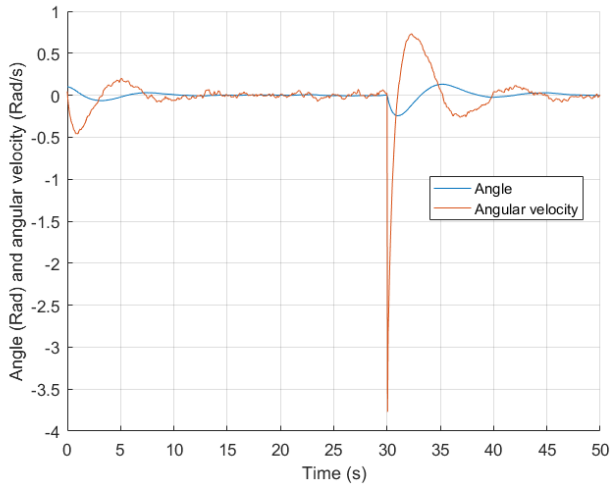


Fig. 7. Angle and angular velocity using the complementary filter. There is an initial angle of 0.1 Rad, a step disturbance with amplitude 5 at time 30 s, and noise with variance 0.001 on the measured angle.

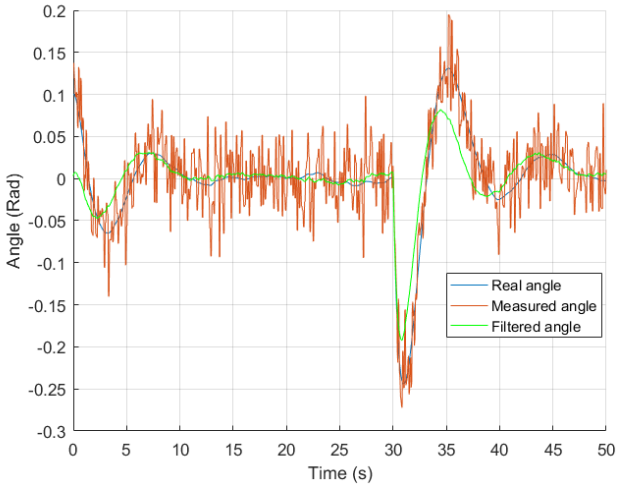


Fig. 8. Real angle, measured angle and filtered angel with complementary filter. There is an initial angle of 0.1 Rad, a step disturbance with amplitude 5 at time 30 s, and noise with variance 0.001 on the measured angle.

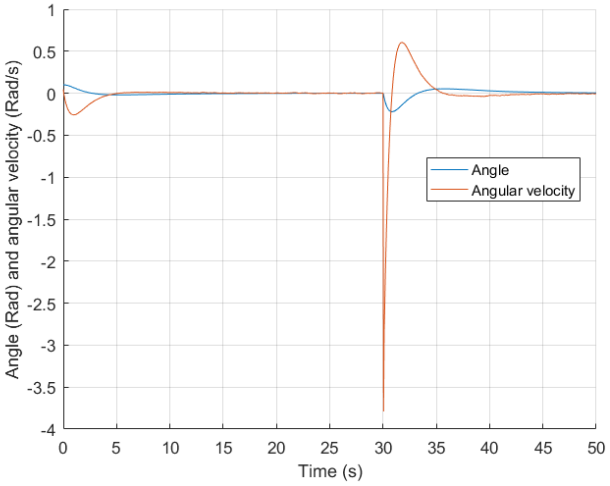


Fig. 9. Angle and angular velocity using the complementary filter. There is an initial angle of 0.1 Rad, a step disturbance with amplitude 5 at time 30 s, and noise with variance 0.001 on the measured angle.

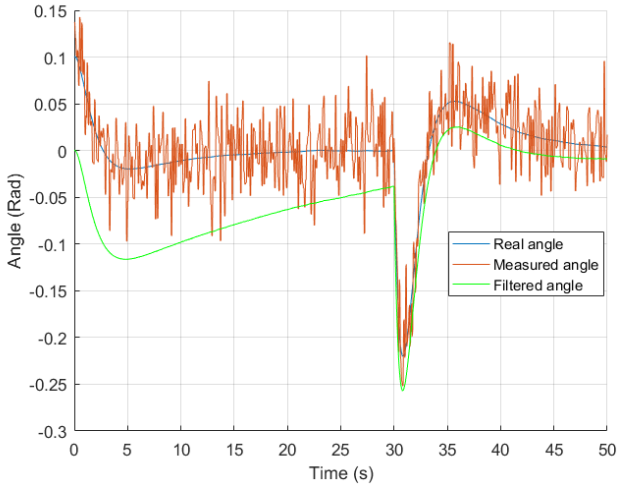


Fig. 10. Real angle, measured angle and filtered angel with Kalman filter. There is an initial angle of 0.1 Rad, a step disturbance with amplitude 5 at time 30 s, and noise with variance 0.001 on the measured angle.

B. Real system

In figure 2 the estimated angle from complementary filter, gyroscope and accelerometer is shown on the actual robot with an external force added to measure the performance of the system. Figure 12 the estimated angle from Kalman filter, gyroscope and accelerometer is shown on the actual robot with an external force added to measure the performance of the system. In table I the measured constants of the robot is shown, table III show our LQR constants for the robot and in table II the Kalman constants are shown.

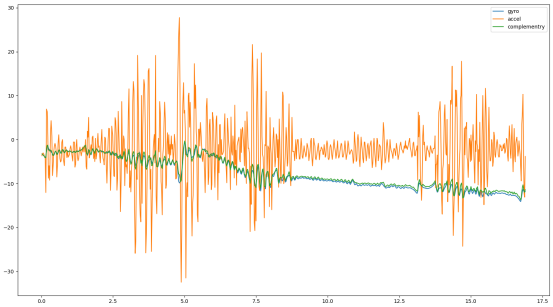


Fig. 11. LQR: Estimated angle from the Complementary filter, Gyroscope and accelerometer at different times. An external force was introduced to the system after 5 seconds

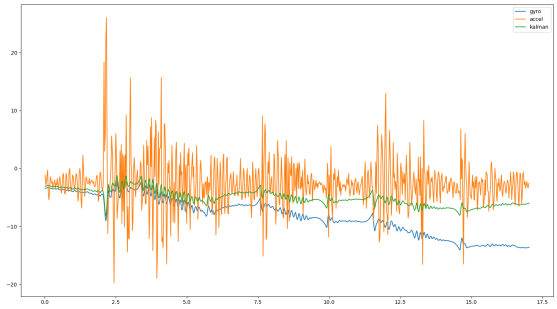


Fig. 12. LQR: Estimated angle from the Kalman filter, Gyroscope and accelerometer at different times. An external force was introduced to the system at around 2.5 seconds

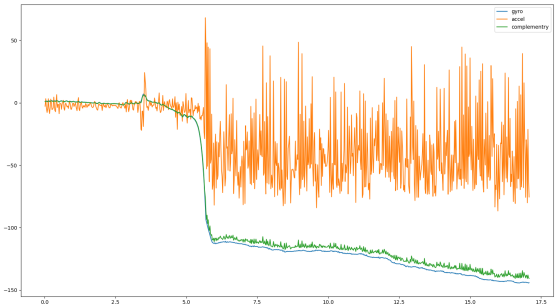


Fig. 13. PID: Estimated angle from the complementary filter, Gyroscope and accelerometer at different times. An external force was introduced to the system at around 2.5 seconds

TABLE I
MEASURED CONSTANTS

m	$0.554kg$
M	$0.060kg$
r	$0.04m$
l	$0.11m$
g	$9.82m/s^2$
I_w	$9.6 \cdot 10^{-5}kg \cdot m^2$
I_b	$0.0067kg \cdot m^2$

TABLE II
KALMAN VALUES

R_v	43
Q_{accel}	0.03
Q_{gyro}	0.00000002

TABLE III
LQR-VALUES

k_ϕ	18.1696
k_ψ	0.08
$k_{\dot{\phi}}$	0.85
$k_{\dot{\psi}}$	0.083

TABLE IV
PID VALUES

K_p	19
K_i	1.82
K_d	24.5

IX. Discussion

Based on figure 11, the complementary filter starts to drift away from the actual value. This was expected since the filter is based mostly on the values from the gyroscope due to the variance of the accelerometer values, which can be seen in equation 1. By using our LQR controller with the complementary filter the robot manage to self balance. However, due to the drift in the motors, the position of the robot gets further away from the starting point. Using the Kalman filter, this drift was removed, which can be seen in figure 12. The values for the Kalman filter that were used on the robot are shown in table II. A high value of the measurement noise was chosen since the accelerometer angle tend to be corrupted by noise and the angle from the gyroscope drifts. The Q values show how much we trust the sensor outputs (see section III-B for more details) compared to the other, where a high value indicates that the sensor was trusted less compared to the other sensor. Based on the result from II, the accelerometer was trusted less than the gyroscope which worked for the system. The values from III are the result from trial and error to calculate the Q matrix in section III-C which balanced the robot. With the PID implementation the robot was successfully successfully balanced. However, if external force was added on the robot it was not successful in reducing the disturbance resulting in the robot oscillating and eventually tipping over, which can be seen in figure 13.

Comparing the results from the different controllers, they suggest that an LQR controller is better than the PID to self balance a robot. However, it might be that the PID is not properly tuned. It is also worth noting that the EV3 with leJOS had a limitation in the speed of the control loop. The minimum time per loop was around 16 ms, and it was not possible to go lower than that if the values from the accelerometer and gyroscope were to be collected. It is possible that a PID controller would work better with a lower downtime per iteration. One could try doing the control strategies in C or the EV3 language, which could help with reducing the delay. Furthermore, it is possible that both controllers could have worked better if 4 motors were used instead of 2 so the robot could recover balance from steeper angles.

As for the filters it looks like the Kalman filter performed better than the complementary filter to estimate the angle. That being said, the implementation has not taken care of the bias offset from the gyroscope which might be what makes the complementary filter follow the gyroscope so closely. So if one takes care of the bias offset from the gyroscope and then use the complementary filter the result might be closer to the Kalman filter.

References

[1] Bong Seok Park Byung Woo Kim. *Robust Control for the Segway with Unknown Control Coefficient and Model Uncertainties*. June 2016. URL: http://dx.doi.org/10.3390/s16071000.

[2] Rolf Johansson. *Predictive and Adaptive Control*. 2015.

[3] Kristian Sloth Lauszus. *A practical approach to Kalman filter and how to implement it*. Sept. 10, 2012. URL: http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/.

[4] Lauren Valk. *Tutorial: Building BALANC3R*. June 23, 2014. URL: http://robotsquare.com/2014/06/23/tutorial-building-balanc3r/.

Balancing Suitcase

David Dahlgren*, Marie Jeanson†, Peter Jonsson‡ and Martin Ottenklev§
Supervisor: Olof Troeng¶
Lund University
*abt10dda@student.lu.se, †ma2488je-s@student.lu.se, ‡tpi13pjo@student.lu.se, §tfy15mot@student.lu.se
¶Olof.Troeng@control.lth.se
https://gitlab.control.lth.se/regler/FRTN40/group-C

***Abstract*—The aim of this project was to make a suitcase balance on one of its short edges. This was done by changing the position of small masses along two of the sides, thus altering the center of mass for the suitcase. An accelerometer kept track of the inclination of the suitcase. Two control methods were investigated: cascade controller and Linear Quadratic Regulator (LQR), where the former was been the main focus. The inner loop performs very well, but work remains regarding implementation and tuning of the outer loop. Due to several problems, mostly concerning electromagnetic disturbances in the accelerometer chip/cables, the performance is not yet satisfactory. The accelerometer per se is demonstrating good behavior when the motor is not receiving power, but as the motor is turned on, some serious EMC noise problem arises. This results in that the regulator acts on an incorrect angle and cannot stabilize the construction. As accelerometers tend to be noisy (even without EM disturbances) a complementary filter was implemented. Due to the fact that we did not manage to balance the construction that was to fit inside the suitcase, no suitcase has been bought. The finished construction can be seen in Figure 1.**

I. Introduction

The purpose of this project is to build a suitcase that will use a stabilization system to balance on one of its short edges. This will be achieved with a brushless synchronous motor with Hall sensors, a 3D accelerometer and 3D gyroscope named LSM6DS3, all of that piloted by a Raspberry Pi 3B. The aim is to be able to control the angle of inclination of the suitcase center of mass relative to upright position by controlling the position of one or two weights inside of the suitcase, so that the angle of inclination is always close to zero. The regulators of choice are cascade control and Linear Quadratic Regulator (LQR). With cascade control the problem can be divided into subproblems, making the entire process a bit less cumbersome to deal with. The idea behind LQR is that the control may be optimized based on desired behavior in certain states, such as inclination, mass movement and applied motor force.

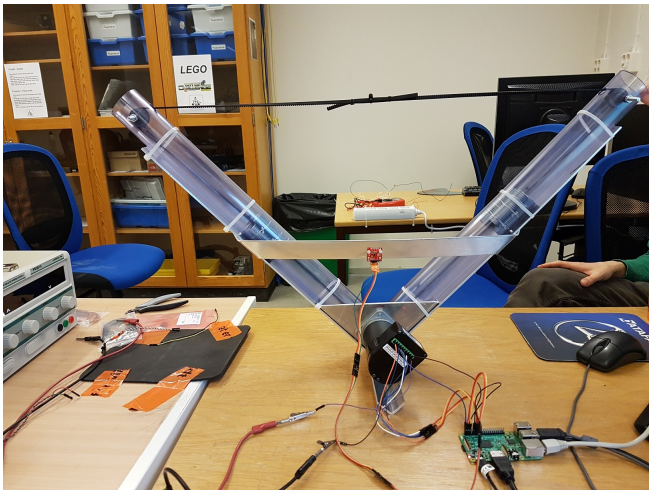


Fig. 1. Final construction

II. Design

In order to simulate and implement a controller for a balancing suitcase, a model of it is needed. To find one, some basic decisions were made about how to stabilize the suitcase. After some discussions on how to design the construction, three different construction models were suggested for this project. The fundamental design concepts can be seen in Figures 2, 3 and 4. There are some pros and cons with the different models, both regarding mathematical modeling and construction of the design.

A. Mathematical modelling, pros and cons

- Design a: The mathematical model is fairly easy, much due to the fact that distance from the small mass to the origin is constant.
- Design b: Modeling is quite difficult as the distance between the small mass and the origin depends on where the mass is positioned along the path.
- Design c: More difficult to model than configuration a, but easier to model than configuration b as the distance between masses to origin is angle-independent.

B. Design construction, pros and cons

- Design a: May be difficult to construct the bent shape in a satisfactory way. Some control difficulties may also arise from the fact that a moment force in the opposite direction will appear whenever the mass is accelerated.
- Design b: Easier to build than configuration a, but the problem with the moment force in the opposite direction is still present.
- Design c: Construction wise it is fairly similar to design b. The problem with moment force in the opposite direction is not present in this case.

Taking the different pros and cons into account, the group decided to go with design c, mainly in order to minimize the problem with opposing moment force.

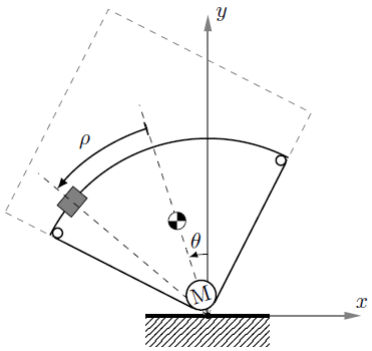


Fig. 2. Design configuration a. Moving a single weight with a fixed distance from the ground contact point of the suitcase.

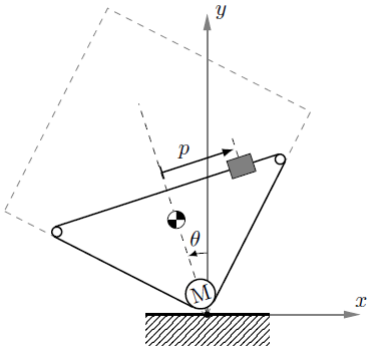


Fig. 3. Design configuration b. Moving a single weight in a straight line.

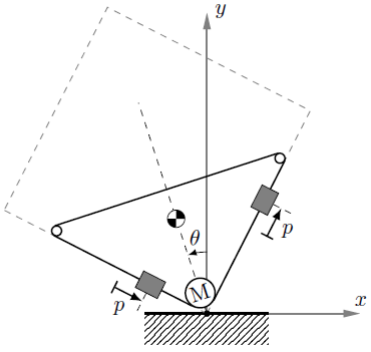


Fig. 4. Design configuration c. Moving two weights along the sides of the suitcase.

III. Modelling

In order to compute a state space equation, Lagrangian mechanics was used. The different parameters for the system are

- θ - angle of the suitcase center of mass relative to upright position.
- M - mass of the body (everything besides the small masses).
- m - mass of the small masses. Preferably they will be equal.
- α - angle of the weights relative to the center of the body. This will be constant for a given design.
- d - distance from origin to center of mass.
- r - distance from origin to nominal point of the small masses.
- p - deviation from nominal point r, for the small masses.
- J - moment of inertia for the body around origin.
- g - gravitational constant.
- F - motor force

A. Mathematical model using Lagrangian mechanics

Kinetic energy for the body:

$$T_{\text{body}} = \frac{1}{2} J\dot{\theta}^2. \tag{1}$$

Potential energy for the body:

$$V_{\text{body}} = Mgd\cos(\theta). \tag{2}$$

Kinetic energy for the small masses:

$$\begin{aligned} T_m &= \frac{1}{2}((r+p)^2 + (r-p)^2)\dot{\theta}^2 + 2m\dot{p}^2/2 \\ &= m(r^2+p^2)\dot{\theta}^2 + m\dot{p}^2. \end{aligned} \tag{3}$$

Potential energy for the small masses:

$$\begin{aligned} V_m &= mg(r+p)\cos(\theta-\alpha) + mg(r-p)\cos(\theta+\alpha) \\ &= mg(r+p)(\cos(\theta)\cos(\alpha) + \sin(\theta)\sin(\alpha)) \\ &+ mg(r-p)(\cos(\theta)\cos(\alpha) - \sin(\theta)\sin(\alpha)) \\ &= 2mgr\cos(\theta)\cos(\alpha) + 2mgp\sin(\theta)\sin(\alpha). \end{aligned} \tag{4}$$

Inserting $\alpha = \frac{\pi}{4}$ (meaning the angle between the two sides is $\frac{\pi}{2}$ rad) in equation 4 yields

$$V_m = \sqrt{2}mgr\cos(\theta) + \sqrt{2}mgp\sin(\theta). \tag{5}$$

Using equations 1, 2, 3 and 5 yields the Lagrangian

$$\begin{aligned} L &= T_{body} + T_m - V_{body} - V_m = \\ &= \frac{1}{2}J\dot{\theta}^2 + m(r^2+p^2)\dot{\theta}^2 + m\dot{p}^2 - Mgd\cos(\theta) \\ &\quad - \sqrt{2}mgr\cos(\theta) - \sqrt{2}mgp\sin(\theta) \end{aligned}$$

Solving the equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = 0$$

results in

$$\begin{aligned} &(J + 2m(r^2+p^2))\ddot{\theta} + 4mp\dot{p}\dot{\theta} \\ &\quad = (Mgd + \sqrt{2}mgr)\sin(\theta) - \sqrt{2}mgp\cos(\theta). \end{aligned} \tag{6}$$

In a similar fashion

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{p}} - \frac{\partial L}{\partial p} = 0$$

gives

$$2m\ddot{p} = 2mp\dot{\theta}^2 - \sqrt{2}mg\sin(\theta) + F. \tag{7}$$

B. State-space form

The states can be introduced as:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} p \\ \dot{p} \\ \theta \\ \dot{\theta} \end{bmatrix}.$$

Linearizing the model around $(p, \dot{p}, \theta, \dot{\theta}) = (0, 0, 0, 0)$ results in

$$(J + 2mr^2)\ddot{\theta} = (Mgd + \sqrt{2}mgr)\theta - \sqrt{2}mgp \tag{8}$$

and

$$2m\ddot{p} = -\sqrt{2}mg\theta + F. \tag{9}$$

With equations 8 and 9, and letting $F = u + \sqrt{2}mg\theta$ (to simplify the state-space equation), gives the state-space equation

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-\sqrt{2}mg}{J+2mr^2} & 0 & \frac{Mgd+\sqrt{2}mgr}{J+2mr^2} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{2m} \\ 0 \\ 0 \end{bmatrix} u$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix},$$

where u is an input signal to the motor, measured in volts. There are two output states in this model: y_1 is the first output i.e. the position of the masses, p, and y_2 is the second output i.e. the angle of the suitcase center of mass inclination, θ.

IV. Electro-Mechanics

A. Material and components

The three most important components have been assigned their own subheading, IV-A1 – IV-A3. All other parts have been compiled in the following list:

- Plastic tubes
- Gearwheel
- Mechanical belt
- Iron rod (to cut up and make the moving weights)
- Cables
- Cable ties
- Various metal parts (structural support, screws, pop rivets)
- Pulleys
- Power box
- Thread

1) Raspberry Pi unit: The Raspberry Pi unit is of the model 3B. It is equipped with 40 GPIO (General Purpose Input/Output) ports that will be used to communicate with the DC motor and the accelerometer. Programming is made in Python.

2) Accelerometer: An IMU (Internal Measurement Unit) named LSM6DS3 manufactured by STMicroelectronics is used to measure angles and angular velocities [4]. There are more things it can measure, but that will not be necessary for this project. Communication with the Raspberry unit is done via I^2C-bus protocol and some code for this was also found and used [1]. The IMU is connected to the Raspberry Pi using the GPIO's for SDA (data line) and SCL (clock line). Current and ground is also provided by the Raspberry unit.

3) DC motor: A brushless DC motor (largest in the BLDC-3 series) is used to change the position of the small masses found in Figure 4 [3]. The motor will be connected to a belt that will be strapped around the construction, using pulleys in the corners. According to the data sheet the motor has a torque of up to $0.6\,\mathrm{N\,m}$. Communication with the Raspberry unit is achieved via the GPIO-pins, but current is provided using an external power box, as it is operated around $24\,\mathrm{V}$. The motor has two built in Hall sensors which is can be used to determine the revolution speed and distance keeping, see Figure 5. Counting the amount of times one of the Hall sensors rises per second, and dividing by two gives revolutions per second. By checking if the other Hall sensor is high or low when a rise or fall edge is triggered, one can get the direction the wheel is turning.

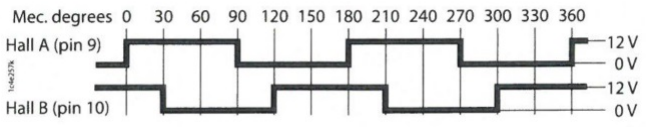


Fig. 5. The hall sensors in the motor will be high twice per revolution. Source: BLDC-3 series data sheet.

B. Construction

Construction wise, the very first thing was to get the motor running and testing the IMU-chip. Once these had been understood and tested, the construction could be initiated. A triangular metal plate was cut out and holes were drilled in order to attach the motor. This plate was then attached with pop rivets to the bottom of a large metal V, with a right angle. This V was supported with another metal piece further up. Along the sides of the V, the plastic tubes were attached with screws and cable ties. The reason for the plastic tubes was that previous years in this project, there have been some problems with the weights wobbling in directions other than along the sides. To prevent this undesired behavior, the weights traveled through tubes.
An iron rod was cut up in eight pieces, in order to create the moving weights with the weight of about $500\,\mathrm{g}$ each, as this

weight had worked in the simulation. Four of these iron rod pieces were attached to each other and to the mechanical belt with threads. They were then inserted in the plastic tubes and the belt was tightened around the gearwheel and the pulleys that had been put in the plastic tubes. For pictures of the construction see either the first page or the appendix.

V. Theory

Two control methods have been under discussion for this project, namely cascade control and Linear Quadratic Regulator (LQR). With a cascade control the problem can be divided into two subproblems, making the overall regulation a bit easier. Also, full state feedback is not needed. Only the position of the small masses and the angle θ needs to be measured. In order to be able to use LQR, full state feedback is required. A decision was made to keep main focus on the cascade controller and implement LQR if there was spare time at the end.

A. Cascade control

The inner system will describe the dynamics between the motor and the positions of the small masses while the outer system describes the dynamics between the angle θ and the positions of the small masses. In order for cascade control to work, the inner system dynamic need to be faster than the outer one. See Figure 6 for a visual representation. Coding will be done directly on the Raspberry unit in Python. The angle reference will always be zero, so the inclination should always be zero.

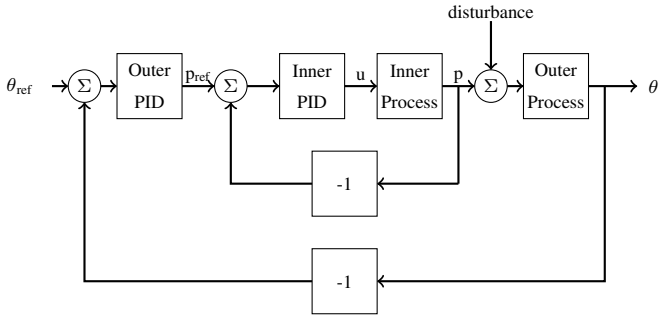


Fig. 6. Block diagram for the system. For a bit more realistic process, u should be saturated in order to mimic the limitations of the motor.

B. Cascade simulations

In order to make sure the regulators would work, they needed to be simulated. The simulation was made with Matlab Simulink. The PID parameters of the inner and outer loops were found using trial and error. The parameters of the inner loop and outer loop are in the Table I. The simulations yielded the graph that can be seen in Figure 7

C. Linear quadratic regulator (LQR)

LQR aims to minimize the quadratic cost function

$$J = \int_{t_0}^{t_1} (x^T Q x + u^T R u) dt$$

TABLE I
PID parameters from simulations.

Parameters	Inner loop	Outer loop
P	7	-1.7
I	2	-0.8
D	50	-0.3

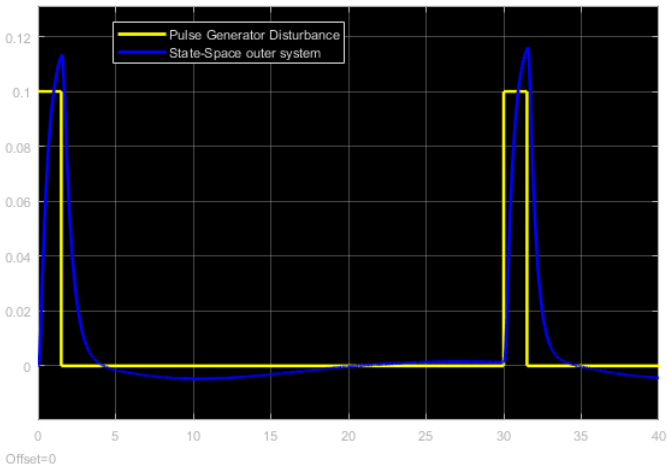


Fig. 7. Simulation of the cascade controller. The pulses simulate disturbances of 0.1 radians. Yellow line is pulse, blue is θ.

where Q and R are design parameters that can be chosen to penalize the states of the system, and the control signal. Q is a matrix with dimensions *states* $\times$ *states* (4×4 in this case) and R is a matrix with dimensions *control signals* $\times$ *control signals* (scalar in this case). See Figure 8 for a visual representation. As the goal of the project is for the suitcase to balance in upright position, the state for θ should be penalized hard in the Q-matrix. R should be chosen such that the control signal u does not exceed the maximum motor force. Then a matrix is calculated with respect to the cost function and used to multiply with the states to get a feedback in the usual LQR manner further described in Glad and Ljung [2].
The simulations of the LQR has not been included due to the fact that it's not the controller that was used in the project.

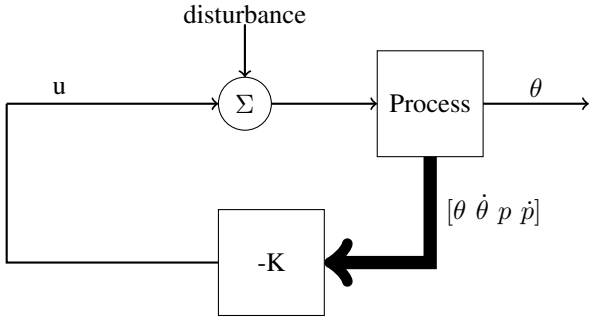


Fig. 8. Block scheme for process controlled by LQR.

D. Complementary filter

As accelerometers generally suffer from a lot of high frequency noise a filter is needed to sort out relevant data, in this case angle measurements. Several methods are available and the one used here is the so called complementary filter. On the short term it uses data from the gyroscope as it is quite precise and not susceptible to external forces. On the long term it uses data from the accelerometer as it generally does not drift. The filter is an iterative process that can be written as

$$\theta_{k+1} = \alpha\ (\theta_k + \omega_k \cdot dt) + (1 - \alpha) \cdot (accData_k)$$

where θ is the angle, ω is the angular velocity (gathered from the gyroscope function of the accelerometer), dt is the sample time, $accData$ is the unfiltered angle from the accelerometer and α is a weighting parameter. $k+1$ and k denotes instances of time.

VI. Coding

The coding was made in the language Python. A structure of how the code is built can be seen in figure 9.

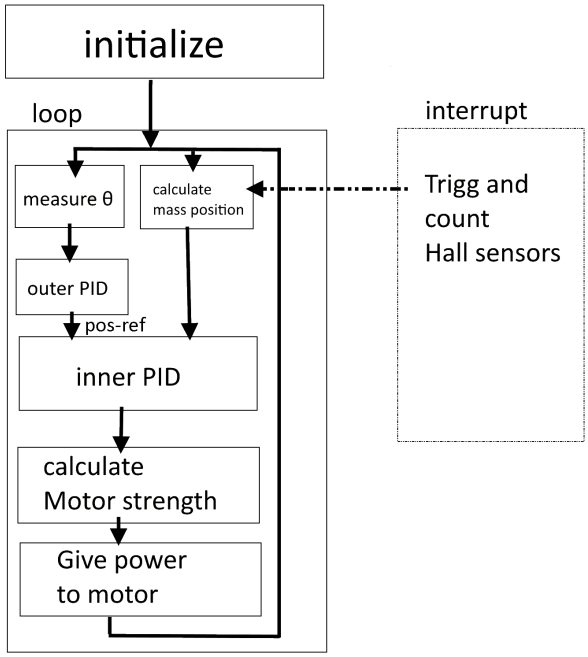


Fig. 9. The coding structure of the entire process.

In the *initialize* block the Raspberry Pi was set up. The data pins was set up, constants initialized and libraries imported. Then the *loop* is started, where the current θ is fetched from the IMU-chip and the mass positions is calculated. The angle of the construction is sent to the *outer PID* which calculates what the angle should be (*pos-ref*), which together with the current mass position is being sent to the *inner PID*. The necessary motor strength is calculated and sent to the motor. Then to process starts over. Meanwhile there is a trigger for when the Hall sensors are changed from HIGH to LOW or vice versa (see Figure 5 to calculate how many revolutions the wheel has turned and the hence how far the masses has moved.

VII. Results

So far we have not been able to balance the construction that is supposed to fit inside a suitcase. As the center of mass will be shifted to a trickier position with the suitcase implemented we want to stabilize the construction before we attach it to a suitcase. Furthermore we are hoping to be able to minimize the construction if the regulators behave satisfactory. LQR implementation has mostly been put aside in order to work with the cascade control. Regarding the cascade controller the inner loop is stabilized with good performance. Performance of the outer loop is difficult to evaluate as the motor is generating electromagnetic interference that affects the accelerometer.

A. Complementary filter

After some trial and error we found that the filter behaves satisfactory with $\alpha = 0.9$. See Figure 10

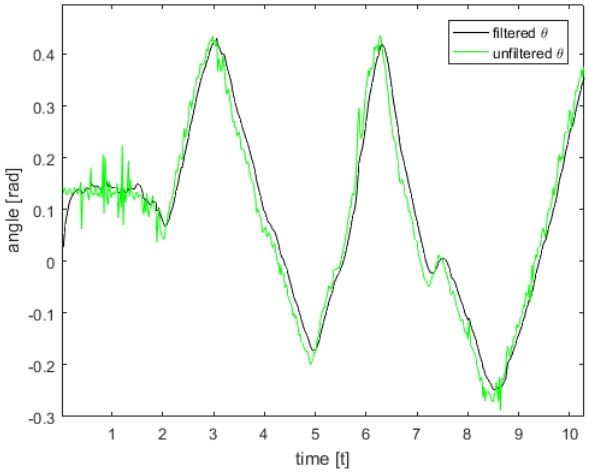


Fig. 10. Complementary filter with $\alpha = 0.9$. Construction moved by hand while pwm is set to zero. As the filter is iterative the filtered angle will always be one step behind the unfiltered one.

B. LQR

Since the cascade controller is not yet fully operational, the effort spent on LQR has been quite low. A few runs has been tried out, but most of them results in bang-bang control, which results in a bit of slipping between gear and strap. The reason for this is not yet clear, but a few different explanations are possible.

- The process may be very unstable, rendering the linearized point very hard to reach.
- Code may need to be optimized. If each iteration takes too long the system may not be able to react before the construction is too far from the linearization point.
- Further tuning of design parameters Q and R is needed.

C. Cascade controller

Problems have been connected to the position of the small masses. As the resolution of the hall sensors in the motor is $90°$, a position error of $\pm 2\,\mathrm{cm}$ is present. This error may propagate as the masses are moved back and forth. At this point the inner loop is working very well. See Figure 12 for how the masses follow the position reference. For the outer loop to work properly a bit of work remains regarding implementation and tuning. PID parameters differs greatly from simulated results. Note both that $D = 0$ in the inner loop, and close to zero in the outer one. Note also and that the inner P-parameter is significantly larger than in the simulations.

TABLE II
PID PARAMETERS FROM EXPERIMENTS. COMPARE TO TABLE I FOR SIMULATION PARAMETERS.

Parameters	Inner loop	Outer loop
P	300	0.5
I	2	3
D	0	0.08

1) Inner loop: The inner loop is properly closed and performance for different tunings of the PID regulator may be seen in Figure 11.

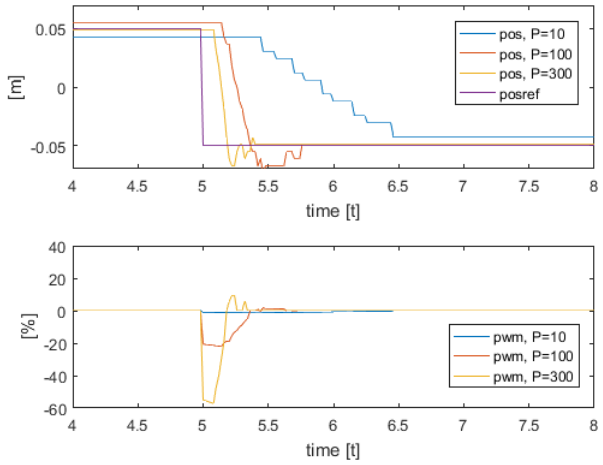


Fig. 11. Step response of the inner loop and pwm strength for different P-values, all with I = 2 and D = 0.

VIII. DISCUSSION

Over the course of the project we have encountered numerous problems of different impact. One major issue has been how to tighten the strap enough to avoid slipping when changing direction of the gear (from i.e. clockwise rotation to counterclockwise rotation). Two ropes has been destroyed in this process before getting the final belt. As mass positions are not measured directly, rather we measure the total revolutions of the gear using the hall sensors and convert it to mass positions, slipping may lead to masses not moving although the hall sensors indicate that they do. Since the cascade controller

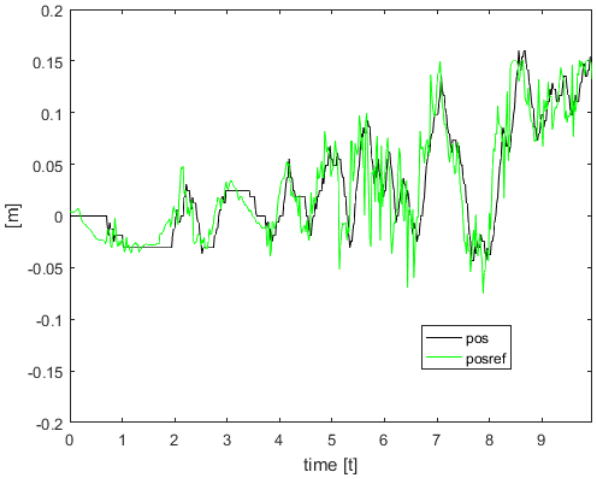


Fig. 12. Position and position reference of the masses when the entire process is running. The construction is not able to balance on its own. Objects are placed on the sides thus making it impossible for the construction to lean more than $\pm$ 10 degrees.

may be divided into two problems we decided to put or focus on that over the LQR implementation. Problems also occurred with cables that did not work when the motor was tested for the first time which took time to find out. With the cascade control the loops could be tuned separately, whereas in the LQR everything has to be tuned at once.

Even when the complementary filter is implemented the accelerometer measurements become rather noisy. After holding the construction at place by hand and running the motor, with the position reference altering between ± 5 cm, we noticed that electromagnetic interference is present. See Figure 13. This is a major problem for which several solutions were tested, including twisting pairs of cables (to cancel out the EMC interference) and implementing different kinds of filters. However, nothing was successful and this disturbance in the angle is probably the main reason for failure. The results of this is that the control signal of the outer loop, position reference for the masses, gets noisy. This can clearly be seen in Figure 12.

The discrepancy between simulated and experimental PID parameters is quite big, most notably for the inner P-parameter and the D-parameters. Implementing derivative parts proved to be a real struggle, probably due to the fact that the system is working in discrete time. This could be the explanation for the large value of the inner P-parameter. One could also take note that no friction is present in the mathematical model while there is a lot of friction in the construction.

Sometimes when pwm is roughly 40 or larger, the motor emits a high pitched noise at the same time as the current provided by the power box increases fast up towards 5-6 Ampere. Therefore we have chosen to set a limitation at pwm ≤ 30.

As can be seen in Figure 11, there is a small delay in the response. Approximately 0.1 seconds for P = 300. This of

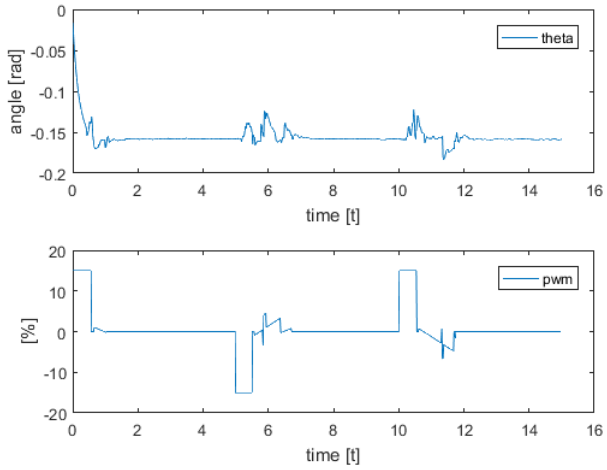


Fig. 13. Theta is in reality constant (construction held at place by hand), but when the motor is running it generates EM-interference resulting in accelerometer data being nonconstant.

course has a negative effect on the result as you want the fastest possible inner loop.

Implementing the derivative part of the PID-regulators proved to be very difficult. Oftentimes it took over the entire regulator leading to the masses more or less simply oscillating within a very short interval from the start position, regardless of position reference. We suspect that this is tied to the fact that derivative gain in discrete time may be very large as the time steps are very small while the measurement differences may be relatively large. Ultimately the derivative parts were set to zero in the inner loop and close to zero in the outer one.

Friction was largely present in the construction. Preferably the circumference of the masses should have been a bit smaller to avoid this. Aside from simple plastic wheels proper bearings could be used as pulleys.

Another problem could be that the motor that was used has a certain, unknown, transfer function, which makes it more difficult to understand how the control signal is really related to the output. It would possibly have been good to make some sort of system identification of the motor, or open it up and make a controller for the motor it self.

Much has been learned in this project. No prior experience with Raspberry Pis existed, but the versatility of the Raspberry Pi has been made clear to the group. Prior to the project only one group member had used Python as a programming language. Also Git was new to the entire group.

References

[1] Maximilian Britton. *Raspberry-Pi-4x4-in-Schools-Project*. Dec. 2017. URL: https://github.com/mwbritton/Raspberry-Pi-4x4-in-Schools-Project.

[2] T. Glad and L. Ljung. *Reglerteori - Flervariabla och olinjra metoder*. Lund, Sweden: Studentlitteratur AB, 2003.

[3] *Information sheet*. Östergrens Elmotor. 2008.

[4] *LSM6DS3 Datasheet*. SparkFun Electronics. 2015.

Appendix

A. Pictures of construction

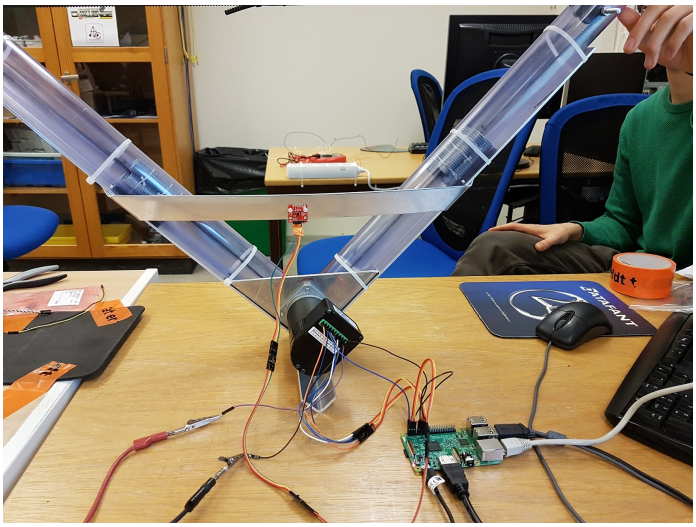


Fig. 14. The construction that is to fit inside a suitcase. Masses can be seen inside the tubes. The IMU is the red chip, and the motor is the black box.

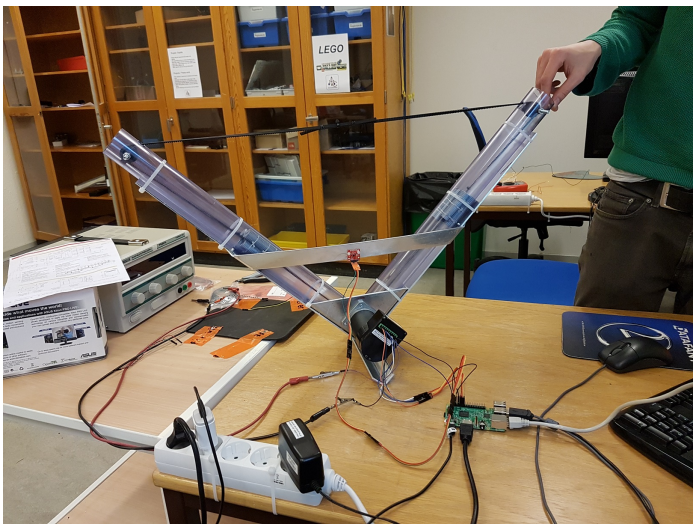


Fig. 15. The construction.

Balancing Suitcase

Henrik Fryklund*, Lucas Lindén† Johan Lindqvist‡, and Martin Sollenberg§
Lund University
*kem12hfr@student.lu.se, †psy12ljo@student.lu.se, ‡elt13jli@student.lu.se, §elt13mso@student.lu.se
https://gitlab.control.lth.se/regler/FRTN40/group-D

***Abstract*—Today we see Segways and Hoverboards daily in our cities and in our streets. This project will use the principles of these inventions to balance a suitcase on one of its short edges. With a reaction wheel placed inside the suitcase, Newtons third law of motion enables us to do just that. The control system, developed in python, is executed on a Raspberry Pi. Using a IMU to read the tilt angle and angular velocity of the suitcase, a control signal is calculated and sent to a brushless DC motor connected to the reaction wheel. Using the LQR control principle to achieve optimal state feedback, the speed of the reaction wheel can be minimized at all time. Minimizing the speed of the wheel is the key to giving the system the ability to maximize the acceleration of the reaction wheel in the needed direction at any time. The Balancing Suitcase can handle small load disturbances if the disturbance is in the opposite direction of the velocity of the reaction wheel. So, one can make a suitcase balance on its short edge, but the storage remaining is utterly limited and thus, the purpose is exchanged from traveling, to the demonstration of an inverted pendulum at fairs and lectures.**

Fig. 1. The balancing suitcase

I. Introduction

Balancing an inverted pendulum is a common showcase problem in control theory, a task which can be quite difficult for a human to do. Automatic control quickly demonstrates its use as it can perfectly balance the inverted pendulum. The simple dynamics of the problem lay ground for e.g. rockets and Segways.

This project was part of the course FRTN40 ”Projects in Automatic Control” held by the department of Automatic Control at Lund University, fall 2017. The goal of the project was to build an inverted pendulum in the form of a suitcase, construct the hardware needed and implement a controller to balance the suitcase on one of its short edges. This could have been done in several ways, but two different options were presented to two project groups by the project supervisor. One option was to use weights in order to balance the suitcase and the other was to use a reaction wheel. This group decided to go with the reaction wheel concept while the other project group went with the weights. Inspiration for the project has been drawn from The Cubli [4], which is a more advanced version of this project. The scope of the project was eight weeks and the work flow consisted of modeling of the system, construction of the hardware and implementation of the software. Followed by verification, testing, and lastly a project report and presentation.

II. Model

A. Modeling

A simplified model of the Balancing Suitcase can be seen in Fig. 2, the parameters can be found in Table I. Using this model a mathematical model was derived using Lagrangian mechanics[3]

$$L = T - V, \tag{1}$$

where L is the Lagrangian, the difference between the kinetic energy T and the potential energy V of the system. Which were derived as

$$T = \frac{1}{2}J\dot{\theta}^2 + \frac{1}{2}J_w\omega^2 \tag{2}$$

and

$$V = mgr\cos(\theta). \tag{3}$$

The Lagrangian equation

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q_j}}) = (\frac{\partial L}{\partial q_j}),$$

where q is a generalized coordinate, was used with equation (1), (2) and (3) to model the system, yielding the derived differential equation

$$\ddot{\theta} - \frac{mgr}{J}\sin(\theta) = \frac{-\tau}{J}. \tag{4}$$

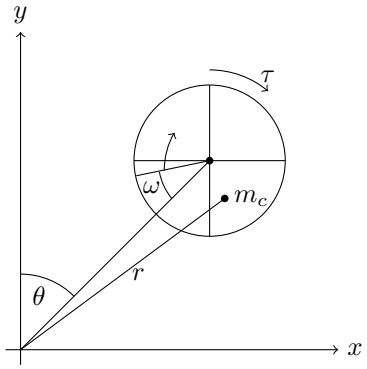


Fig. 2. Simplified model of the Balancing Suitcase with parameters. m_c in the figure is an arbitrary location of the center of mass.

TABLE I
PARAMETER DECLARATION.

Variable	Unit	Description
θ	rad	Angle around vertical axis
$\dot{\theta}$	$\mathrm{rad\,s^{-1}}$	Angular velocity around vertical axis
ω	$\mathrm{rad\,s^{-1}}$	Angular velocity of reaction wheel
g	$\mathrm{kg\,m\,s^{-2}}$	Gravitational constant
r	m	Distance to center of mass of the system
J	$\mathrm{kg\,m^2}$	Moment of inertia of the system
J_w	$\mathrm{kg\,m^2}$	Moment of inertia of reaction wheel
τ	$\mathrm{N\,m}$	Motor torque
m	kg	Mass of the system
h	s	Sampling time of the system

Equation (4) describes the system without friction and motor dynamics. The relation between the motor torque and acceleration of the wheel was formulated as

$$\dot{\omega} J_w = \tau. \tag{5}$$

Table I describes all parameters used in the model.

B. Linearization and State-Space

The differential equation (4) was linearized around $\theta = 0$ and $\sin(\theta) \approx \theta$ for small angles of θ.

By denoting the state vector as

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \\ \omega \end{bmatrix}$$

the equations (4) and (5) can be represented in the following continuous state-space model

$$\dot{x} = Ax + B\tau = \begin{bmatrix} 0 & 1 & 0 \\ \frac{mgr}{J} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{-1}{J} \\ \frac{1}{J_w} \end{bmatrix} \tau.$$

The system was discretized, yielding the discretized system on the following form [1]

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma\tau(kh) \\ y(kh) &= Cx(kh) \end{aligned} \tag{6}$$

where h is the sampling time and k the period

$$\begin{aligned} \Phi &= e^{Ah} \\ \Gamma &= \int_0^h e^{As} ds B. \end{aligned}$$

III. CONTROL

A. Control Theory

The Linear Quadratic Regulator (LQR) is an optimal state feedback regulator that controls a dynamic system by minimizing the following function for an infinite-horizon discrete system on the form (6)

$$J = \sum_{k=1}^{\infty} (x_k^T Q x_k + u_k^T R u_k + 2 x_k^T N u_k) \tag{7}$$

where Q and R are weight matrices for the state vector x and the control signal u. N is the correlation between the state and input signal. By solving for the optimal feedback, one yields that the optimal static feedback law [2]

$$u(t_k) = -Lx(t_k)$$

$$L = (B^T S B + R)^{-1}(B^T S A + N^T)$$

where S is the unique, positive semi-definite symmetrical solution obtained by solving the discrete time algebraic Ricatti equation [2]

$$A^T SAS(A^T SB + N)(B^T SB + R)^1(B^T SA + N^T) + Q = 0.$$

B. Control Implementation

As the system is static, a LQR with a static feedback law will be sufficient to control the process because of the possibility to decide the weight parameters for the state transitions and the control signal in the cost function, equation (7). The controller should be able to keep the control signal to a minimum and stabilize the system at its unstable equilibrium. The control signal is scaled by a factor 100. This gives a sufficiently large change in velocity, to produce an acceleration of the reaction wheel, large enough to stabilize the suitcase. As the output signal is on Pulse Width Modulation (PWM) form, there is a need for saturation of the signal as the limits for the PWM are [0, 100] %.

IV. Simulation

To simulate the behavior of the system, a Simulink model was created which can be seen in Fig. 3. The motor was roughly estimated with a saturation of its maximum output torque and a time delay for the torque. The parameters for the simulated model were measured and entered into a separate MATLAB script where the state-space model and the controller were calculated. Zero mean white Gaussian noise was added to the measured output signals and a disturbance on the input signal in order to verify the stability of the model.

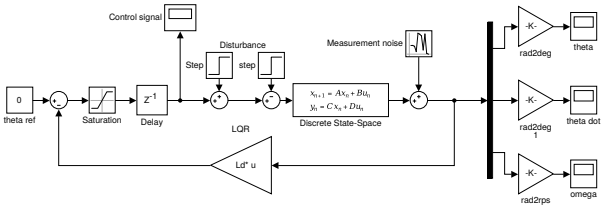


Fig. 3. Simulink model for the derived mathematical model with a LQR controller

V. Electro-Mechanics

This section will describe the different mechanical and electrical components of the Balancing Suitcase. The placement of the components can be seen in Fig. 4 and their mechanical parameters Table II.

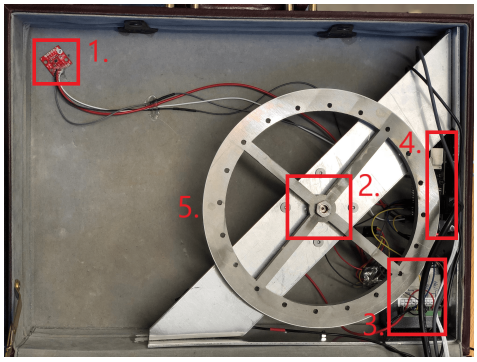


Fig. 4. Component placement inside the Balancing Suitcase. 1. IMU: 2. Motor: 3. Embedded Motor Controller: 4. Raspberry Pi: 5. Reaction Wheel

A. Raspberry Pi

A Raspberry Pi 3 Model B[1] was used as the processing unit in the Balancing Suitcase. The Raspberry Pi has 40 General-purpose input/output (GPIO) with support for Inter-Integrated Circuit (I^2C) communication and PWM which was needed to communicate with the IMU and control the Motor.

[1]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

B. Inertial Measurement Unit

The Balancing Suitcase used an Inertial Measurement Unit (IMU) of model LSM6DS3, which has an accelerometer and a gyroscope that was used to measure the angle and velocity at which the Balancing Suitcase was currently tilted. The protocol used for communication between the Raspberry Pi and the IMU was I^2C.

C. DC Motor

The reaction wheel was driven by a brush less DC motor of model BLDC3-ZWX01. The motor had an embedded controller which controlled the speed, direction and braking of the motor. Two hall sensors were available and were used to measure the speed of the motor. The default output of the hall sensors was $12\,\mathrm{V}$ and $0.1\,\mathrm{mA}$, which is above the input voltage level for the Raspberry Pi. Due to a constant output current, it is enough to place a resistance between signal and ground to get the desired voltage output of 3.3 V.

D. Reaction wheel

The reaction wheel was designed with spokes and a large rim in order to maximize the moment of inertia while minimizing the mass. Small holes along the rim were added to enable increments in the moment of inertia. The reaction wheel was water cut in aluminum. In Fig. 5 the reaction wheel design is shown.

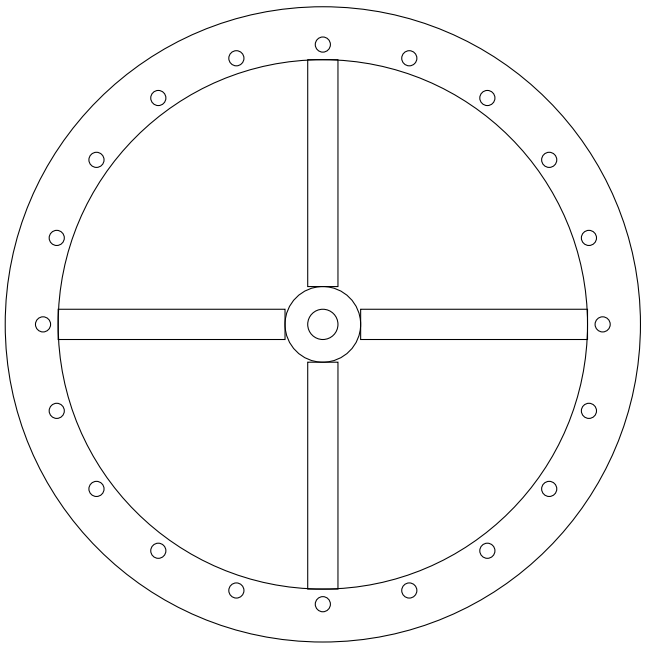


Fig. 5. Design of reaction wheel.

E. Suitcase and Frame design

The suitcase attributes can be seen in Table II, where only the heavier and larger components are included. A few components from the inside of the original suitcase were removed in order to make room for the frame. The frame

construction consists of a triangle shaped aluminum frame designed to fit inside the Balancing Suitcase. The design of the frame is shown in Fig. 6. The frame was used in order

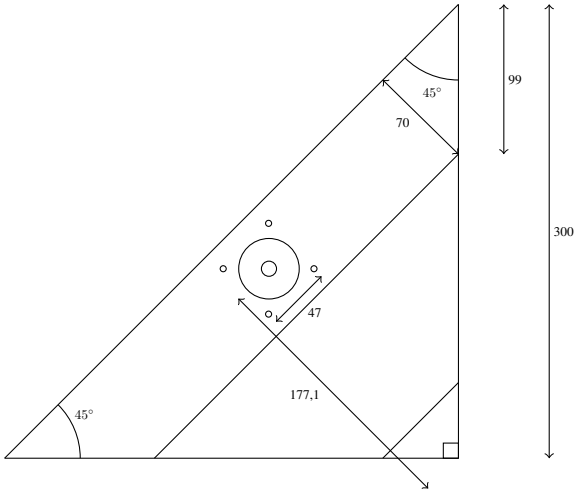


Fig. 6. Sketch of frame design. The measurements presented in the sketch are in mm.

to stabilize the Balancing Suitcase and to mount the hardware components.

F. Component placement

Most of the components are placed in the balancing corner of the suitcase, this to ensure a low center of mass, see Table II for values. This allows the motor to handle greater angles. The motor was placed further away from the balancing corner, to make room for the reaction wheel. The design of the frame was made with aspect to keep the distance from the balancing corner to the motor short, i.e. to have a low center of mass. The only exception for placement of the components close to the balancing corner was the IMU, which instead was placed in the top corner in order to maximize its sensitivity.

TABLE II
MECHANICAL PARAMETERS

Parameter	Unit	Value
Motor with frame:		
Mass	1.18	kg
Suitcase:		
Mass	1.86	kg
Height	0.48	m
Width	0.33	m
Wheel:		
Mass	0.44	kg
Thickness	1.0	cm
Inner radius	9.80	cm
Outer radius	11.80	cm

VI. SOFTWARE

The software was developed in Python, which was divided into classes where each class had a specific task. The following subsections goes through the purpose of each class.

A. Hall

In order to measure the speed of the motor, communication with the hall sensors had to be established. The output from each of the hall sensors is a square wave with two periods per mechanical revolution. Sensor B is phase shifted 30 degrees from sensor A.

The output from sensor A was read when an event was triggered on either a rising or falling edge on the Raspberry Pi GPIO. The wheel had to revolve a quarter of a revolution in order to update the speed of the motor. By keeping track of the level of hall sensor B, the direction of the wheel could be determined. When the controller reads the latest values, it is the median of the latest three stored values in order to reduce noise and falsely triggered events.

B. IMU

The IMU continuously measured the acceleration and the rotational movement of the Balancing Suitcase. The acceleration was measured by an accelerometer which is stable for low frequencies but have high frequency noise. The rotational movement was measured by a gyro which is good at measuring high frequencies but suffers from DC-drift. To get reliable measurements, sensor-fusion through a complementary filter was used. When called upon, the IMU class returned the output from the complementary filter

$$\theta_k = \alpha(\theta_{k-1} + \gamma h) + (1 - \alpha)\delta \tag{8}$$

with raw data from the IMU as input. α is a weighting constant for the filter, h is the sampling time, γ is scaled data from the gyroscope and δ the measured low-pass scaled acceleration data. The raw values from the IMU are thus converted to the tilt angle of the Balancing Suitcase as well as the angular velocity. An offset of the IMU tilt angle in upward position can be adjusted.

C. Motor

The motor class contains the methods for controlling the motor. The methods are used for starting and stopping the motor, changing duty cycle of the PWM signal and changing direction of the wheel.

D. Control

The control class has a reference to the Hall, IMU and motor classes, i.e. it has access to their methods. The controller starts by reading its control parameters from a configuration file. When the setup is finished, the control loop starts. The control loop calls the measurement methods of the Hall and IMU classes. These values are then used for calculating the control signal which is sent to the motor class.

E. __main__

The __main__ class acts as an initiator of the other classes and handles shutdown of the Balancing Suitcase.

VII. RESULTS

This section presents the results of the project, divided into the subcategories; Simulation and Implementation.

A. Simulations

The system according to the Simulink model in Fig. 3 was simulated with an initial value on the angle of the suitcase, $\theta = 2°$. The velocities of the suitcase and wheel where initially set to zero. The input reference was set to zero to force the suitcase to be in its upward position. A step disturbance was added at time 7 s to time 7.5 s. As can be seen in the result in Fig. 7, which shows the angle of the suitcase, the controller was able to stabilize the system from the initial value and also handle the step disturbance. The weight matrices were

$$Q = \begin{bmatrix} 100000 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

and

$$R = 1000 \quad N = 0$$

for equation (7). The following feedback vector was produced

$$L = [-42.8509 \quad -6.4332 \quad -0.0743].$$

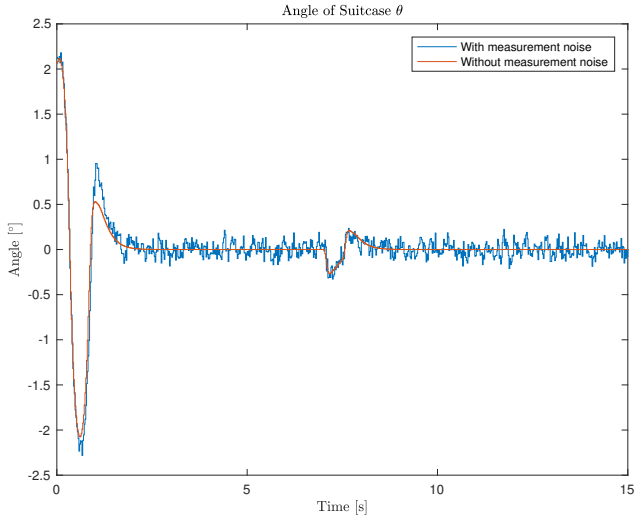


Fig. 7. Angle θ for the suitcase from simulation, with and without measurement noise

B. Implementation

The main idea for the frame design was inspired by one of the previous projects of the Balancing Suitcase which left their frame for future groups. The constructed frame could be fitted snugly in the suitcase and could be mounted with screws and nuts through the sides of the suitcase. All electronic components were fastened with screws or Velcro inside the suitcase. The reaction wheel was successfully mounted with an expansion bolt onto the motor axis. The final construction was sufficiently rigid for the suitcase to balance on its corner.

The software was fully implemented in Python on the Raspberry Pi. After calibrating the weight matrices the Balancing Suitcase was functional, with the parameters

$$Q = \begin{bmatrix} 10000 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$R = 1 \quad N = 0$$

for equation (7). The following feedback vector was produced

$$L = [-108.0241 \quad -16.3314 \quad -0.2529]. \tag{9}$$

The last value of the L vector was reduced by a factor ten in order to improve the results. In figure Fig. 8 the measured angle for the real suitcase is plotted. At the beginning the suitcase is put into place by hand and therefore an increasingly larger angle can be seen there. Then it can be seen that the angle is stabilizing, though around a small offset of approximately $0.6\,°$. The variance of the angle measurements when stabilized is approximately $0.02\,°^2$.

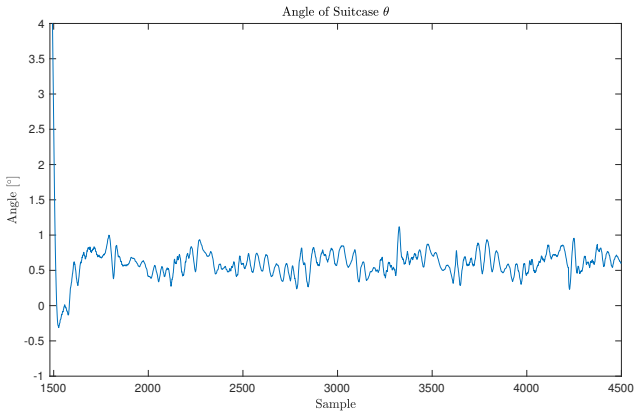


Fig. 8. Angle θ for the suitcase from real measurements

The main results of the project was the successful construction and the implementation of the controller in the Balancing Suitcase.

VIII. Discussion

During the duration of the project there was a fundamental focus on keeping the center of mass low. The idea behind this was to allow large angles since the magnitude of the needed torque for any angle would be smaller with a lower center of mass. However, another approach would be to construct the suitcase with a higher center of mass, thus rendering the whole system slower and maybe more robust.

What posed the greatest obstacles during the project was the embedded motor controller, getting usable readings from the sensors and the reaction wheel design.

During testing, the embedded motor controller proved problematic. The embedded controller was aggressive with overshoots on the velocity and gave maximum torque with every

change of the PWM signal. The idea of using a lookup table was thought of but discarded for a solution with integrating the signal. The signal was then functional, but caused the system to be slower and thus lose stability, so it was removed. Attempts at smooth torque control ultimately failed, due to the unknown characteristics of embedded motor controller.

The importance of thoroughly validating the signals from the IMU was underestimated and that caused some trouble. For some time during the project, signals that were reasonable but not correct were used, and caused some issues and instability. This was resolved by conducting thorough tests for calibration of the filter. The same thing happened with the Hall sensors. Initially, the resolution of the hall sensors was too poor and could not handle values close to zero effectively.

When designing the reaction wheel there was an issue with the first iteration. The result was an unbalanced wheel, unusable in the project. The second iteration was successful, however it was severely delayed and the first iteration was instead lathed and balanced to avoid falling further behind schedule. Thus the second reaction wheel has not been tested and the currently mounted might be slightly unbalanced.

The Simulink model is missing the true motor dynamics, which caused the simulations to not yield a good feedback law. Proper motor dynamics would have altered the model of the system and thus the feedback law. Perhaps giving a feedback law not needing altering.

The reason for reducing the third L parameter in equation (9), which corresponds to ω, was that we saw a correlation between ω, θ and the control signal. The correlation was that ω and θ multiplied with its respective L parameter canceled out, causing the control signal to ignore the angle of the suitcase. This caused the suitcase to fall over. Seeing this correlation we lowered the third L parameter.

The main results were satisfactory even though, as can be seen in Fig. 8, the suitcase balances around an offset of $0.6\,^{\circ}$. When this offset was added to the calibrated IMU offset the performance worsened. The group discussed this offset in length but could not find a reasonable explanation, therefor it is left as is. While it can be difficult to place it successfully in balancing position, when the position is found, it is stable. The system can balance for long periods of time and even handle small disturbances, such as vibrations in the balancing surface and nudges to the suitcase itself. The Balancing Suitcase could be improved, how and why is mentioned in the section future work. The biggest lesson taken from this project is the importance of handling signals correctly and the importance of choosing the components based on solid research.

IX. Conclusion

The inverted pendulum problem can be solved in many ways, this project has shown one of those ways; with a reaction wheel inside of a suitcase. The hardware was constructed with the purpose of making the suitcase more rigid, which it did. The control was implemented with the purpose of stabilizing the system, which it did. The Balancing Suitcase was designed to balance, which it does.

X. Future Work

The Balancing Suitcase can be further developed. One feature that could be implemented is a swing-up controller. This controller would make the Balancing Suitcase ”jump up” from a standing position to the balancing position. This would be done by accelerating the wheel to an appropriate speed, and then braking the wheel with an appropriately positioned external brake. At the balancing position the software could switch controller back to the current controller.

The project in this iteration has not implemented any dynamics of the motor. Performing system identification to map PWM signal to output torque, could improve performance in terms of stability. One way to do this is to hang the Balancing Suitcase upside down, run different PWM sequences and measure the torque.

If a more effective motor was to be installed, a battery could be installed in the suitcase and a charging connection in order to remove the negative effect of the cables on the motion.

References

[1] Karl-Erik Årzén. *Real-Time Control Systems*. KFS i Lund AB, 2014.

[2] Torkel Glad and Lennart Ljung. *Reglerteori Flervariabla och olinjra metoder*. Studentlitteratur, 2003. ISBN: 978-91-4403003-6.

[3] Joseph L. Hellerstein et al. *The Reaction Wheel Pendulum - Synthesis Lectures on Controls and Mechatronics*. Morgan & Claypool, 2007. ISBN: 978-1598291940.

[4] Igor Thommen Mohanarajah Gajamohan Michael Merz and Raffaello DAndrea. “The Cubli: A Cube that can Jump Up and Balance”. In: *in Proc. IEEE/RSJ International Conference of Intelligent Robots and Systems* (2012), pp. 3722–3727.

Acknowledgment

We would like to give special thanks to our fellow student and friend Andreas Abramsson for helping us make a CAD model of the wheel. To Giorgos Nikoleris for water cutting the wheel and to Pontus Andersson for the support of the construction of the frame and fitting the wheel to the motor. Finally we would like to thank our supervisor Olof Troeng for his support throughout the project.

Quadcopter Attitude Estimation

Jonas Alfredsson*, Ivan Llopis† and Álvaro Torregrosa‡
Lund University
*jonas.alfredsson.486@student.lu.se, †iv1122ll-s@student.lu.se, ‡al7500to-s@student.lu.se
https://gitlab.control.lth.se/regler/FRTN40/2017/group-E

***Abstract*—This report will analyze and implement the Mahony filter which is used to fuse information from two or more sensors to estimate the attitude of, in this case, a small quadcopter with limited processing capabilities.**

The attitude can only be an estimation since the quadcopter has no information regarding its orientation other than its imperfect onboard sensors. There are a couple of filters that can be used for this, however, since processing power is limited, the Mahony filter has been proven to be very efficient in regards to its computational difficulty.

The advantage of using a filter that combines the output from different sensors is that the shortcomings of either one of them can be compensated for. The high frequency accuracy from the gyroscope will be the main contributor when analyzing movement, while the low frequency stability of an accelerometer is used to compensate for the gyro drift that is otherwise inevitable.

The filter is first implemented in the readable way using rotational matrices, to then be rewritten in a more efficient way by using something called quaternions. Finally this filter is translated to the C programming language to gain the last bit of efficiency to be able to run on the low powered processor that is located on the Crazyflie quadcopter by Bitcraze. The results were a quadcopter that could fly in a controlled manner and stably hover when not receiving any control inputs.

I. Introduction

This project will explore modern methods of Unmanned Aerial Vehicle (UAV) attitude estimation, using non-linear complementary filters. The goal is to implement a useful algorithm that can be used in a real UAV to estimate its attitude and keep it from tumbling to the ground. The model of quadcopter that will be used for this is the Crazyflie 2.0 from Bitcraze (shown in figure 1), which will have its firmware modified with the help of the tools that are provided on their homepage [3].

Attitude estimation of a quadcopter is a problem in control that has been studied for years [9], but has recently gained more interest since the technology have become more affordable. Using simple methods of complementary filters are often insufficient so non-linear complementary filters, such as the Madgwick filter or the Mahony filter, are preferred as they provide much better results [13][11]. These two filters have also been designed around the limitations of the hardware these hobbyist UAVs are equipped with. Since they are designed to be as light as possible, the processor onboard tend to be very low powered, and the filters will have to be as easy to compute as possible while still providing a good estimation of the orientation that will permit the drone not to crash.

The well known Kalman filter could have been used and tuned to provide a very high degree of accuracy, but it is far more computationally intensive in comparison. For the weaker processors this filter might be too complex for the system to update the estimation with a high enough frequency that would be necessary for a stable flight. That is why alternative methods have been created in order to produce a suitable result with much lower overhead [12].

In this report the Mahony filter will be the one that is studied, implemented and used for attitude estimation on the real quadcopter. Other filters will be mentioned during comparisons, but they will not be explained in depth since that is out of the scope of this report.

To begin with the mathematical background of the system and filters will be analyzed (including rotational transformations and their parametrizations), to get a understanding of how they actually estimate the system. These mathematical functions will then have to be implemented in efficient C code to be able to run on the embedded system that is present on the quadcopter. The goal is to be able to replace the current estimator on the Crazyflie with the Mahony filter, implemented in C, and have it fly in a satisfactory manner.

All the code used in this project will be available in the GitLab repository visible in the title. If the text refers to a folder, it is probably located therein.

Fig. 1. The quadcopter used in this project. The Crazyflie 2.0 by Bitcraze.

II. Modeling

A quadcopter in flight is a very complex system, with a lot of parameters that needs to be taken into account. Fortunately for us, the supervisor of this project has graciously provided us with an advanced simulation of a UAV that is designed and implemented in MATLAB's Simulink. This model is the

one that used and found on the Gitlab repository for those interested in studying it, but since this model was given, no analysis of the inner workings will be made.

What this model does is that it imitates how the Crazyflie quadcopter would behave if one were to fly it in different patterns. Inside this Simulink model there will be an output of realistic values from the simulated onboard accelerometers and gyroscopes that can be used in the Mahony filter for estimation of its orientation. To be able to compare the accuracy of the created filter, the simulation will also provide the "true" rotation of the quadcopter model. This would then enable a opportunity to tune the filter's parameters to better estimate the simulation.

In figure 2 there is a visual of how the outputs from the model is used when testing the designed filter. The accelerometer output does not come with gravity included, so the always downwards pointing gravity acceleration will have to be transformed to the body frame and then added to the output. Both the sensor signals will then have white noise added to them to better reflect an imperfect real-world situation. These noisy signals are then passed trough the Mahony filter, which will be explained in detail in the Modeling chapter, that then returns a rotation matrix that is the estimation of how the UAV is oriented. The rotation matrix defines how much the body-frame have rotated in comparison to the fixed frame which is the room/earth.

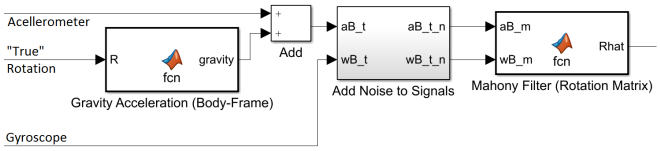


Fig. 2. Coupling scheme of the Mahony filter in Simulink.

This estimated rotation matrix (`Rhat`) is then compared to the "true" rotation matrix (`R`) that comes straight from the Simulink model. The comparison is done in the error metric block, found inside the model, which outputs two different error measurements that are useful.

One is the "Rotational Error" while the other is the 2-norm of the difference between the two matrices. The former is the metric that will be primarily used in the Results sections, as it will provide an easier to understand plot with the measurement being sort of a scale how much off the estimated rotation is compared to the true one. If one were to look at the equation:

$$\texttt{rotErr} = 0.5 \cdot trace(\mathbb{I}_{3\times 3} - \hat{R}^T \cdot R) \qquad (1)$$

the best result would be that the matrix multiplication will result in the identity matrix. This is only true if the estimation and the true rotation are identical, which would then make the whole equation equal 0. The worst that could happen is that the estimation is pointing in the completely opposite direction. This would then result in the equation being equal to 1. An example of how the rotation matrices of a situation like this would look like is presented below:

$$\texttt{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \texttt{Rhat} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Visualized these matrices would point in the direct opposite direction, like this:

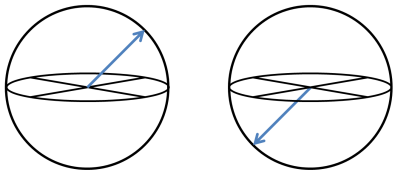


Fig. 3. Visualization of the true rotation to the left and the estimated one on the right. This would yield the result 1 from equation (1).

The scale on how "wrong" the estimation is would then go from 0 to 1, with 0 being the best and 1 the worst as seen in figure 3.

III. Electro-Mechanics

In regard to the electrical and mechanical part, the amount of physical construction will be very limited during this project. However, the Crazyflie does not come assembled, so simple instructions on how to piece it together can be found on the Bitcraze site [5]. But that is about as far as it goes in regard to building something physical, the rest is done in the firmware that is flashed over to the quadcopter. The tools used for this are all found on the same page as the assembly instructions, and from experience the most reliable way of getting this to work is to install everything on a Linux machine. There exist a virtual machine image that can be downloaded that has everything necessary already installed, but there has been trouble getting this to communicate properly with the drone [2].

Nevertheless, something that does relate to the mechanics are the sensors that are mounted on the vehicle. The output from these will be used by the Mahony filter to estimate the attitude, and to our disposal there is an 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer [4]. However, for the basic variant of the Mahnoy filter, which is the one to be used, the magnetometer will be ignored as it is not necessary to achieve a stable flight. The reason for this is explained in the Modeling chapter.

All the onboard sensors are located inside a single compact chip with the model number `MPU-9250`. This can measure a rotational rate change of up to 2000 degrees per second and accelerations up to $16g$, which is way more than will be needed for this project [10]. The electrical signals are translated by the chip to numbers that represent angular velocity in degrees per second for the gyroscopes, or accelerations in m/s^2 for the accelerometers. This means that the outputs from these sensors are directly usable in the code.

IV. Control

There are a couple of different ways of estimating the attitude of an aerial vehicle, but basically everyone uses some kind of accelerometer and a gyroscope working together. This is also the case for the Crazyflie, as it will be its 3-axis accelerometer and 3-axis gyroscope that is used for controlling the aircraft.

Gyroscopes are a type of sensor that perform really well for high frequency variations, i.e. for rotations of the quadcopter, but struggles at measuring steady states. The reason is that gyroscopes drift over time and so does their measurements, which is why these sensors are not ideal to be trusted by themselves [14]. On the other part, accelerometers perform really well when measuring steady states but struggle at keeping track of fast variations in attitude since they are comparably quite noisy [1]. The conclusion here is that neither a pure accelerometer-based or a pure gyroscope-based estimation system can be completely trusted. A different type of sensor will have to be considered, one which can combine the best features of each one of them at their preferred frequencies.

The solution is a type of sensor called, maybe not unsurprisingly, a *fusion* sensor. This type of sensor gathers information from the available sources and combines, predicts and filters their information in order to obtain a much higher quality measure of the actual state of a system [8]. There are many types of fusion sensors, but a few examples are the Non-Linear Complementary filter, the Mahony filter, the Madwick filter, and the Kalman filter.

As stated in the introduction, this report will focus on the Mahony filter, as it is relativly simple in comparison to the Kalman filter, and will provide more than sufficient accuracy in regard to its complexity. The filter will be implemented twice with the first time using calculations with the "standard" rotation matrix for readability and then again using quaternions for a performance increase. The functionality should be identical in regard to the test cases presented here, as the results of the implementations only differ in certain extreme cases that will be explained later.

A. The Mahony Filter

This estimator will read the output data from an accelerometer and a gyroscope and fuse them in a way that ensures accuracy and stability [13]. The estimator is going to rely very heavily on the measurements from the gyroscope when approximating how far the vehicle has rotated since the last measuring point. The values from the accelerometer will be fused with the gyroscope's output signal via a PI-controller in the final step to compensate for gyro drift.

While trying to implement this solution, the first problem is to handle the two different inputs as they are both in different magnitudes and units of measurements. An accelerometer gives values of acceleration (m/s^2) along the three different axis directions the sensor is able to measure, whilst the gyroscope gives values of rotational velocities (rad/s) around the aforementioned axes. In other words, the nature of the two sensors are very different.

Mahony proposed solving this using the gyroscope for deciding any rotations done by the quadcopter, compared to the earth/room it is located in. Gyroscopes are very good at these kinds of measurements, and will provide sufficient estimations for hobbyist use. The problem occurs when the quadcopter hovers, i.e. very low frequency movements. The signal from the gyroscopes will then falsely indicate that the quadcopter is slowly turning, because of the gyro drift, and thereby topple over the aircraft. This is where the accelerometer comes to use.

Gravity is a constant acceleration that affects every object and will be picked up by the accelerometers. This is especially relevant in steady state, when the aircraft is not doing a translation, as the only acceleration actuating on the accelerometer would be gravity. Gravity acts constantly in the same direction with respect to the global frame over time, hence it is possible to estimate the attitude of the quadcopter by studying the decomposed accelerations that the gravity induces on the sensor. By simply stating that the acceleration on the quadcopter should always be straight down (in parallel with the gravitational vector), any deviations from this will be corrected by a PI-controller with the accelerometer as input.

A shortcoming of this approach is that this filter has no way of correctly estimating in what direction the drone is facing. It can only guarantee that the drone is level with the earth so it can hover autonomously. However, in most cases the direction the quadcopter faces is directly observed and controlled by the operator of the drone, and therefore not a deal breaker.

B. Filter Equations – Rotatinal Matrix

The basic formula for the Mahony filter is that there is the reference frame, which is the room, and a body-frame that is attached to the quadcopter. The matrix R is then the rotation matrix defining how the body-frame is oriented in regard to the room. By taking measurements of the changes in rotational velocities from the gyros, the filter will gradually update this matrix as time goes on. As the digital controller isn't continuous, the update formula will contain a sample time, Δt, that is dependant on how fast the processor can perform the calculations. The update formula will therefore be

$$R_t = R_{(t-1)} + \dot{R}_t \cdot \Delta t = R_{(t-1)} + R_{(t-1)}\Omega_\times \cdot \Delta t \qquad (2)$$

with

$$\Omega_\times = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} \qquad (3)$$

where ω is the rotational rate around the axes of the body-frame. Having the rotation velocity multiplied with the sample time will yield a distance, which is then added to the previous rotation matrix to create the current.

In this form the calculations only takes into account the data directly from the gyroscope (rotational rates). Mahony suggested a way to improve this estimation by fusing the

gyroscope measurements with those from the accelerometers via a correction factor:

$$\omega^{'} = \omega + \delta\omega \tag{4}$$

This correction factor, $\delta\omega$, will be calculated by a PI-controller implemented in the estimator like this:

$$\delta\omega = \left(K_p + K_i\frac{1}{s}\right)e = \left(K_p + K_i\Delta t\right)e \tag{5}$$

Here K_p and K_i are parameters to be tuned for optimal performance, and e being an "error vector". This error will be derived from the accelerometer measurements, and what Mahony suggested was that:

$$e = a \times d \tag{6}$$

where a is the normalized values from the accellerometer and d is the gravitational vector transformed to the body-frame of the quadcopter. The cross product of these two vectors will then output the error vector used in Eq. (5).

The way to transform the gravitational acceleration to the local axis is by multiplying the transposed rotation matrix R^T with the vector of gravity accordingly:

$$d = R^T g = R^T \cdot \begin{bmatrix}0 & 0 & -9.816\end{bmatrix}^T. \tag{7}$$

The vector a is a measure of accelerations based on data from the accelerometer in the current sample time frame. The gravity is said to always point straight down in the reference frame (towards the center of the earth). If the vector a is parallel with g (i.e. the quadcopter is not tilted) it will result in the cross product of these being equal to 0. If the two vectors are not parallel, then a value greater than 0 will be achieved which here is used as the error, e, to be corrected.

The tricky part to visualize is that the filter interprets an increased value along the x-axis as a rotation around that axis, i.e. the quadcopter's z- and y-axis are the ones that are changing by increasing x. By taking the cross product of the gravity vector (z-axis) and, for example, a tilted y-axis, it will result in just the x component being grater than 0. This way the accellerations along the axes can be translated to rotations, which is what the controller on the quadcopter understands.

This accelleration deviation is filtered through a PI-controller to determine its aggressiveness in correcting the gyroscopes values. The integral action here is what keeps the gyro drift in check.

This works for rotational corrections around the x and y axes, but there will not be any useful acceleration measurements to stop drifts around z. This was the problem mentioned earlier regarding that the quadcopter can not reliably know which way it is facing solely based on information from these sensors. An extension of the Mahony filter exists which includes the magnetic field from the earth so the vehicle knows which way the magnetic north is. However, since the drone's operator should always be present in these situations, direction is not something that is necessary to compensate for in this case. When observing plots in the Results section, large deviations from the "true" orientation often stems from that the drone does not face the expected way, while the rest of the estimations are actually really good.

An additional limitation that comes with this is that doing loops should be avoided. The filter will try to correct the acceleration vector to be parallel with the gravitational vector. If it is parallel with the quadcopter being right side up or flipped on its back is irrelevant for the calculations, as they result in 0 either way. Since the Crazyflie will not be configured for inverted flight, this situation is best avoided.

For the last step in Mahony's method, the ω in equation (3) will be replaced with this new and improved angular velocity from (4). The update formula for the rotational matrix will then be

$$\dot{R} = R\Omega_{\times}^{'}$$

where the angular velocity tensor $\Omega_{\times}^{'}$ is obtained through the corrected rotational rate vector $\omega^{'}$ as in the expression:

$$\Omega_{\times}^{'} = \begin{pmatrix} 0 & -\omega_z^{'} & \omega_{y}^{'} \\ \omega_z^{'} & 0 & -\omega_x^{'} \\ -\omega_y^{'} & \omega_x^{'} & 0 \end{pmatrix}$$

After this is implemented, an optimal value on K_p and K_i will have to be experimentally obtained. In the firmware source code of the Crazyflie it can be found that suggested values are $K_p = 0.8$ and $K_i = 0.002$ [2].

C. Filter Equations – Quaternions

Before proceeding with the implementation of the Mahony filter in quaternion form, it is recommended to read the appendix descibing in detail what a quaternion is. A very short explanation is that they are basically the imaginary numbers extrapolated into three dimensions, and that way you can describe a direction and a rotation in just four numbers.

Quaternions are preferred for their compactness and robustness when used to calculate rotations. Implementing this in the calculations of the Mahony filter is therefore highly desirable for when the code is to be run on a system with limited computing power. Instead of keeping track of a 3×3 matrix, a 1×4 vector is all that is needed.

Referring back to equation (2), the rotation matrix R is then substituted by the quaternion q, and the update formula will become

$$q_t = q_{(t-1)} + \dot{q}_t \cdot \Delta t \tag{8}$$

where $\dot{q}$ will contain the special quaternion multiplication that can be seen in Eq. (13).

In the following equation, the $\otimes$ symbol signifies quaternion multiplication, while $\cdot$ still has the same effect on vectors as "normal" multiplication.

$$\dot{q} = \frac{1}{2} \cdot q \otimes \omega^{'} = \frac{1}{2} \cdot q \otimes \left(0, \omega_x^{'}, \omega_y^{'}, \omega_z^{'}\right) \tag{9}$$

Here $\omega^{'}$ is the same as the one from (4), but the calculations leading up to $\delta\omega$ will look different. Nevertheless, just like before, the filter begins with a measurement of the gyroscope,

ω, and the normalized values from the accelerometer, a. Then the gravitational vector, transformed to the body-frame of the quadcopter, d is needed. Compared to the previous one in (7), d is not formulated based on the rotational matrix R, but as a function of the body-frame quaternion q.

$$d = \mathrm{Im}\{q^{-1} \cdot g \cdot q\} = \begin{bmatrix} 2(q_x \cdot q_z - q_w \cdot q_y) \\ 2(q_w \cdot q_x + q_y \cdot q_z) \\ q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \tag{10}$$

Refer to (14) to understand the notation of q in the matrix.

Afterwards it only remains to calculate the error vector e. Cross-multiplying the normalized acceleration a with the gravity vector d, as seen from the body-frame, is not affected by any quaternion rule, and will work the same as in (6):

$$e = a \times d$$

After that, the improved velocity estimation $\omega^{'}$ is once again reached via the correction factor $\delta\omega$ with the help of a PI-controller, as seen in (4) and (5):

$$\omega^{'} = \omega + \delta\omega = \omega + (K_p + K_i \Delta t)\, e.$$

And now that q and $\omega^{'}$ are known, the rate of change $\dot{q}$ from Eq. (9) can finally be executed. Therefore, because $\dot{q}$ has been calculated, the recursive update formula for q in Eq. (8) can be applied. However, a final step is required to be carried out on q before it can be used as an updated estimation of the orientation, and that is to normalize it accordinlgy

$$q = \frac{q}{\|q\|}. \tag{11}$$

Why the quaternions needs to be normed is for the reason that an un-normalized one does not really correspond to a rotation in a uniform scale. The equations expect the quaternion to be normalized, and after the calculations are done some floating point rounding errors will have crept into the results. Having this last step prevents these errors from cascading into the next update step [6].

D. Implementing Mahony Filter in C

The mathematical explanations until now have been relatively easy to follow, and they are more or less directly implementable in Matlab since it handles matrix operations in a very simple way as seen by the user. However, the code used to abstract the operations in Matlab is far too computationally intensive to be used in the Crazyflie quadcopter, so the previous operations needs to be translated into C code that can then be compiled and executed on the processor.

This is a common limitation for these small UAVs, and that is also why Mahony and Madgwick filters might be preferred over the Kalman filter. The lines of code needed are far fewer and less computationally expensive. Furthermore, these calculations are able to be made even more efficient by doing them in quaternion form rather than using Euler angles or rotational matrices.

Inside the source folder "`crazyflie-firmware`", from the Bitcraze's git repository, there exist a lot of files for all the different mechanics there needs to be to keep it in the air. Since everything is open source it is possible to rewrite the estimator algorithm to a more preferred one. To enable the use of a custom Mahony filter, the following files needs to be modified:

- `Makefile`
- `stabilizer.c`

and another two files needs to be created:

- `group_e_filter.h`
- `group_e_filter.c`

The contents of all these files are found inside the `crazyflie-firmware-modified` folder on the GitLab repository linked in the title of this report. If you would like to try to build this firmware yourself, please refer to the `README` inside the same folder.

Nevertheless, what is happening is that instead of calling the Kalman filter, the code from `stabilizer.c` calls the Mahony filter that is inside `group_e_filter.c`. This filter then does the quaternion calculations explained in the previous chapter, and outputs an updated rotation estimation for the onboard controller to use. This controller is what keeps the drone in the air, and it has not been modified in any way.

If the source code is to be studied the different steps are clearly named so their corresponding step here in the report can be found. What makes the code hard to read is that almost every step is broken down to the most simple of operations ($+$,$-$,$\cdot$,$/$), and every element of vectors have to be treated separately. This results in a lot to read, but is much faster computationally wise.

A little magic trick that is done to speed up the normalization step can be found at the bottom of `group_e_filter.c`. It is called "the fast inverse square root" and can be done because the code is written using 32-bit floating-point numbers in the calculations. Here no division or square root operations are made, only a bit-shift with the number `0x5f3759df`. This is much faster while only yielding an error that is lower than 0.175%. For the interested reader it might be useful to look up why this work in the referenced bibliography [7].

V. Results

This section will begin with a study of how the filter created in Simulink reacts to inputs, to then inspect how it holds up against the Kalman filter in a real-world test.

A. Simulink Simulation

As stated in the Modeling chapter, the equation used to obtain the error graph is the one found in equation (1).

The simulation is going to be moving the quadcopter in a bit of a jerky manner around the origin, and after 10 seconds return to the starting position completely level with the floor and remain stationary. In the best case estimations the plot should always stay at 0, as that signifies perfect estimation.

If the simulation is run for 20 seconds, the plot in figure 4 can be obtained. Here the Mahony filter is used with the

weights $K_p = 0.8$ and $K_i = 0.002$, and the plot will show a measurement on how much deviation from the true orientation the estimation is.

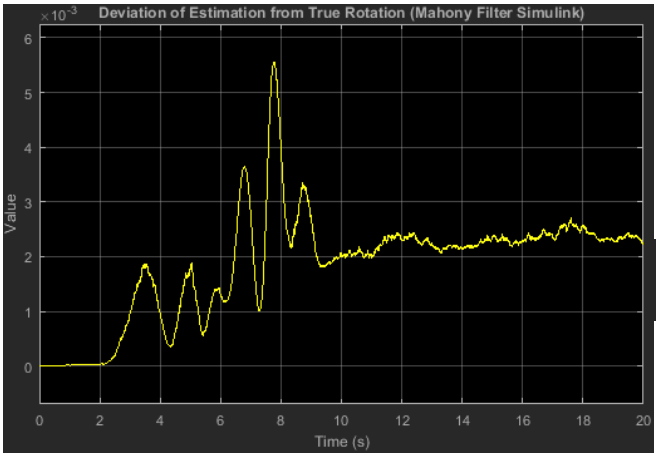


Fig. 4. The rotational error (from 0 to 1) between the estimated and the true rotational matrices. This is the deviation if the Mahony filter is used with the weights $K_p = 0.8$ and $K_i = 0.002$.

Please note that the scale on the y-axis is 10^{-3}, so the largest spike reaches just beyond 0.005 difference. At the end you can clearly see that there is still a stationary error that grows slightly over time, even though there is drift compensation. However, this plot does not tell the whole story. If the rotational matrix were to be decomposed into its three axis components, the actual directions where the deviations occur would become visible.

As stated in the Modeling chapter, there is no way of this filter to compensate for drifts around the z-axis (which way the UAV is looking) as there is not enough information regarding those translations. The only thing the filter can do is to make sure that the drone is hovering stably parallel to the ground. The remaining error seen here therefore stems from the drone not facing the way expected.

B. Real-World Test

Moving on to the real test, where the firmware of the Crazyflie is updated with a new estimator, the following plots can be obtained.

What is visible in figure 5 is the value of the four separate components of the quaternion that is the the estimated rotation. Because of the double column format of this report, the size of this graph is a bit small and a bit hard to see. However, these graphs can all be loaded in Matlab by running the "create-plots" script found in the `matlab-plots` folder.

Additionally, a two dimensional top-down overview of the drone's flight path can be seen in figure 6. The test begins at rest in position $[1.2, 3]$, for the quadcopter to then quickly rise into a position where it hovers for a second. It then moves back and forth between two spots in a straight line three times, to then move upward in a spiral. This is the sinusoidal pattern that can be seen in figure 5. At the very end it does a hard landing in $[-0.2, 4]$, which is why the quaternion plot is very messy in the last bit.

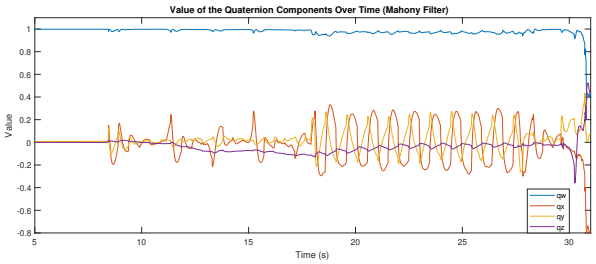


Fig. 5. Components of the quaternion used in estimation of the quadcopter's attitude. These values are extracted from the Crazyflie's log file.

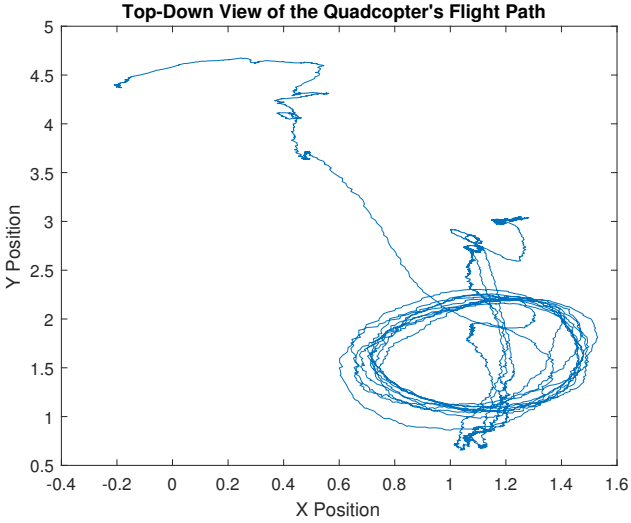


Fig. 6. Top-down overview of the drone's flight path in the real-world test.

However, what is more interesting is how the Mahony filter compares to the Kalman filter that is usually used on this. In figure 7 each component of the quaternion is split up into its own graph. The blue line is the estimation done by the Kalman filter while the red one is the Mahony one, and the closer they are the better.

As can be seen they are very similar and, as it is set up now, the Kalman filter will have to be seen as the closest to the truth.

VI. Discussion

In regard to the complexity of the filter mathematics, Mahony is much simpler than Kalman. However, the processor on the Crazyflie is actually powerful enough to run both these filters in parallel which is why the comparison plots were able to be obtained. The mentioning regarding the computational limitations does not hold true in the case of the Crazyflie, but there are smaller and cheaper aircraft out there that do have this limitation. Analyzing the possibility for simpler filters allows the use of cheaper hardware, which is usually good.

Commenting on the data in figure 7, the plot created by the Kalman filter should be seen as closer to the truth. However,

the Mahony filter obviously does a fine job at keeping the estimation close to the more advanced filter. This is also backed up by the fact that the simulated error in figure 4 is very small.

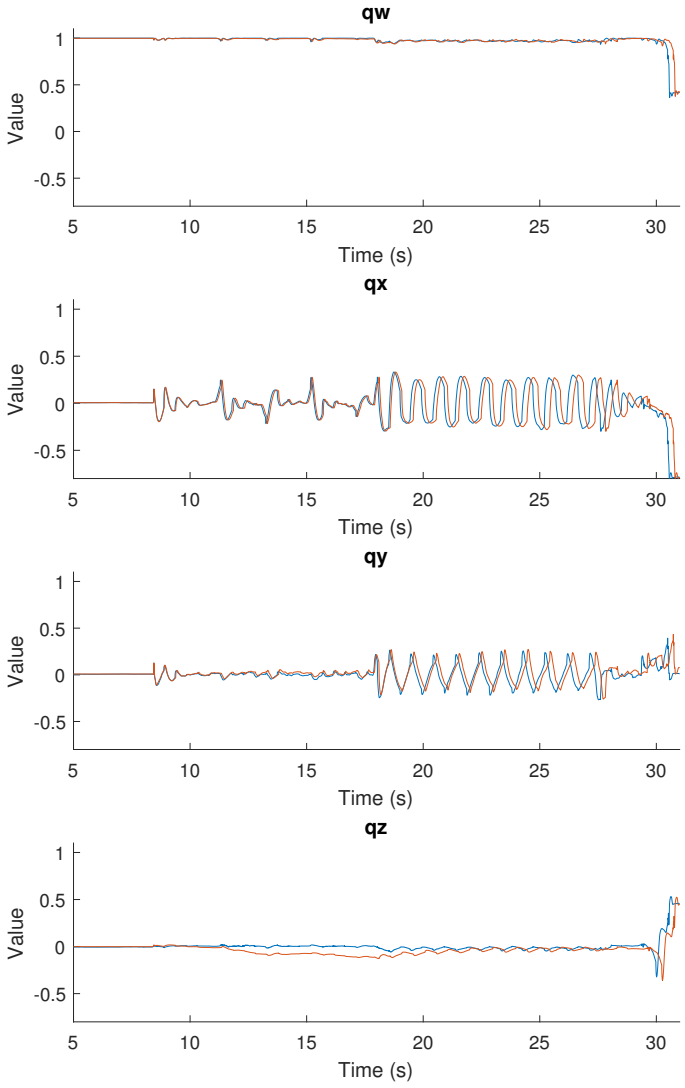


Fig. 7. Components of the quaternion used in estimation of the quadcopter's attitude. The blue line is the estimation done by the Kalman filter, while the red line is the estimation done by the Mahony filter. They are plotted in the same graph to show the differences.

What is a problem is the lack of information to compensate for drifts around the z-axis. A further expansion of the Mahony filter is possible, where the inclusion of data from the onboard magnetometer can be used to identify the magnetic north. This way the remaining errors could be minimized as well.

This would be something interesting to look at in a later project, as well as finding out how this filter would stand up against its main competitor: the Madgwick filter. They both have about the same computationally complexity since their structure is very similar. Their main point of difference is how they calculate the "error vector" in Eq. (6). How this translate to real world differences, and finding out which is more accurate could be a good idea for next years students.

References

[1] Difference Between. *Difference between Gyroscope and Accelerometer*. 2017. URL: http://www.differencebetween.info/difference-between-gyroscope-and-accelerometer.

[2] Bitcraze. *Bitcraze Virtual Machine*. Mar. 2017. URL: https://wiki.bitcraze.io/projects:virtualmachine:index.

[3] Bitcraze. *Crazyflie 2.0*. July 2017. URL: https://www.bitcraze.io/crazyflie-2/.

[4] Bitcraze. *Crazyflie KIT electronics explained*. July 2015. URL: https://wiki.bitcraze.io/projects:crazyflie2:hardware:specification.

[5] Bitcraze. *Getting started with the Crazyflie 2.0*. Aug. 2017. URL: https://www.bitcraze.io/getting-started-with-the-crazyflie-2-0/.

[6] Eric Brown. *Quaternions: How*. Feb. 2010. URL: https://physicsforgames.blogspot.se/2010/02/quaternions.html.

[7] Wikipedia contributors. *Fast inverse square root — Wikipedia, The Free Encyclopedia*. [Online; accessed 19-December-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Fast_inverse_square_root&oldid=811298679.

[8] J. L. Crowley and Y. Demazeau. *Principles and Techniques for Sensor Data Fusion*. May 1993. URL: http://www-prima.inrialpes.fr/Prima/jlc/papers/SigProc-Fusion.pdf.

[9] Emil Fresk and George Nikolakopoulos. *Quaternion Based Attitude Control for a Quadrotor*. July 2013. URL: http://folk.ntnu.no/skoge/prost/proceedings/ecc-2013/data/papers/0927.pdf.

[10] InvenSense Inc. *MPU-9250*. June 2016. URL: http://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf.

[11] Sebastian O.H. Madgwick. *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*. Apr. 2010. URL: http://x-io.co.uk/res/doc/madgwick_internal_report.pdf.

[12] Olliw.eu. *IMU Data Fusing: Complementary, Kalman, and Mahony Filter*. Sept. 2013. URL: http://www.olliw.eu/2013/imu-data-fusing/.

[13] Jean-Michel Pflimlin. R. Mahony Tarek Hamel. *Nonlinear Complementary Filters on the Special Orthogonal Group*. June 2010. URL: https://hal.archives-ouvertes.fr/hal-00488376/document.

[14] Wikipedia. *Inertial measurement unit — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-November-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Inertial_measurement_unit&oldid=810158293.

APPENDIX

Quaternions and Spatial Rotations

Quaternions are a number system that extends the complex numbers and they were first described by Irish mathematician William Rowan Hamilton in 1843 and applied to mechanics in three-dimensional space.

To get a grasp regarding what is going on a little bit faster, a comparison to the more familiar *complex numbers* can be made. They are defined by introducing an abstract symbol, i, which satisfies all the usual rules of algebra but with the addition of the rule that $i^2 = -1$. With this additional rule, directions in 2 dimensions can be described through numbers containing a real and an imaginary part.

In the same way the quaternions can be defined by introducing the abstract symbols i, j, k which satisfy the rules $i^2 = j^2 = k^2 = ijk = -1$. These would then be able to describe directions in three dimensions. The quaternions also follow the usual algebraic rules but with the important exception of the commutative law of multiplication. What this means is that the order of operations matter when dealing with quaternions, i.e. $ij = k$, while $ji = -k$. This property will prove to be very useful in later calculations.

Quaternions are generally represented in the form:

$$q = a + bi + cj + dk = (a, b, c, d)$$

where a, b, c, d are real numbers, and i, j, k are the fundamental quaternion units previously mentioned. They form a four-dimensional associative normed division algebra over the real numbers, and therefore also a domain denoted by $\mathbb{H}$ (for Hamilton):

$$\mathbb{H} = \{(a, b, c, d) \mid a, b, c, d \in \mathbb{R}\}.$$

Since quaternions are vectors of dimension 4, basic operations such as addition and multiplication becomes more complicated, and custom algorithms are necessary for the calculations to be correct. Below are two examples of operations that will be used in the estimator algorithm (i.e. addition and multiplication).

$$\begin{aligned} q_1 + q_2 &= (a_1, b_1, c_1, d_1) + (a_2, b_2, c_2, d_2) \\ &= (a_1 + a_2, b_1 + b_2, c_1 + c_2, d_1 + d_2) \end{aligned} \tag{12}$$

$$\begin{aligned} q_1 \otimes q_2 &= (a_1, b_1, c_1, d_1) \cdot (a_2, b_2, c_2, d_2) \\ &= (a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2, \\ &\quad a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2, \\ &\quad a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2, \\ &\quad a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2) \end{aligned} \tag{13}$$

These might seem complicated, but they actually streamline the calculations that are necessary to do when estimating rotations. Please also note the use of the symbol $\otimes$ here to signify that quaternion multiplication differs from the normal vector multiplication $(\cdot)$.

In 3-dimensional space, according to Euler's Rotation Theorem, any rotation or sequence of rotations of a rigid body or coordinate system about a fixed point is equivalent to a single rotation by a given angle θ about a fixed axis (called the Euler axis) that runs through the fixed point. The Euler axis is typically represented by a unit vector $\vec{n}$. Therefore, any rotation in three dimensions can be represented as a combination of a unit vector $\vec{n}$ and a scalar θ.

Quaternions give us a simple way to encode this axis–angle representation in four numbers, and can be used to apply the corresponding rotation to a position vector, representing a point relative to the origin in $\mathbb{R}^3$. In this case, the scalar would be the rotation w of the body, and the vector would be the axis on which the body rotates. Therefore, the quadcopter's movement can be represented in quaternion form as

$$q = (w, \vec{n}) = (w, x, y, z) = w + xi + yj + zk \tag{14}$$

with $q \in \mathbb{H}$, $w \in \mathbb{R}$ and $\vec{n} \in \mathbb{R}^3$. Then the formulas for the addition and the multiplication can be written more compact like this:

$$q_1 + q_2 = (w_1, \vec{n}_1) + (w_2, \vec{n}_2) = (w_1 + w_2, \vec{n}_1 + \vec{n}_2)$$

$$\begin{aligned} q_1 \otimes q_2 &= (w_1, \vec{n}_1) \cdot (w_2, \vec{n}_2) \\ &= (w_1w_2 - \vec{n}_1 \cdot \vec{n}_2, w_1\vec{n}_2 + w_2\vec{n}_1 + \vec{n}_1 \times \vec{n}_2) \end{aligned}$$

where $\cdot$ is the dot product and $\times$ is the normal cross product.

Quaternions, compared to Euler angles, are simpler to compose and have the benefit of not being affected by the "gimbal lock" problem. Gimbal lock is the loss of one degree of freedom in a three-dimensional, three-gimbal mechanism that occurs when the axes of two of the three gimbals are driven into a parallel configuration. This "locks" the system into rotation in a degenerate two-dimensional space. In simpler terms this means that there exist some extreme cases where two rotational axes become aligned, and the calculations will not know how to rotate the quadcopter to get away from this position.

The cause of gimbal lock is representing an orientation as three axial rotations with Euler angles. A potential solution therefore is to represent the orientation in some other way. This could be as a rotation matrix, a quaternion, or a similar orientation representation that treats the orientation as a value rather than three separate and related values.

Compared to rotation matrices quaternions are more compact, more numerically stable, and may be more efficient. Furthermore, for a given axis and angle, one can easily construct the corresponding quaternion, and conversely, for a given quaternion one can easily read off the axis and the angle. Both of these are much harder with matrices or Euler angles.

A quaternion rotation $p' = qpq^{-1}$ (with $q = a+bi+cj+dk$) can be expressed as a matrix rotation $p' = Rp$, where R is the rotation matrix given by:

$$R = \begin{pmatrix} 1 - 2s(c^2 + d^2) & 2s(bc - da) & 2s(bd + ca) \\ 2s(bc + da) & 1 - 2s(b^2 + d^2) & 2s(cd - ba) \\ 2s(bd - ca) & 2s(cd + ba) & 1 - 2s(b^2 + c^2) \end{pmatrix}$$

where $s = \|q\|^{-2}$ and if q is a unit quaternion, $s = 1$.

The axis $\vec{n}$ and the angle θ corresponding to a quaternion $q = a + bi + cj + dk$ can be extracted via:

$$\vec{n} = (x, y, z) = \frac{(b, c, d)}{\sqrt{b^2 + c^2 + d^2}} \tag{15}$$

$$\begin{aligned} \theta &= \arccos \frac{a}{|q|} = \arccos \frac{a}{\sqrt{a^2 + b^2 + c^2 + d^2}} \\ &= \arcsin \frac{|\vec{n}|}{|q|} = \arcsin \frac{\sqrt{b^2 + c^2 + d^2}}{\sqrt{a^2 + b^2 + c^2 + d^2}} \end{aligned} \tag{16}$$

Quadcopter with Artificial Intelligence

Andreas Abramsson*, Emil Wåraeus†, Rijad Alisic‡ and Siriffo Sonko§
Lund University
*mas12aab@student.lu.se, †fys12ewa@student.lu.se, ‡rijad.alisic.710@student.lu.se, §mat11sso@student.lu.se
https://gitlab.control.lth.se/regler/FRTN40/2017/group-F

***Abstract*—This is a report of a project in the course Projects in Automatic Control at Lund University. It builds on a previous project in the course Real Time Systems where drone position control was implemented. The goal was to make a small drone follow a person using an on-board camera. The camera sent a video feed to an AI via radio communication which classified the images, detected if there were humans in them and specified where they were. The relative position of the person was then used as input for a controller which calculated a control signal for the drone. The project was mostly a success as the controller received positional values that should be good enough for the controller. Due to lack of time only one dimension, height, was implemented and had issues with oscillation.**

I. Introduction

The first good face-detection algorithm was implemented by Paul Viola and Michael Jones, named the Viola-Jones Algorithm, was created in 2001 [26]. This used hand-coded features fed into a support vector machine (SVM). A similar algorithm was invented in 2005 which used the Histograms of Oriented Gradients (HOG) technique and was invented by Navneet Dalal and Bill Triggs [10]. The HOG-features are compared with some reference image features to determine if the image is of some specific class or not. In 2012, the era of deep learning started and the ImageNet competition was won with a Convolutional Neural Network (CNN) [18]. At this time CNN was mostly used for classification, not detection. By 2015, the R-CNN was invented by Professor Jitendra Malik at UC Berkley, which used region proposals and a CNN to do object detection (detecting the position as well as the class of some object in an image) [14]. and was way faster then just sliding a CNN over the whole image. This technique has been iterated upon with Fast R-CNN [13], Faster R-CNN [24] and the recent Mask R-CNN [15]. In 2017, all these technique were outperformed by a YOLO-net (You Only Look Once) [23], if time to run is taken into account. More specifically, this was achieved with the second iteration of YOLO (YOLOv2) [22]. YOLO segments the image in a grid where each cell in the grid predicts possible partial bounding boxes. YOLO also gives a confidence of each class in the bounding box. So this network handles all tasks that were previously separated into different algorithms, which contributes to YOLO's efficiency.

The results of this project could be useful in several applications. For instance it could help enabling safe human-robot interactions or identification of humans in autonomous search and rescue missions. It could also be a cheaper option for surveillance of large areas where it could replace multiple security cameras. Another use is within the media industry, where an autonomous drone could be safer, give higher quality video shots and be cheaper for certain types of scenes. This project is a continuation of a previous one in the course Real Time Systems, were the goal was to implement a position regulation algorithm on the Crazyflie platform [25]. A picture of the final quadcopter is seen in Figure 1. Code from that project was used for example in the GUI and the control

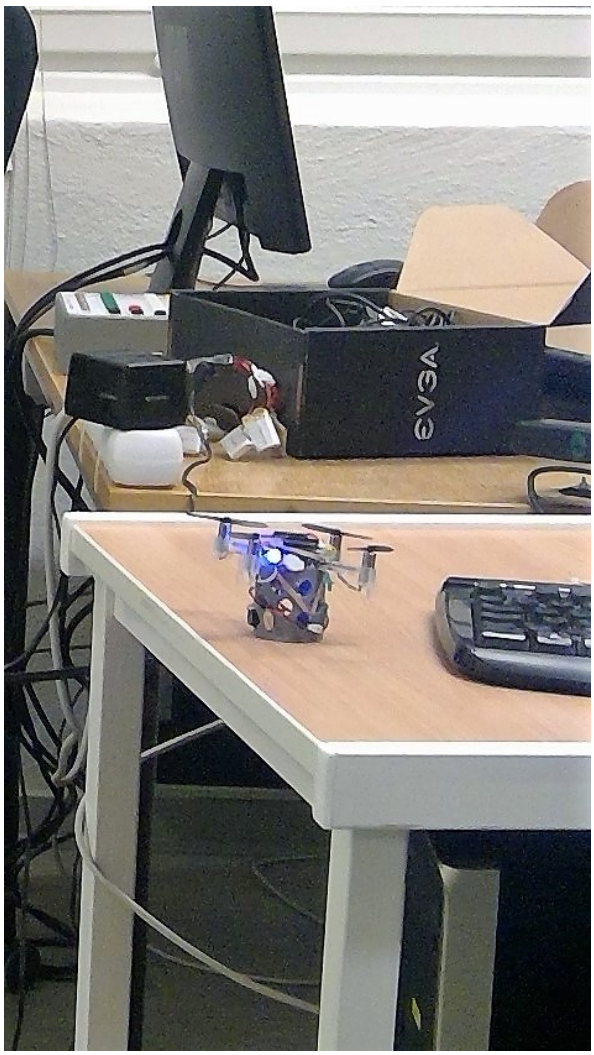


Fig. 1. The figure shows the final Crazyflie 2.0 quadcopter, outfitted with both structure that both holds the camera in place, and protects it from breaking when colliding.

system. The goals of the project can be divided into several milestones:

1) Get the chosen neural network up and running on the graphics card.
2) Ensure that the neural network can provide a real-time stream of classified images using a demo camera.
3) Get the drone to fly with a person controlling its movements.
4) Get the radio communication from the camera to the computer to work smoothly.
5) Classify images from the wireless camera.
6) Control the drone using data from the AI.

II. Implementation and Design

A. Deep Learning

To successfully make a drone follow a person the drone needs to know where the person-objects are. This can be done through object classification and tracking. Deep learning is an excellent tool to create a person-tracking algorithm, and there are a number of different network architectures that could be suitable for this task. The ones worth considering are Faster R-CNN [24] and YOLOv2 [22]. Both of these networks are quite new and have been proven to be reliable and fast. The real-time aspects of the network is a very important feature, as the classification needs to run at a sufficient speed to update the relative position of the drone to the tracked person.

The group decided to use the YOLOv2 network created by Joseph Redmon, Santosh Divvala Ross Girshick and Ali Farhadi [22], which has similar performance to other state of the art networks, such as Faster R-CNN Inception Resnet v2 [17], but with a significant speed increase. The speed of YOLOv2 made the network suitable for this project, as hardware was limited to a midrange Graphics Processing Unit (GPU) (Nvidia GTX 1060). The YOLO network comes in two different sizes, YOLO and Tiny-YOLO. Tiny-YOLO is significantly faster and can run at up to 200 Hz on a decent GPU. But this comes at a great cost in accuracy. After testing both these networks, normal YOLO was deemed to be the best network for the task, running at a sufficient 13 Hz on the given GPU. Worth mentioning about Tiny-YOLO is that it ran at 5 Hz on a CPU, which might be sufficient on an embedded platform or Raspberry Pi.

YOLOv2 is written in Darknet, which is a C-based deep learning framework. This is not consistent for the project, as everything else is implemented in Python. Therefore the project group made use of YAD2K [4], which is a translator of YOLOv2 to Python using Tensorflow and Keras [1], with a bare-bones implementation overview seen in Appendix **??**.

YOLO is a fully Convolutional Neural Network which benchmarks at 78,6 % mAP on VOC 2007 [16], compared to SSD500 (which is another fast tracking network) which comes in at 76,8 % [27]. mAP is the unit which performance is measured in for object detection, and stands for mean average precision.

One of the first iterations of the Neural Networks were the *Fully Connected Networks* [19]. These networks are typically good at learning non-linear functions of higher degree. But these networks are not very good at image classification or object tracking. As shown in Figure 2, the fully connected network has all of its neurons in layer *n* connected to layer *n+1*. This is not a very efficient way to approximate functions, as a large part of the network tends to be unused or *dead*.

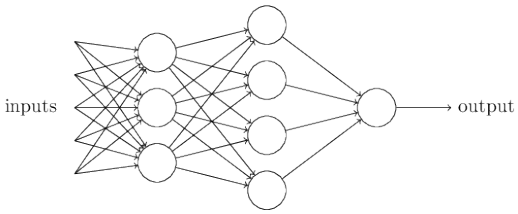


Fig. 2. The figure shows a fully Connected Neural Network, where the circles are the ”neurons” and the arrows are weights. Every neuron in each layer takes the weighted average of the previous layer of neurons.

Convolutional Neural Networks make use of weight-sharing to increase the efficiency of the network. This means that instead of having all neurons in layer *n* connected to all neurons in layer *n+1*, the convolution makes use of a *patch, stride* and *filters* to achieve this efficiency. Convolutional Neural networks can be described as shining a flashlight on the previous layer, and looking only at the part where the light shines on that layer, the *patch*. Then one moves the the flashlight a certain number of neurons/pixels, *stride*, in horizontal and vertical steps until the whole picture has been examined. The same weights are used in every step, and the network is therefore weight-sharing. Convolutional layers also have a depth, or a certain number of *filters*. Each filter does not share weights with the other filters. This means that different filters can look at different features in the previous layer. Typical convolutional networks might range from 1 to over 100 layers deep. But for classification and tracking tasks they are typically around 10-30 layers deep. A convolution layer example is shown in Figure 3.

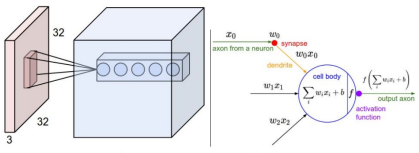


Fig. 3. The figure shows a convolution layer in a CNN. The convolution is done on a layer that is 3 channels deep (e.g., a RGB image). One may see that the resulting layer has a longer channel dimension. Each entry on a certain channel correspond to a specific filter applied on a certain part of the previous layer. [9]

During testing, the video from the wireless camera appeared very noisy. An approach to denoising images was made with a neural network architecture called an *autoencoder*. This consists of an encoder and a decoder. The encoder is a convolutional neural network that compresses the image into a tensor with a smaller volume then the input image. A decoder is then used to uncompress the tensor into an image of same

size as the input. The decoder consists of convolutions and upsamples. The autoencoder was then trained as a network on images with artificially added noise and the output was set to be the image without noise. The artificial noise was recorded from the wireless camera. The principles of an autoencoder can be seen in Figure 4 and Figure 5.

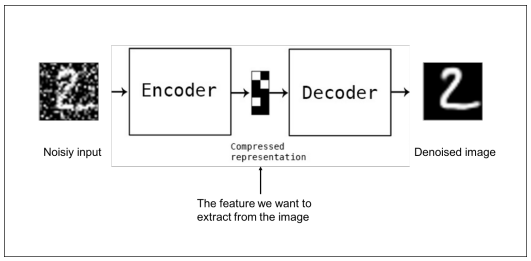


Fig. 4. The figure shows an autoencoder denoising on hand written digits from the MNIST dataset. One may see that the image is compressed into a smaller, feature tensor. Thus it is trained to only save relevant information, and discard the rest. The decoder is trained to restore the image from the feature tensor.[8]

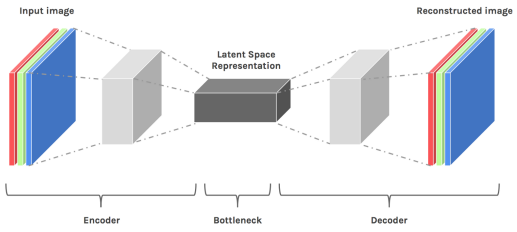


Fig. 5. The figure shows a convolutional Autoencoder. Here, the encoding layers use convolutional layers to extract and compress the image into a feature tensor. The decompressing layer then uses convolutional layers and upsampling to combine the features back into an image. [6]

B. Communications Design

1) Crazyflie to computer: A small camera with radio transmitter [12] was mounted on the Crazyflie [25]. The camera took analogue pictures and sent them as an analog stream. A radio receiver [7] received the image data and passed it on to a digitizer[11] that converted the analog stream of pictures from the camera to a digital stream. A stationary camera [5] was initially used to find the position of a person in order to test the YOLO network before the work continued on with the small camera [12] that was mounted on the drone.

2) Computer to Crazyflie: The Crazyflie used the Crazyradio PA, which is a long range open USB radio dongle, to communicate with the computer. It is based on nRF24LU1+ [20] from Nordic Semiconductor and a 20 dBm power amplifier. It was programmed with firmware for the Crazyflie and had a range of 2 km together with the Crazyflie 2.0. The camera transmitted over the 5.8 GHz frequency band. This band is part of the 802.11n standard and is widely used in homes and in consumer electronics. The camera has four different bands to choose from on the 5.8 GHz band and each band has eight different options. All these frequencies can be seen in the tableI. The BOSCAM FPV Wireless RC305 receiver also works over the 5.8 GHz band but only has eight channels. The chosen band was limited according to which band the BOSCAM could use. The entire communication path between the various hardware that was used is shown in Figure 6.

TABLE I
THE TABLE SHOWS THE FREQUENCY BANDS FOR CAMERA MOUNTED ON DRONE.

Frequency Table								
Frequency Band A	5865	5845	5825	5805	5785	5765	5745	5725
Frequency Band B	5733	5752	5711	5790	5809	5828	5847	5866
Frequency Band E	5705	5685	5665	5645	5885	5905	5925	5945
Frequency Band F	5740	5760	5780	5800	5820	5840	5860	5880
Frequency Band R	5658	5695	5732	5769	5806	5843	5880	5917

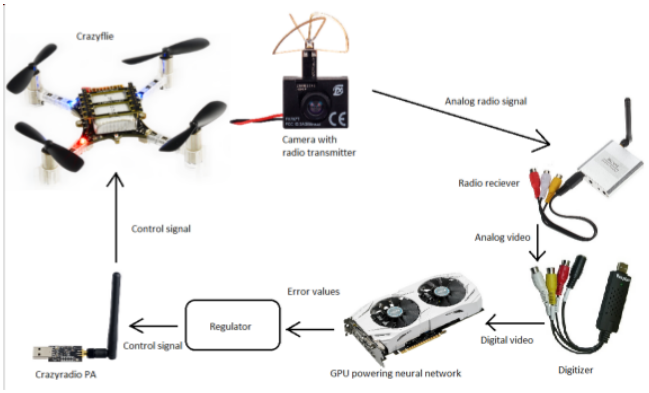


Fig. 6. The figure shows the Various hardware used for the project. The flow of information from the various pieces is also shown.

C. Software Design

A project of this type, where different components with different software basis need to be merged together requires a structure that allows for intercommunication between different software modules. An almost natural choice was to use Python 2, due to its flexibility between different operating systems. As a consequence of this, Python's flexibility allowed parallelism for various parts of the software to be developed and tested on Ubuntu 16.04, Ubuntu 14.04 and Windows 10.

1) Robot Operating System (ROS): Another reason for choosing Python 2 was its integration with ROS. ROS is a set of software libraries that allow the user to create a messaging network for real-time implementations and is for this reason the backbone of this project. One is able to create a network of nodes that in turn manage topics. These topics effectively work like mailboxes, where each node can subscribe to a couple of mailboxes and receive the information stored within them when a publisher of the topic publishes new information to them. Since multiple nodes can subscribe to a topic and each such node receives the message when it is published to the topic, the nodes can receive messages that are accessed without having to worry about mutual exclusion aspects.

Another type of messaging that is used in ROS is called *services*. Services work similarly to topics, but instead of passing one-way messages from many publishers to many subscribers, this construct works on a one-to-one basis with a two-way message path. A node sends a request message to a service and waits for the owner of the service to respond with a message. ROS's framework is thus ideal for communication between the camera, GUI, AI, regulator and Crazyflie. A representation of how ROS sees these components is shown in Figure 7.

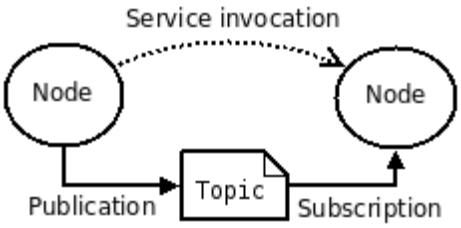


Fig. 7. The figure shows a schedule of ROS graph concepts. The solid lines are one-way one-to-many message path, whereas the dashed line represent a two-way one-to-one message path.

2) GUI: The graphical user interface was built with the *Tkinter* module for building the window. As mentioned previously, the GUI could contain a ROS node through which it communicated with the outside world. The node subscribed to two topics, one of which published the output boxes from the YOLOv2 network imprinted on the original image and the second topic it subscribed to published the position from the state estimator which also had the output from the YOLOv2 network as input.

3) Regulator: Another thing that the GUI script started was the regulator. The regulator was initialized in a different thread in order to ensure that the quadcopter input signal change could be applied independently of the internal GUI update time or some other process that could halt the update input signal. The regulator instance that was created continuously ran the PID and PD controllers, moving the quadcopter to the reference point in space by sending the control signals directly to the crazyflie through the Crazyradio PA.

4) Testing modules: While the aforementioned implementation was used for the final product, there were some additional hardware and software modules that were used during development. The first one was that the AI was initially implemented for an ASUS Xtion camera. It used a ROS node to publish its image matrices through the *openni/kinect_vision* driver. The change in code in order to use the small radio transmitting camera did not require any major changes. All one had to do was to change the input channel from a ROS node to a frame grabber implemented through the *pygame* module.

In order to test the input signals to the crazyflie, the *joy* drivers was used to map the inputs from a Xbox 360 controller to a ROS node that publishes to a topic. This topic is subscribed to by a submodule in the regulator thread. The regulator thread then used this as input signals to the crazyflie instead of using the PID controller input.

III. Control

A. Drone Control

There was an attitude controller on the embedded system in the drone that controlled the speed of the motors and kept the drone somewhat stable. The outer controller that will be described here focused only on the positioning of the drone. The output of this outer system was a vector with four commands: *roll*, *pitch*, *thrust* and *yawrate*. The inner control loop then handled the control of the actual actuators. The complete system can be seen in 8.

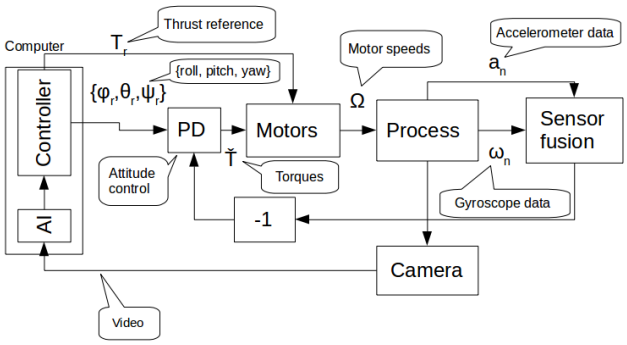


Fig. 8. The figure shows a schematic over the entire control system.

The purpose of the project was to have the drone follow a person. When a person had been detected, its position in the image was converted into positional error values which were to be minimized. There were three separate PID-controllers for each positional dimension. An error in the y-direction (height) corresponded to an increased or decreased thrust. An error in the x-direction (sideways) on the other hand affected the roll rate, meaning that the drone rotated in order to keep the person centered in its view (instead of moving sideways by rolling as would be the other option). The final dimension, the z-dimension or depth corresponded to an error in distance to the target person and thus controlled the pitch of the drone.

A person's position in the image was given by the upper left and lower right corner of a bounding box surrounding them (in pixel coordinates). Finding the error in the x- and y-directions is thus pretty straightforward. By defining the center of the box as the center of the person, the error values were given as distance in pixels from the center of the image (or some other arbitrary point). The error in z-direction on the other hand was be a little bit trickier. It could be defined as the size of the bounding box in either one or two dimensions, but then the distance would vary for people of different size. A person could also appear larger by spreading their arms for example. It was the method that was tested however, before considering other options, which would be more precise but would also require more work.

B. Yolo Theory

YOLOv2 consists of 23 convolutional layers with batch normalization, maxpooling and Leaky Relu activations. In

table **??** one can see an overview of the used architecture. Note that the networks split into a parallel network from leaky_re_lu_13 until concatenate_1.

This version of YOLO produces an output-tensor with the shape of 19x19x425. This prediction reflects the 19 by 19 grid cells the input image is segmented into. Bounding boxes and classes are predicted in each grid cell, which are put through a threshold to determine which predictions that are objects and not. This is visualized in Figure 9 (but with 7 by 7 cells instead of 19 by 19).

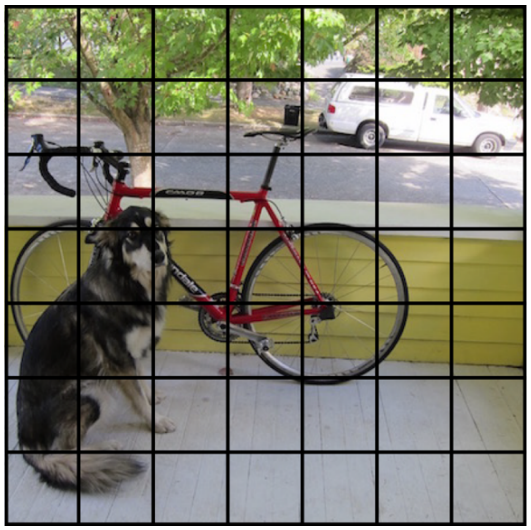


Fig. 9. The figure shows an example of grid cells, similar to the ones used by the YOLO network when deciding bounding boxes.[23]

Each filter in the output-tensor makes a certain part of the prediction. Every prediction consists of 5 filters + the number of labels. These 5 filters predicts the probability of an object being present, x, y, width and height. In this version of YOLO a total of 5 different objects can be predicted in each grid-cell. Therefore the depth of the output tensor is $5 * (5 + 80) = 425$ [23]. This is visualized in figure 10

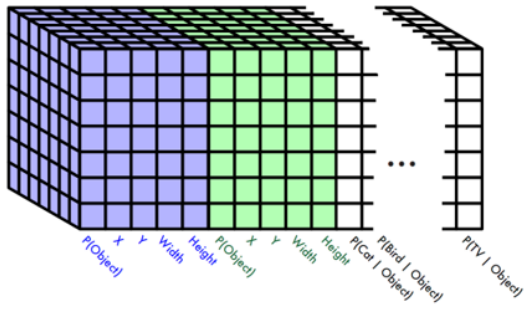


Fig. 10. The figure shows the output tensor from YOLO. Here, one may see the 5 (colored) filters determining if and where a general object is. In the last filters (white cells), one may see the probability of each of the cells belonging to a certain class. [21]

IV. Results

A. Object detection

The results of using YOLOv2 as an object detection algorithm showed promising results. While testing it using both the ASUS camera and the camera onboard a Surface Pro, it had a very high detection rate and certainty. An example of this is shown in Figure 11. The object detection tried to incorporate the entire human within the bounding box, resulting in a larger box if e.g., the person being detected has their arms extended. The only issue is that it would define other objects as persons every now and then. While the wireless camera showed very promising initial results, it became apparent that it was just a lucky time window that it was tested on. In general, the images were very susceptible to noise from all the surrounding electronics. The YOLOv2 network was unable to detect objects in some workplaces.

Fig. 11. The figure shows the classification done by the YOLO9000 network using an ASUS Xtion camera. The bounding boxes show where the persons are and the text above them shows what type of object they are and the corresponding certainty.

B. Camera

The image quality was not good enough for the network to be reliable and we believe this is partly due to noisy environment but also partly due to non compatible hardware, since some improvements were made by changing the environment and the picture did not become worse when removing the antennae on both the camera and the receiver. The specs should be good enough but the image quality still isn't great. Not only were there noise in the image, but there noise also distorted the image in the vicinity of the discolored pixels. One may see this effect clearly in [2].

C. Autoencoder

However, due to the noisy image received from the wireless camera, an autoencoder was created to remove the noise. The resulting autoencoder, which was trained on noise obtained from the wireless camera, see Figure 12, was very effective at removing the additive noise. However, when testing the autoencoder filtering on the wireless camera feed, it was clear that the noise was not only additive, but there were some spatial distortions that came along with it. The resulting image

was nonetheless clear of any discoloring noise, which was most obvious when the camera stream recorded dark images.

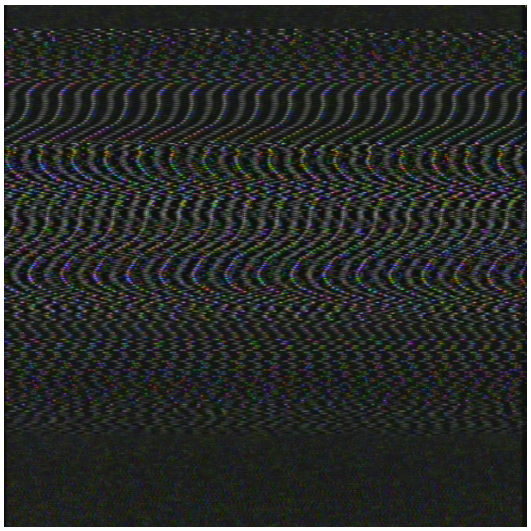


Fig. 12. The figure shows the additive noise which was extracted from the wireless camera.

Fig. 13. The figure shows the result from the trained autoencoder on additive noise. Original images taken from the BioID dataset.

D. Controllers

The performance of the PID controller that was designed to control the thrust, is shown in [2]. One may see that the detection is missing for some short time periods and that the crazyflie shows some very oscillatory behavior. One may also see that it is easy for the controller to steer the crazyflie up to its preferred reference value, but when a decrease in height is needed, the motors shut of and it is difficult for the drone to break the fall and start rising up again, which is also seen in [3].

V. Discussion

The project went on smoothly and was on schedule, much due to the great work during the planning phase. The delivery of the camera components was a bit delayed but during that time work could still proceed on the neural network as we had the GPU up and running from the start and could test it using a ASUS Xtion camera.

The wireless camera was one of the biggest uncertainties of the project beforehand as it depends on three separate components from different manufacturers. It proved to be easy to get the camera to send a video stream to a computer although problems arose when the images was to be imported into the test program. It was most likely because OpenCV had issues with the video format which resulted in distorted images. PYgame was compatible and therefore chosen to show the stream in the GUI and to send it to the AI. There were also some problems with radio disturbances in the lab which distorted the images significantly as well as shown in Figure 12

While designing the system, the communication interference was not given much thought. More thought should have been given to be able to guarantee an interference free communication, such as finding an optimal communication frequency or choosing a disturbance free environment. An autoencoder provided the possibility to a posteriori removal of the noise, taking a dataset (the BioID dataset in this example), adding the noise to it and finally training the autoencoder to remove it proved to be very effective as seen in Figure 13. However, the image is distorted such that wavy patterns occur exactly in the vicinity of the noise shown in Figure 12. Thus, a better approximation would be to manually distort the training pictures near the noised pixels as well.

While the YOLOv2 network proved to be a strong tool in detecting people, the position of a person projected on to a two dimensional surface failed to give the depth coordinate. Due to the fact that any one person could also appear of different size and shape depending on how they are positioned relative to the camera, one can not rely on blindly mapping the size of the detection box to the distance. A more invariant feature is the size of a face. One can use this as a distance mapping since the size of the face is relatively invariant throughout the human population (and definitely if one is talking about the same person). The problem with the pre-trained YOLOv2 network is that is has not been trained on human faces.

To account for this, several ideas have been proposed. An initial idea was to use the image within the bounding box as a new input for a face detection algorithm. By using a *canny edge detector*, one might use the edges detected in the image to find the best fit for an ellipse, which roughly represents a face. The problem with this method is that there are a lot of edges in a ”ordinary” picture, both within the face and outside of it, resulting in the ellipse often fitting around some random continuous edge. A different method was to try and apply the Viola-Jones algorithm within the ”person” box frame. This proved to be very successful, however one had to have a face that was perfectly aligned just like in the training set for the algorithm. Therefore, a more general approach was needed.

A different idea was to train the YOLOv2 network to detect faces. However, finding a labeled dataset that was large enough was difficult. The BioID dataset, consisting of roughly 2000 images, was being considered, since it seems that it the CNN detection is invariant to the orientation of the objects on the screen. The problem that was left to solve was thus to find a way to fine-tune the YOLOv2 network without losing its strengths. The implementation of retraining the model in *keras*

did not seem to work for the BioID dataset. Thus, a new implementation of YOLOv2 written in Darknet and converted though Darkflow was considered. However, there was not enough time to explore this direction.

Some difficulties with the crazyflie arose during testing. The camera, together with the rig made to attach it to the crazyflie, proved to be a bit too heavy for the crazyflie. Some alterations were made to reduce the weight, for example by removing excess parts. While this allowed the crazyflie to respond to higher control signals without giving an error signal, it was still not enough. It is most obvious when looking at [2]. There, the crazyflie can easily provide enough thrust rise upwards and hover, however, when it comes to descending, it lowers its thrust so much that the the descent becomes more like a free fall. Due to its heavy weight, it has a difficulty breaking this fall before hitting the ground.

A different problem that arose was that the internal attitude control of the crazyflie malfunctioned, causing the system to be difficult to stabilize around a point in the horizontal plane. Different crazyflies were also tried, but they also showed the same behavior. Although the systems were troubleshooted, there were no apparent errors found.

Videos of the control for the height is referred to in section IV-D. One may see that the system shows typical behaviors that one might expect with a PID controller, i.e., there are overshoots and oscillatory behavior. With a better choice of PID parameters, one might be able to create a robust controller. If the internal attitude control were to function properly, one could have used the Ziegler-Nichols step or frequency methods to find some optimal values for the PID parameters.

VI. Conclusions

In conclusion of the project, it was found that it is possible to include artificial intelligence to set reference values in a drone control system in real-time when the drone is paired with a PC that is running a consumer GPU.

Hardware, in terms of streaming a video feed by means of radio communication, was proved to be a a serious limitation. Future directions might be to upgrade the hardware so it can send images distortion free, or implement some algorithm that cleans up the signal. Another approach might be to retrain the neural network on faces/heads, which might give a more robust depth reference.

References

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Rijad Alisic. *Crazyflie GUI test*. 2017. URL: https://youtu.be/nyst_-VUgTU (visited on 12/19/2017).

[3] Rijad Alisic. *Test of Crazyflie*. 2017. URL: https://youtu.be/9Zj4P4Mg95o (visited on 12/19/2017).

[4] allanzelener, shadySource, and gardaud. *YAD2K: Yet Another Darknet 2 Keras*. https://github.com/allanzelener/YAD2K. 2017.

[5] *ASUS Xtion*. Dec. 13, 2017. URL: https://www.asus.com/3D-Sensor/Xtion_PRO/.

[6] *Autoencoders - Deep Learning bits*. Jan. 11, 2018. URL: https://hackernoon.com/autoencoders-deep-learning-bits-1-11731e200694.

[7] *BOSCAM FPV 5.8G 8CH 200mW Audio Video Wireless RC305 Receiver*. Dec. 13, 2017. URL: https://www.gearbest.com/fpv-system/pp_226494.html.

[8] *Building Autoencoders in Keras*. Jan. 11, 2018. URL: https://blog.keras.io/building-autoencoders-in-keras.html.

[9] *Convolutional Neural Networks (CNNs / ConvNets)*. Jan. 11, 2018. URL: http://cs231n.github.io/convolutional-networks/.

[10] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[11] *easycap video capture dc60*. Dec. 13, 2017. URL: https://www.fpvmodel.com/easycap-dc60-usb-2-0-high-quality-video-capture-with-audio-great-for-fpv_g131.html.

[12] *FX798T Micro FPV Camera*. Dec. 13, 2017. URL: https://www.getfpv.com/fx798t-micro-fpv-camera-5-8ghz-37ch-25mw-vtx.html.

[13] Ross Girshick. "Fast R-CNN". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015.

[14] Ross Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[15] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[16] *he PASCAL Visual Object Classes Challenge 2007*. 2017. URL: http://host.robots.ox.ac.uk/pascal/VOC/voc2007/ (visited on 12/20/2017).

[17] Jonathan Huang et al. "Speed/accuracy trade-offs for modern convolutional object detectors". In: *CoRR* abs/1611.10012 (2016). arXiv: 1611.10012. URL: http://arxiv.org/abs/1611.10012.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[19] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[20] *nRF23LU1 + 2.4GHz RF transceiver core*. Dec. 13, 2017. URL: https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24LU1P.

[21] *Object Detection (D2L4 2017 UPC Deep Learning for Computer Vision)*. Jan. 11, 2018. URL: https://www.slideshare.net/xavigiro/object-detection-d2l4-2017-upc-deep-learning-for-computer-vision.

[22] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[23] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[24] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 91–99. URL: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf.

[25] *Test of Crazyflie*. 2017. URL: https://www.bitcraze.io/crazyflie-2/ (visited on 12/20/2017).

[26] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. 2001, I–511–I–518 vol.1. DOI: 10.1109/CVPR.2001.990517.

[27] *Yolo Website*. Dec. 14, 2017. URL: https://pjreddie.com/darknet/yolo/.

Reflow Oven Control

Claudine Chaplais*, Jose Maria Martinez Moreno†, Simon Paulsson‡ and Jens Thieme Almkvist§
Lund University
*claudine.chaplais.5222@student.lu.se, †jo6139ma-s@student.lu.se, ‡tfy13spa@student.lu.se, §mas13jal@student.lu.se
https://gitlab.control.lth.se/regler/FRTN40/group-G

***Abstract*—This is an attempt to convert a small kitchen hot air oven to an oven usable in reflow soldering applications. With the use of simple consumer electronics control of the oven was achieved. The system was identified with a gradient based procedure and found to be a SOTD, and thus a PID controller was chosen. The controller was implemented in an Arduino board and actuated on the oven by a solid state relay. A simple thermistor probe was used as temperature sensor. Due to sensor failure in the late stages of the project, the controller was implemented only for around 130°C, instead of around 220°C. Results showed that the oven was not fast enough in cooling down, even with the hatch open.**

I. Introduction

A. Background

Reflow soldering is the process of soldering components to circuit boards by temporarily attaching the components using solder paste. The board is then subjected to controlled heat, melting the solder and permanently attaching the components to their contact pads.

The department of automatic control at Lund University has expressed a desire for reflow soldering. There are several commercially available reflow soldering ovens to buy, but the department is interested in seeing if one could be made from on site. Therefore. the purpose of this project was to look into the possibility of controlling a regular hot air oven so that it's temperature follows a reflow soldering temperature curve.

B. Reflow soldering

For reflow soldering the temperature has to follow a certain curve that is specific to the each type of solder. A conceptual graph of the temperature is shown in Figure 1 [7]. The soldering procedure is outlined below.

- Heat to a certain temperature (usually around 150°C), but not too fast (1-4°/s).
- Hold this temperature for about 1-2 minutes. This is to expel some components in the solder paste and to give the board an even temperature.
- Heat to about 220°C to solder melt the solder for about 45 seconds.
- Cool down, around 4°C/s, until room temperature.

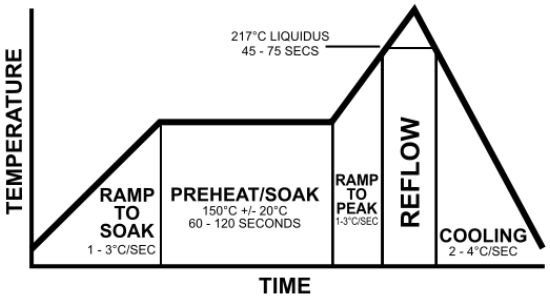


Fig. 1. A conceptual temperature curve of a typical reflow soldering process. [7]

C. Objective

The goal of this project is to control a hot air oven to follow a predetermined temperature profile. The range of temperatures and temperature changes are to be corresponding to those found in reflow soldering temperature profiles.

D. Hardware

project-automatic-control.bib The oven used in this project is a STEBA STKB19[6]. The controller is implemented on an Arduino Uno microcontroller [4].

II. Modeling

A. The oven

Figure 2 shows a principle sketch of the oven. The oven consists of two separated spaces, a baking space and an insulating space between the baking space and ambient environment. This is where all the electrics are located. The oven is heated by four electric resistors located at the top and the bottom of the baking space. There is also a fan circulating the air in the baking space to distribute the heat evenly.

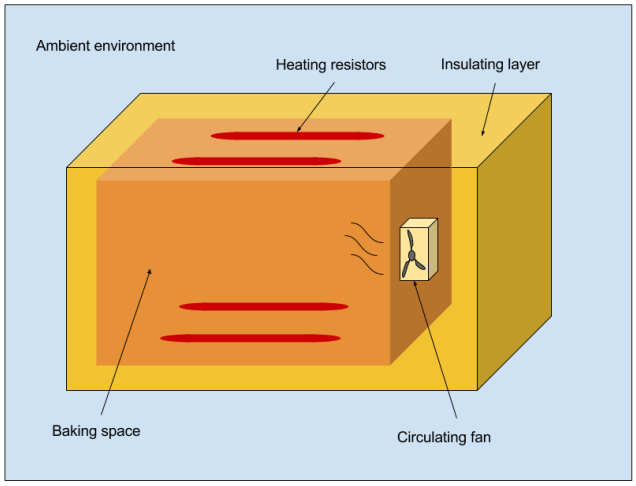


Fig. 2. Principle sketch of the oven.

For modeling the oven, Newton's law of cooling is used. It states that the rate of heat loss of a body is directly proportional to the difference in temperature between the body and its surroundings. The heat transfer version of the law is expressed with equation (1).

$$\frac{dQ}{dt} = h \cdot A \cdot (T(t) - T_{amb}) = h \cdot A \cdot \Delta T(t) = k \cdot \Delta T(t) \quad (1)$$

where,

Q is the thermal energy,

h is the heat transfer coefficient,

A is the heat transfer area,

$T(t)$ is the temperature of the oven,

T_{amb} is the ambient temperature.

The oven consists of three instances where heat is exchanged, each instant with different heat transfer constants. These three instances are as follows: heat transfer from the resistors to the air inside the baking space, heat transfer from the baking space to the insulating space and heat transfer from the insulating layer to the ambient environment. Choosing the temperatures of the resistors, the air inside the baking space and the air in the insulating space as one state since they are similar in heat capacity. Now the system can be expressed on state space form as in equation (2).

$$\begin{aligned} \dot{x}_1 &= \beta(x_2 - x_1) + \epsilon u \\ \dot{x}_2 &= \alpha(x_1 - x_2) + \gamma(t_{amb} - x_2) \\ y &= \begin{bmatrix} 0 & 1 \end{bmatrix} X. \end{aligned} \quad (2)$$

In the state space representation in equation (2), the heat transfer constant and surface area of the heat transfer have been combined into one single constant. Since it is a second order system (SOTD), with the approximations, it is appropriate to use a second order model for the identification.

B. Measuring

In order to estimate good values for the constants, measurements of the states had to be done. The measurements were initially done with a thermometer with a thermocouple probe to measure the air temperature inside the baking space and the insulating layer. The thermometer was read manually at time intervals of 5 seconds and the values were noted down. For measuring the temperature of the heating resistors, a pyrometer was used instead of a thermometer to avoid burning the leads.

As seen in Figure 3, initial tests showed that the cooling time of the oven was too slow for the purpose of reflow soldering and some kind of cooling is necessary to achieve a higher cooling rate.

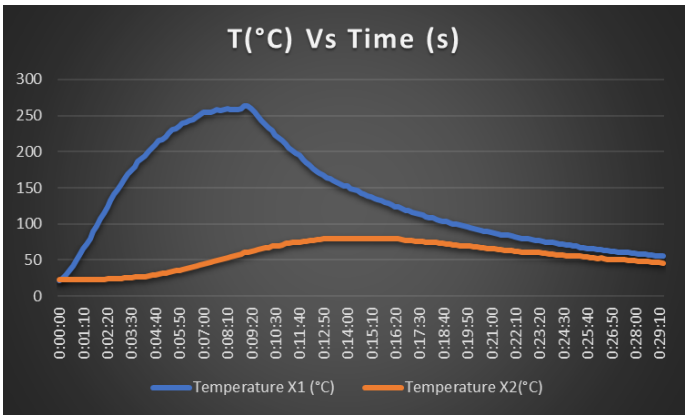


Fig. 3. Graphic of heating and cooling process with door closed. The blue line is the oven baking temperature and the orange is the insulating layer's temperature. Notice that the cooling of the baking compartment is far from 6 celsius/second.

As it can be appreciated, it takes more than 25 minutes to cool down again. The temperature needs to decrease approximately 6°C per second. The increased cooling could be very simply accomplished by opening the oven door when needed. The opening of the door could be done manually by the operator at certain time points or a way of automatically opening the oven door could be implemented.

C. System identification

For identification a gradient based identification proctimeedure as proposed in [5] is used.

This identification starts with using a relay function to obtain input and output signal data around a steady state. Then a model is guessed, simulated output based on the recorded input is generated. The real and the simulated output are then compared and a gradient of how to change the model parameters is used to get a better and better model. The advantage of this is that only a few oscillations of the system are required, which is good since the oven takes up to 25 min to cool down from 230°C. When identifying around 128°C the SOTD expressed in equation (3) was identified,

$$\frac{0.01621}{s^2 + 0.2728s + 0.000397} \tag{3}$$

with a DC gain of 40.83 and the time unit of seconds. The closeness to the real process is shown below in Figure 4.

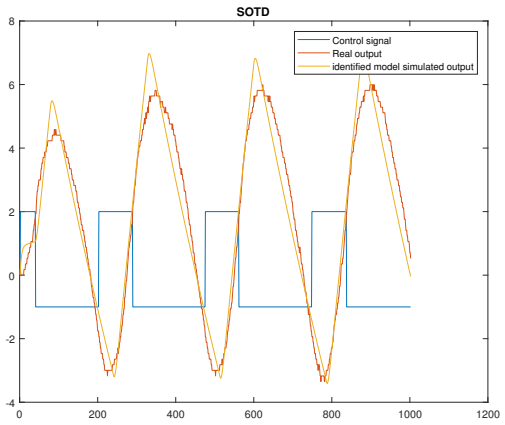


Fig. 4. Control signal from a relay controller along with the process output and the simulated output from the identified transfer function.

D. Simulation

Simulations were performed in the simulink system shown in figure 5 was used. This environment can follow different curves, add load disturbance and add a artificial gain to the process to see what would happen if the process gain would be different.

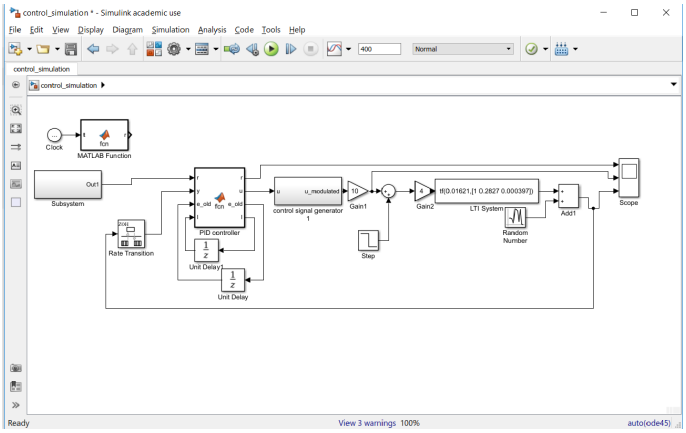


Fig. 5. The simulink model that was used to simulate control of the process.

In this environment two types of curves can be followed. Firstly a reference curve similar to a typical curve that is required to be followed when soldering and secondly a ramp up to a steady state. Furthermore, load disturbance can be applied if wanted and an added process gain to see what would happen if the process has higher gain than identified. The process model used was the identified one.

The design parameters that were used can be seen in table I.

TABLE I
CONTROLLER PARAMETERS

Parameter	Value
kp	1
kd	10
ki	0.01

The results of the simulations can be seen in Figure 6, Figure 7 and Figure 8.

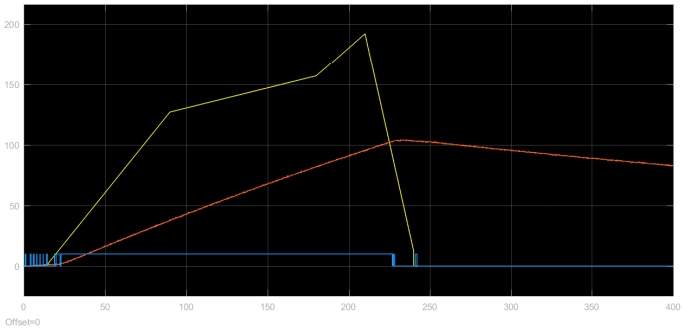


Fig. 6. Result of control simulation with parameters form table I. As can be observed the process is to slow, considering that the input signal is saturated.

As can be observed in Figure 6 the process is simply to slow to achieve our purpose. To rectify this in simulation, to see what would happen if it would have been faster, a artificial process gain of 4 was introduced. This produced the result below in Figure 7.

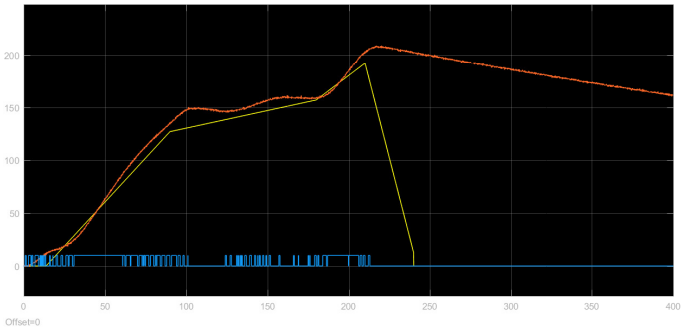


Fig. 7. Result of control simulation with parameters form table I. An artificial process gain of 4 is here introduced. Here, due to the increased gain, the controller is a little to aggressive though since it overshoots.

This shows that the controller is a little to aggressive given the new process.

To see how well the controller could handle load disturbances a reference curve up to a stationary value was also tried. A load disturbance was introduced a 250 seconds in the form of a step with value -2. The results in Figure 8 show that it does not affect the output signal much.

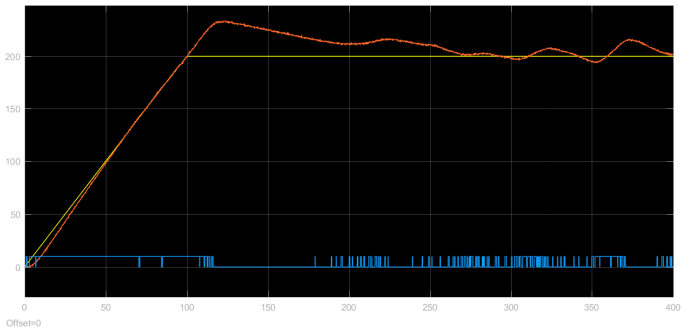


Fig. 8. Result of control simulation with parameters form table I. The artificial load of 4 is kept. Here a load disturbance of -2 is introduced at 250 seconds.

III. Electro-Mechanics

The oven. with all it's peripherals and the Arduino Uno together with the LCD-display is shown in Figure 9.

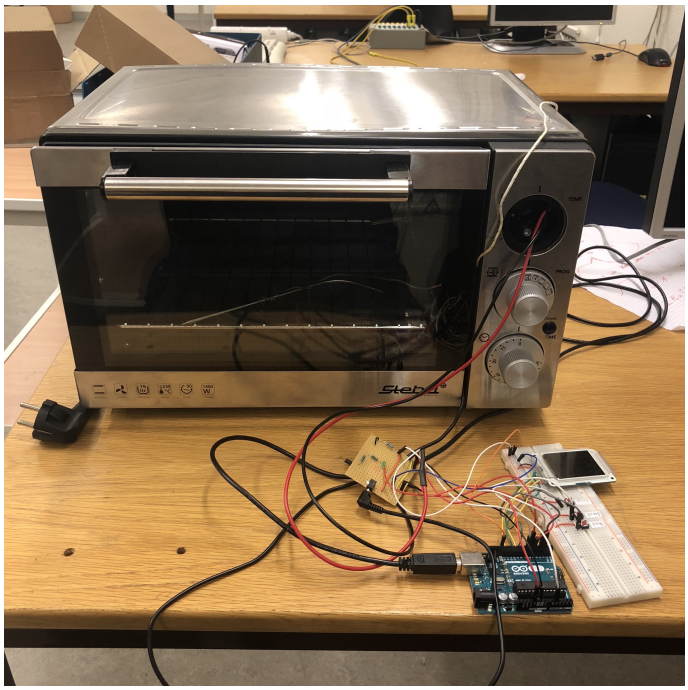


Fig. 9. Photograph of the oven, Arduino Uno, LCD-display. The temperature sensor is inside the oven.

A. The oven

The wiring of the Steba KB19 is a simple setup. The oven relies on heat-induced resistance in the heating elements to do the closed loop on-off control. Figure 10 shows the wiring diagram of the unmodified oven. The most important parts are the following:

- Power supply: Connected to the grid (230 V).
- Timer: A dial which lets the user set the time for oven to be on.
- Process: A dial that lets the user set which mode to cook in (i.e. fan on/off, upper/lower heat).
- Fan: In order to move the air inside the oven. Turned on by starting the timer and choosing correct mode.
- Resistance R1, R2, R3, R4: These are the heating resistance which heat up the oven.

The timer dial has to be turned up for the oven to be turned on. It works as the on button for the oven. This feature will be kept after modifying it as a safety feature. It will ensure that the operator always is able to manually turn of the power to oven mechanically if a catastrophic error would occur in the software or electronic hardware. It also serves as an extra safety feature should the oven be left unattended and an error occurs since the timer will turn of the oven eventually.

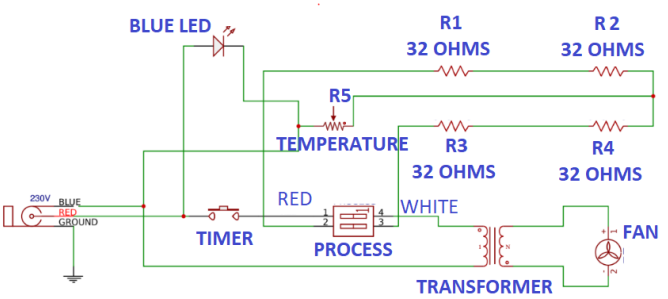


Fig. 10. The wiring diagram of the unmodified Steba KB19.

The oven was modified by replacing the on-off controller with a solid state relay that could be controlled by the Arduino board. The Arduino board opens and closes the relay by sending 5V or 0V to the relay, which then provides the oven's resistors with either full power or no power. This way, in order to get a control signal other than full power or no power, the control signal needs to be generated by a pulse width modulator. The circuit diagram of the modified oven is shown in Figure 11.

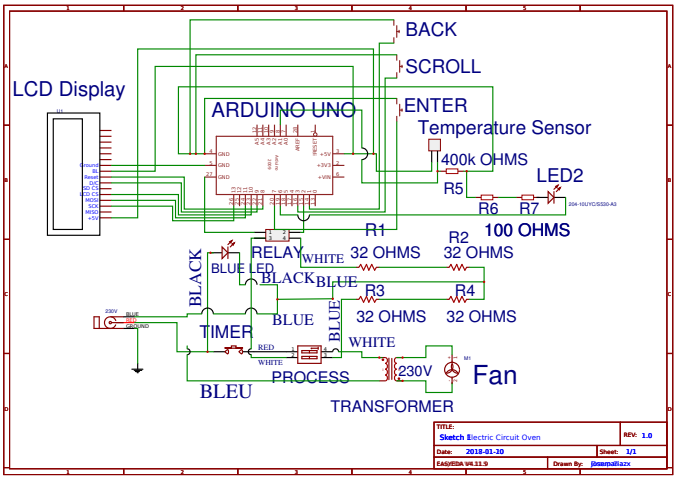


Fig. 11. The modified wiring diagram of the Steba KB19.

B. Temperature sensor

Different temperature sensors were considered and two types were tested, a 3-wire PT100 [10] and a simple steak thermometer probe [9]. The PT100 was considered because of it's accuracy, stability and well documented characteristics [2]. The steak thermometer was considered due to it's availability and simple setup configuration. Both types are relatively cheap. However, the PT100-sensor tested had too much thermal inertia and was abandoned. The steak thermometer probe also has some thermal inertia, but not nearly as much as the PT100.

The steak thermometer probe is a simple thermistor. The resistance of the thermometer is exponentially dependent on the temperature. In order to get the resistance as an input to the Arduino board a simple voltage divider was used. Equation (4) describes the relationship between resistance and measured voltage.

$$V_{out} = \frac{R_1}{R_1 + R_{sensor}} \cdot V_{ref} \tag{4}$$

The reference voltage (5V from the Arduino board) produced a high enough output voltage for the Arduino to read with reasonable resolution without the need for amplification.

C. User interface

The user interface consists of a TFT LCD display [3] and three simple push buttons. The user can choose between preprogrammed temperature curves. The connections for the display and buttons can be seen in Figure 11. A diagram showing how the user interface works is shown in Figure 12. There are three buttons, enter, scroll and back. Pressing enter moves the indicator to the right, pressing scroll moves the indicator downwards (then back up top when pressing scroll while at the lowest menu item) and back brings the indicator back a step to the left.

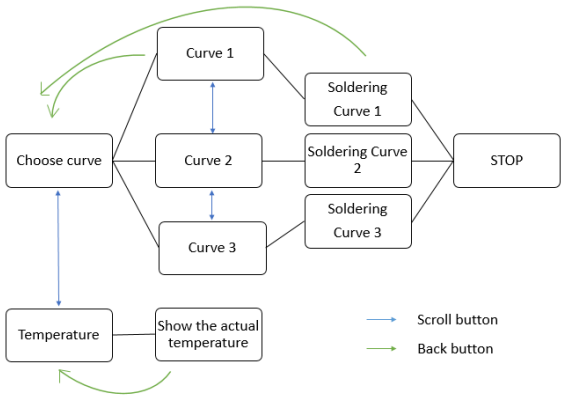


Fig. 12. A flow diagram of how the user interface works.

D. Microcontroller

To control and run it all, an Arduino UNO board is being used. It is a cheap and user friendly microcontroller based on the Atmega 328P. The Arduino UNO is somewhat easier to use since it can be programmed with the Arduino sketch language. This language is based on C, and since the whole board essentially is an Atmega 328P, writing C code works as well. Figure 11 shows how the board is connected to everything else.

IV. Control

A. Control design

The controller chosen for this project was a PID controller implemented on parallel form. This design choice was made because of the difficulty of implementing e.g. a model predicting controller compared to a PID controller. As can be seen in (3), the system is almost an integrator, which would suggest that a PD controller would be sufficient. However, since the problem really is the servo problem (not the regulator problem) with a continuous reference curve, a small integrator part can be justified to help with the tracking of the curve. Also, the rough model shown earlier in (3) would suggest that a second-order time-delay (with our system exhibiting no delay) system describes the system accurately enough. The accuracy would be increased if the system was identified as a linear system around two important points, which would be at around 150°C and 220°C. This would also mean that gain scheduling would be required, using two different controllers, one below e.g. 180°C and one above.

Early tests of the system showed that the hatch to the oven needed to be opened in order to cool the oven fast enough. See Figure 3. Here, the most important thing would be the cooling rate and not the temperature, which would require a different controller. The system also changes dramatically when the hatch is open which further motivates the need for a different controller when cooling.

In order to avoid having to implement a multiple input single output system(how much to heat the resistors and how to open the hatch), the hatch was to be opened a set amount and then use the rods for controlling the temperature descent.

Initial thoughts for choice of the parameters was to use a constrained gradient based tuning as proposed in [8]. This was abandoned in the end due to lack of knowledge and time. Instead, parameters were estimated with the help of plotting step responses in MATLAB. The sampling time was chosen to 1 second because of the slow dynamics of the oven. Since the actuator can only do the two binary states on/off a PWM which divided the sampling second into tenth was used.

Because of the limitations imposed by the temperature sensor the system was identified at 128°C, which in steady state corresponds to a 20 % duty cycle. This also meant that the controller performed worse than it would, if the oven was hotter. The temperature probe responds almost linearly from 100°C to 180°C, but at room temperature the response is very non-linear.

Final tests showed that opening the hatch was not enough to cool the oven as quickly as needed, see Figure 14. At higher temperatures, the cooling would be faster, but it would still slow down when approaching 120°C.

Also, there might be a problem with having the control signal saturated at on for a long time since the heat from the heating elements takes time to transfer to the air inside the oven. This means that there is a need for a software implemented saturation to avoid overheating. Control wise, this is captured by the model, but the controller is controlling the baking space temperature. In this case it is the heating resistors that could overheat without the baking space getting dangerously hot (e.g. if the hatch is open and you are trying to achieve maximum temperature).

B. Implementation

The controller is implemented on an Arduino UNO board. The controller would be faster if it was implemented using fixed point operations to accommodate for the board's somewhat lacking performance but since the process is slow, there is no real need for speedy calculations and therefore the controller was implemented with floating point operations.

During testing it was discovered that the temperature probe could not handle sustained exposure to temperatures above 170°C. Therefore the team decided to implement the solution with a maximum temperature of 130°C to guarantee that probe could be relied on.

The program consists of two parts, interrupt routines that handles the control aspect and the main loop, which handle the user interface. A diagram showing the program structure can be seen in Figure 13. Two interrupt routines handle the sampling and actuation. One handles the output of the control signal and it actuates with a frequency of 500 Hz. The other routine is responsible for sampling, calculating and updating the control signal. This is done with a frequency of 1 Hz.

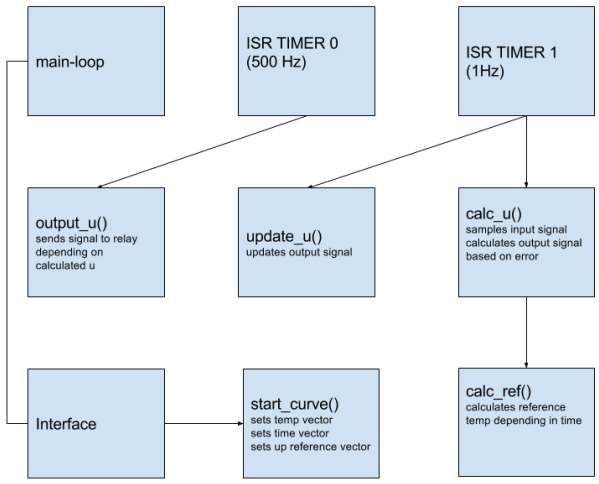


Fig. 13. Picture displaying major functions and what they do. Arrow means that the function calls the pointed to function. The interface is not detailed here, se Figure 12 instead.

The controller is preprogrammed with 3 different temperature curves as is. These are coded as vectors in the source code and the user interface determines which curve to run. The temperature curves consists of one time vector and one temperature vector. The temperature vector holds the different temperature values in °C in chronological order for the temperature curve and the time vector holds the times in seconds for when these temperatures should be reached.

From these two vectors the reference is generated, by creating a linear equation between points in the temperature and time vectors, so that the reference becomes a straight line between the two values. This is done once the user has chosen a curve and started it.

The input is then compared to the reference to generate the error in degrees which the control signal then is calculated from. The control signal is calculated as described in (5),

$$u = k_p \cdot e + k_i \cdot \sum e + k_d \cdot (e - e_0). \tag{5}$$

The integral part of the control signal is checked for anti-windup by before added. The anti wind-up function checks if $-128 < e + \sum e < 127$ and sets it to one of the extreme values if it outside that range. The total control signal is saturated at ± 10 before being updated. This is done once every second by a interrupt service routine set up to trigger on TIMER1.

The control value is then used as the duty cycle for the PWM that is the final output. For example, an output value of 7 results in a duty cycle of 70%. A negative control value means that the controller wants to cool, but as it has has no way of doing so, it just menas that the output is set to 0.

The interface code is a series of switch-statements that relies on a counter that increments or decrements when a button is pushed. When the user has traversed to a 'start curve'-option, a function that initializes the time- and temperature vectors corresponding to the chosen curve and starts the controller is called.

To capture results, the serial port recording software Coolterm [1] was used. The controller program simply sent the relevant values separated by commas to the serial port, it was recorded and saved by Coolterm in a text file. This text file was then inserted into a MATLAB workspace with the use of the import wizard, and the plot function was used to plot the temperature measured, reference signal and control signal against the time.

V. Results

Below are some results from tests shown. The red line is the reference, the blue line is the measured temperature and the yellow line is the control signal.

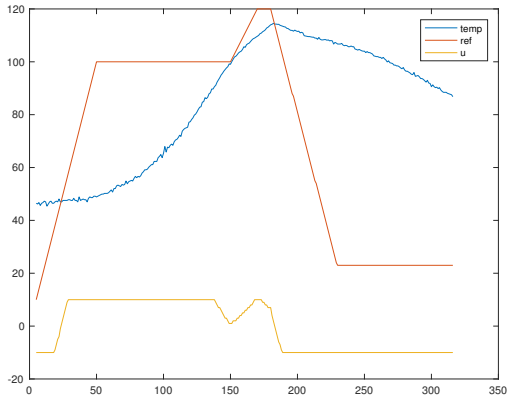


Fig. 14. A complete run of curve 1 with the hatch fully opened.

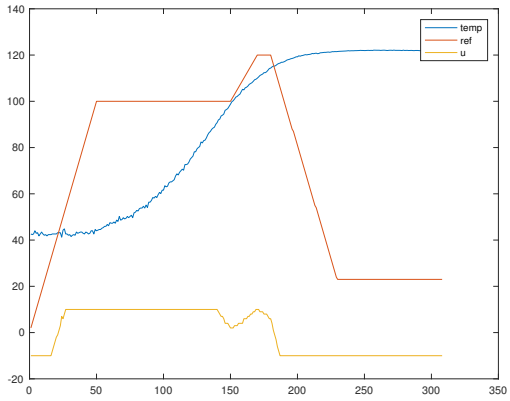


Fig. 15. A complete run of curve 1 with the hatch closed.

Figure 14 and Figure 15 shows the same reference curve, beginning at roughly the same temperature but with the difference that the hatch was fully opened in Figure 14 and kept closed in Figure 15. Even with full actuation and closed hatch, the oven lags way behind in heating, although this could be attributed to some inertia in the temperature sensor as well. As for the cooling, even with hatch opened full, there is no way that a decrease rate of 4°C/s can be achieved without any other means of cooling.

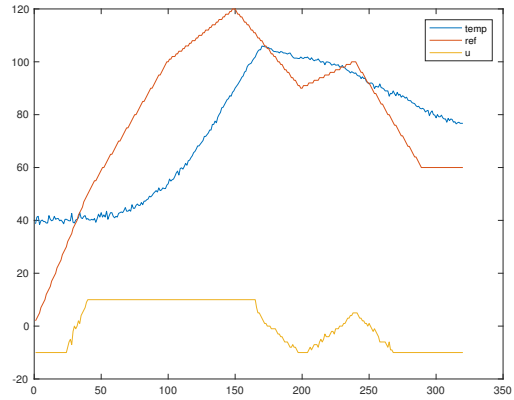


Fig. 16. A complete run of curve 3 starting at 40°C, hatch open.

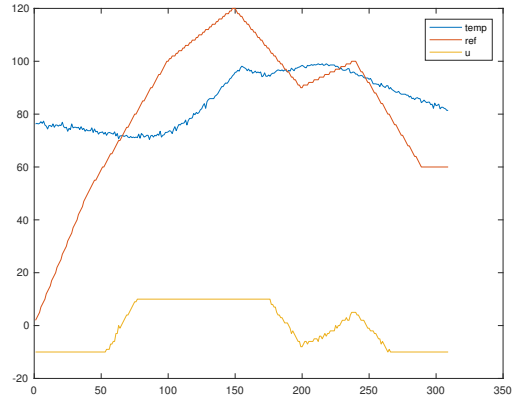


Fig. 17. A complete run of curve 3 starting at 80°C, hatch open.

Figure 16 and Figure 17 shows the same reference curve with the exception that the initial temperature is different. This shows that the controller is actually slower when started at higher temperature. This is probably due to the thermal inertia in the resistors.

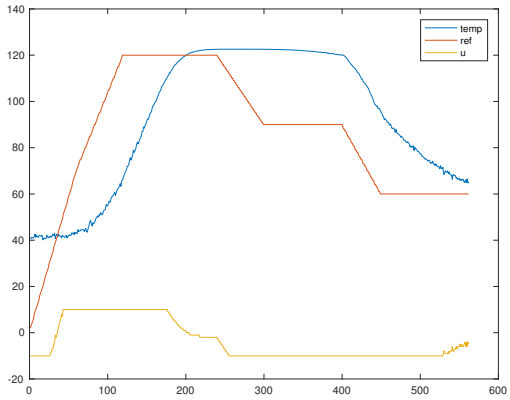


Fig. 18. A complete run of curve 3 starting at 40°C, hatch initially closed, opened at 400 seconds.

Figure 18 shows the same reference curve as Figure 16 and Figure 17 but with the hatch initially closed and opened at 400 seconds. This run shows some promise, and tells about the need for having the hatch closed when heating and active cooling when needed.

VI. Discussion

A. The hardware

This project has proven the importance of knowing the limitations of the hardware that are being used. The team discovered very late in the process that the oven was too slow in both heating and cooling down (even with the hatch open) to be suitable for reflow soldering. The team thought it would be sufficient with opening the hatch to achieve the desired rate of cooling but as the results show, that is not the case. At higher temperatures, around 230°C, opening the hatch might cool the oven fast enough, but as the temperature difference between oven and ambient environment decreased so did the rate of cooling. This means that a proof of concept. as in having a cooling rate of 4°C/s, was not possible for the lower temperatures in the tests.

Having the proper sensor for the job is also something the team discovered is crucial to getting good results. Much work was dedicated to finding a suitable sensor, then installing it, only to find out that it was too slow. The PT100 was stable and very accurate. It is also very well documented, which means that it is easy to get correct temperature readings. The 3-wire connection version that was tested also doesn't drift over time if properly set up, which made it an ideal candidate for this application. It was also rated for up tp 400°C. It's only, and also fatal, flaw was that it was very slow. The thermal inertia in the probe was too high for this type of application and this sensor could not be used.

The steak thermometer probe, which is the one used, is far from perfect. It was easy to set up, just a simple voltage divider. But there is no documentation of how it behaves and it was discovered that it responded exponentially. It did not stand prolonged exposure to high temperatures. During late stage testing one probe was destroyed. This was just at 170°C, well below the required temperature needed for reflow soldering. But working with limited time, it was decided that an implementation at lower temperatures to prove the concept of the oven was better than no implereport-2017-group-G.bblmentation at all.

B. The software

Splitting up the the software development resulted in two standalone programs that worked on their own but not together. The two programs needed to be merged in order for the complete program to work as intended. This proved to be a lot more difficult than expected. It would have been better to have developed the program as one, but due the time limits of this project it was not possible. The problem could have been avoided, or at least mitigated, if a more detailed plan of the two programs' structures and resource requirements were made before software implementation. Alternatively, there could have been possible to use a programmable LCD-screen which would run the interface code and just send what choices had been made in the menu to the Arduino. This would mean that their is two programs workin on separate units and no merging would have been required.

C. Downstream development

The results show that process probably would benefit a lot from model predictive control, if the system could be identified accurately enough. If the slow time constant could be predicted, better results could be achieved.

For possible future development of this project, a thermocouple probe as a temperature sensor is an option worth considering, since they are common, reasonable priced and fast. They also come in variants rated to withstand temperatures well above 400°C. A way of cooling the oven would be beneficial as well.

References

[1] Dec. 2017. URL: http://freeware.the-meiers.org.

[2] Mats Alaküla et al. *Grundläggande Elektroteknik*. 3rd ed. Lund: KFS AB, 2010.

[3] *Arduino LCD Screen*. Arduino. Dec. 2017. URL: https://store.arduino.cc/arduino-lcd-screen.

[4] *Arduino Uno*. Arduino. Dec. 2017. URL: https://store.arduino.cc/arduino-uno-rev3.

[5] Josefin Berner and Kristian Soltesz. "Short and Robust Expermiments in Relay Autotuners". In: *Proceedings of the 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation; ETFA2017* (2017).

[6] *KB19*. Steba Elektrogeräte GmbH and Co. KG. Dec. 2017. URL: http://steba.com/product/baking/bake-oven-kb-19/?lang=en (visited on 12/20/2017).

[7] *Reflow soldering*. Dec. 2017. URL: https://en.wikipedia.org/wiki/Reflow_soldering.

[8] Kristian Soltesz, Chriss Grimholt, and Sigurd Skogestad. “Simultaneous design of proportional-integral-derivative controller and measurement by optmimisation”. In: *IET Control Theory and Applications* (2016).

[9] *Stektermometer Digital Svart Ica Cook and Eat*. Ica Sverige AB. Dec. 2017. URL: http://www.icahemma.se/Stektermometer-Digital-Svart-ICA-Cook--Eat-p-15792.aspx (visited on 12/20/2017).

[10] *Temperaturgivare Pt100, RTR-M08-L050-K04, Emko Elektronik A.S.* Distrelec Group Inc. URL: https://www.elfa.se/sv/temperaturgivare-pt100-emko-elektronik-rtr-m08-l050-k04/p/30094643?q=pt100&page=3&origPos=21&origPageSize=50&simi=99.6 (visited on 12/20/2017).

Path-Following Balanduino Using Computer Vision

Jonas Christensen Strömgren*, Marcus Peterson †and Joel Sirefelt‡
Lund University
*tfy13jst@student.lu.se, †tfy12mpe@student.lu.se, ‡joel.sirefelt.910@student.lu.se
https://gitlab.control.lth.se/regler/FRTN40/group-H.git

***Abstract*—This project examines and develops a method for path following with a Balanduino using computer vision and control theory. The Balanduino already had a functional balancing controller at the start of the project. Using color filtering from OpenCV a well functioning path detection was made. Utilizing this path detection a PID controller managing turn rate and an optimized speed controller function were designed. These combined made the Balanduino follow the colored track in a smooth and fast manner.**

I. Introduction

A. Balanduino

The Balanduino is an Arduino compatible open source balancing robot kit created in a kickstarter campaign by TKJ Electronics in 2013.[5]

By combining accelerometer, gyroscope and rotary encoder data, the Balanduino is able to keep itself balanced even when pushed. It uses a highly optimized and tuned PID controller and a Kalman filter to keep the robot balanced.

B. Path following robots

Path following robots are nothing new, there is an ocean of DIY (Do-It-Yourself) guides and university papers which can be found on the web.[3] [2] Most of them use an array of infra-red leds and photo diodes placed on the robot directed down towards the ground to recognize the path. They control the robot with a PID controller where the output signals is the torque in the robots right and left wheel. Using computer vision to follow a line is an area less researched and gives room to simultaneously use the sensor(camera) for other uses while following a line, like avoiding obstacles.

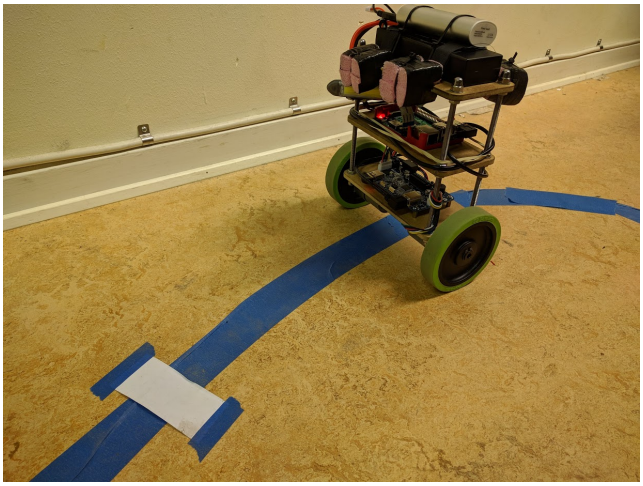


Fig. 1: The finished robot

II. Modeling

The modeling of the system depends on what problem one is trying to solve. Initially it was thought that the problem to solve was that of an inverted pendulum balancing by two wheels, with the only input being the torque to each wheel. It was similar to the inverted pendulum on a cart problem that is a common control problem. Instead of a cart the wheels of the system serve as a base and the wheel center point is the same as the pendulum endpoint. The dynamic equations of motion were derived using force and moment calculation in three dimensions. A few assumptions were made regarding the system for simplification. First is that the wheels are in constant contact with the ground and they do not slip on the surface. Second is that the nonlinear system is linearized around a small angle of the pendulum from the vertical. Assisting in the derivation of the equations were also a master thesis by a few students trying to create a robotic walker.[1] This lead to a state-space representation as below:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi}_p \\ \ddot{\phi}_p \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & A_{23} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & A_{43} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi_p \\ \dot{\phi}_p \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ B_{21} & B_{22} \\ 0 & 0 \\ B_{41} & B_{42} \\ 0 & 0 \\ B_{61} & B_{62} \end{bmatrix} \begin{bmatrix} T_{LW} \\ T_{RW} \end{bmatrix} \tag{1}$$

Where x represents the position, ϕ the pitch of the Balanduino, ψ the yaw, and $T_{LW}\&T_{RW}$ the different torques. The development of a control-scheme to control the position and yaw began, but the model had to be abandoned. It was found that the internal balancing control and the systems connected to it where so deeply programmed that to access the torque would be to disturb this ecosystem, which would've been outside the scope of our project.

The next idea for modeling was then to look upon a vehicle driving along a road and trying to optimize around the road and the coming curvature.

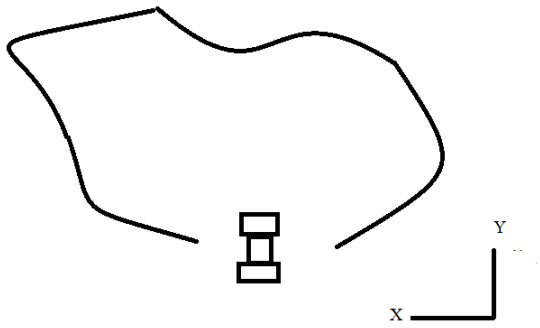


Fig. 2: Sketch of the modelling

The Balanduino was looked upon as a two-input - two-output system. The inputs were speed and angle references, and the outputs the actual measurement from these. These could then be used to estimate the states x and y, and be used to optimize a path following trajectory using some algebraic method. The equations of motion were as seen in equation 2.

$$\begin{bmatrix}\dot{\theta}\\ \dot{x}\\ \dot{y}\\ \dot{v}\end{bmatrix} = \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & \cos(\theta)\\ 0 & 0 & 0 & \sin(\theta)\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\theta\\ x\\ y\\ v\end{bmatrix} + \begin{bmatrix}1 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}u_1(t-h_1)\\ u_2(t-h_2)\end{bmatrix} \quad (2)$$

However, the group was hit by a backlash when the modeling group was halved. This lead to the decision to further simplify the model into an offset-correcting model. This was chosen due to its simplicity and good possibilities of integration with the camera. Thus the the model simply turns into the equations seen in equation 3&4.

$$\dot{\theta} = \Omega u_1 \quad (3)$$

$$v = u_2(t-h_2) \quad (4)$$

Here omega is the turning speed of the Balanduino, the time constant is dependent on the system speed and u_1 & u_2 are turn and speed, respectively. However, since the internal balance control is not perfect and is heavily interconnected with the turning, it is proper to model the system as the output having some measurement disturbance, due to the unstable nature of the inverted pendulum.

Later discovered during testing, it seems that control signals might also be having issues with disturbances. Hence, the model becomes as equation 5&6.

$$\dot{\theta} = \Omega u_1 + w \quad (5)$$

$$\begin{bmatrix}y_1\\ y_2\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}\theta\\ v\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}u_1\\ u_2(t-h_2)\end{bmatrix} + \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v_1\\ v_2\end{bmatrix} \quad (6)$$

The different time constants will be determined through testing. The h_2 time constant was deemed to be to small to be of any consequence to the system, whereas the Ω constant was determined to be 1.9 rad/s.

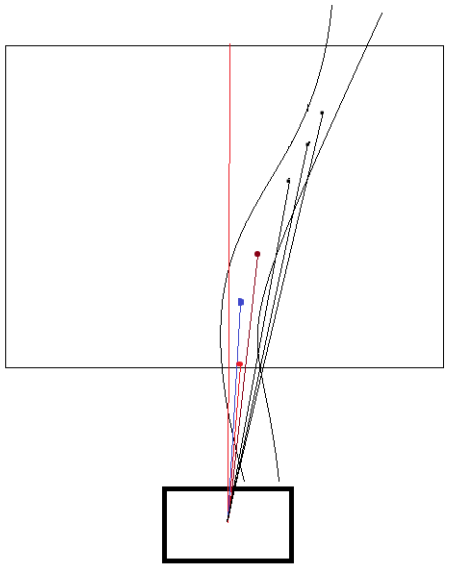


Fig. 3: Picture of the path and what angles are calculated

The modeling of the path can be found in Figure 12.

This is the path that the computer sees, and the objective is to minimize the different offset angles as quickly and smoothly as possible. Here the rectangle enclosing the road a bit ahead of the Balanduino is what the camera actually sees, since the camera does not see straight down. The distance from the wheel axis to the first point the camera can see was measured to be 17 centimeters. A conversion between pixels and centimeters was measured and thus we could transform the offset distance to an offset angle between the road and the Balanduino. This is referred to as offset angle theta in the picture. If the road is straight ahead of the Balanduino, we have zero angle offset.

However, this is purely to control that the Balanduino stays on the track. To be able to control future values and follow smoothly, we also need to look ahead. To do this we chose two points at different fixed distances ahead. These two points together with the first point are used to control the following of the path. They can be seen as the reference generating points, which are summed up using different weights to be able to counteract future offsets properly. The equation is as below.

$$error_\theta = 0.5 \cdot \theta_1 + 0.65 \cdot \theta_2 + 0.65 \cdot \theta_3 \quad (7)$$

Where the different θ begin from the point closest to the Balanduino and following subscripts move outwards. The closest point is always at the closest end of the picture, the others at fixed pixel distances in the picture. This means that the absolute real world distance might change due to oscillation in the balancing of the Balanduino, but nonetheless it has been shown to be rather robust.

The $error_\Theta$ variable will be the input to the controller. The offsets have almost equal value, due to the speed we are

reaching. It is important to try to correct current errors, but future values must be accounted for since the points are rapidly approaching. One can easily see that this equation also yields less turning when the angles have opposite values. The reason behind this is that a strict following of the path is desirable, but it is more important to predict and correct for future errors as well, otherwise the current error will be quickly corrected, but the future error even greater. This can lead to oscillatory and erratic behavior of the process, and losing the vision of the path, which is not desirable.

Lastly, to provide good robustness in the operation and not losing the track from sudden curves in the road, we need to regulate the speed so as to not go too fast when trying to do sharp turns. To do this, we anticipate the curvature of the road. This is done by taking three points far away from the Balanduino and calculating the angles.

These angles provide an understanding of the curvature of the road ahead and allows for control of the speed in such a way that when the roads curvature is high the speed is turned down so as to be able to follow the path smoothly.

The different angles are then fed into a function as below.

$$speed = 100 - 17 \cdot (tanh(\frac{|\theta_4| + |\theta_5| + |\theta_6|}{40} - 3) + 1) \quad (8)$$

The tangent function is here used to disregard small changes of the curvature of the path. The change in speed value is also limited to 0.25 per loop, which means, since the camera speed is around 50fps, that we can maximally change the speed 12.5 per second. This is to smooth out the input change of the speed, so as not to get rapid changes, which has been proven to be detrimental to the process.

A problem occurs when the curvature is too large and the camera can't see any of the points. Therefore when the image analysis doesn't discover one of the three points they are set to a value of 95 which will drastically slow the robot down.

A. Simulation

To try and simulate the system in MatLab a Simulink model was created, with the parameters taken from the mathematical modeling, see Appendix A. This system was used to tune a PID-controller using the Ziegler-Nichols[6] method to get an apt controller for the process. This controller was later tuned to get even closer convergence to the track, with parameters on parallel form being: $P = 1$, $I = 1.2$, and $D = 0.17$. The initial controller tuning seems to have been too lenient due to a lower value of Ω which had not been appropriately measured. This simulation has not been proper enough to converge with the real world values and will require further work, so as to be used as a good tool. The speed seems to not have as large of an effect on the error as has been observed in the real world application, thus the model is not correct.

III. Implementation

A. Hardware

The following hardware were used in the project.

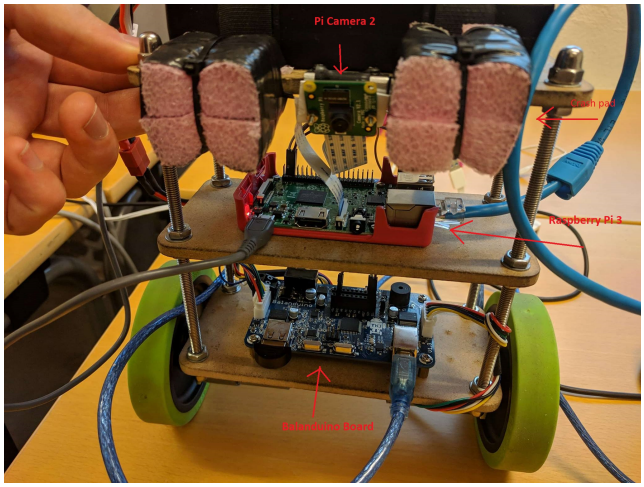


Fig. 4: Picture of the Balanduino with the added hardware modifications. Arduino board on the first level, Raspberry Pi on the second level, batteries and camera on the third level.

- Balanduino Balancing Robot
- Raspberry Pi 3
- Raspberry Pi Camera V2 Daylight
- Color coded tape track

The balancing Balanduino had already been built before the project was started. What has been added and fixed to the robot in the form of hardware is a Raspberry Pi 3 and a Raspberry Pi camera V2 Daylight. Figure 4 shows the robot refitted with the added hardware.

The camera was fixed using a aluminum plate that was bent and attached to the third level of the wooden structure with screws. Since the Raspberry camera had circuitry on the back of its board a layer of tape was put in between the board and the plate in order to prevent short circuits. The Raspberry Pi was attached to the second level wooden board with screws. On the top level an extra mini USB battery was put in place to power the Raspberry Pi. A Serial port cable connects the Raspberry Pi with the Balanduino.

Some crash pads were built and assembled to both sides of the robot using zip ties, so as to protect the components when the Balanduino falls over. A small flashlight was added in the end to make the robot see the tape better in dark areas of the room.

B. Software

The first sketch of the software system was designed as seen below in Figure 5. The system uses one monitor to keep track of the inputs and outputs to the control function. The OpenCV thread handles the image analyzing, the Regul thread calculates the new outputs/control signals while another thread sends the outputs/control signals to the Balanduino.

This software architecture was redefined because of two main reasons, one being that running multiple threads on Python has not been proved to work well in real-time. The problem has been discovered to happen because of the GIL,

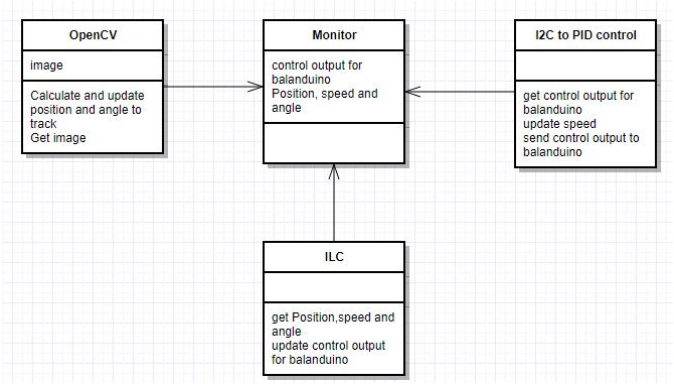


Fig. 5: Simplified UML class diagram.

Global Interpreter Lock in Python. GIL prevents multiple threads from accessing a global variable. The other reason why the software structure has been changed is because several tests have been conducted using the I2C (Inter-Integrated Circuit), which caused poor performance. The Balanduino already uses its I2C for balancing and when too many steering values are sent through that connection from the Raspberry Pi the Balanduino can't handle the balancing anymore. This has been solved by using a slower communication route, the serial port.

As for the threads the software structure has been redesigned to use only two threads. Previously the Regul and I2C threads were running too frequently because of their small computational demands which slowed down the OpenCV thread. Reading through a blog post by Adrian Rosebrock on increasing FPS (Frames per second, camera) of a Raspberry Pi videostream it was also deemed necessary to have a separate thread for grabbing the images from the camera.[4] The software layout was then changed accordingly, see Figure 6. One thread which handles the PiCamera stream and one main thread which takes frames from the PiCamera thread. The main thread then analyses the frames using OpenCV, calculates the new control commands and sends them to the Balanduino via Serial communication. The main thread also measures the FPS.

1) Computer Vision: Figure 7 shows an example of a frame from a video taken with the Raspberry Pi camera at full resolution.

OpenCV uses Numpy which is a package for scientific

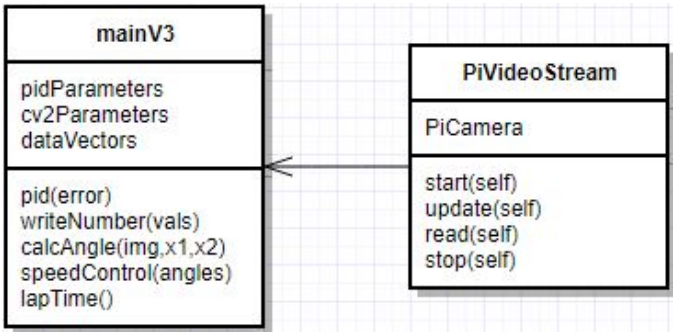


Fig. 6: New UML class diagram using only two threads.

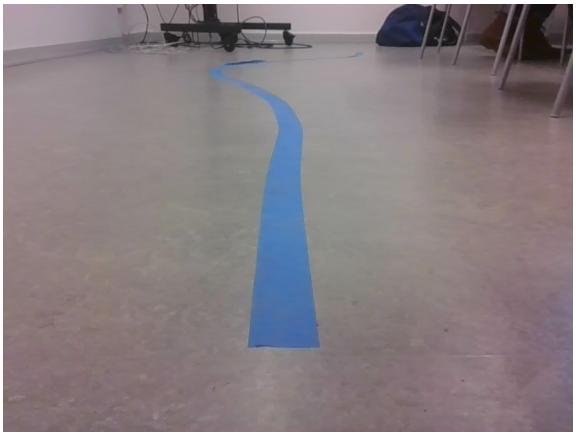


Fig. 7: Frame captured from Raspberry camera

computing with Python. It makes it possible to represent each frame as a double array of pixels. Using built in functions in the Open-CV package everything except the color blue in the RGB spectra can be filtered out. Combining these two functions gives a Numpy array where each pixel that represents a blue color in the original frame is converted to the color white(255,255,255) in the RGB scale and all other the other elements is represented with the color black(0,0,0). Below in Figure 8 this Numpy array is shown. There are a lot of different factors that change the characteristics of the blue colored tape in terms of RGB values that the camera picks up. These are for example, angle from camera to tape, distance from camera to tape, lighting in the room, shadows and background. Therefore this RGB filter had to be very specifically designed to the conditions of the place were the track was. Tests were conducted using many pictures of the track at different places in order to design a good working color filter. To filter out noise a method in Open CV named morph was used. The function slides through all the pixels in the picture making the software a bit slower and decreases the frames per second.

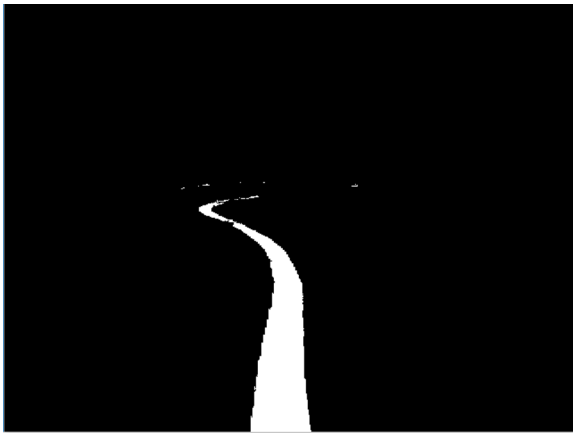


Fig. 8: Similar frame were everything except the blue color tape is filtered out using open-CV

2) Communication: To communicate with the Balanduino a package named pySerial was used. The main Python thread sends x and y positional offsets as strings encoded into bytes via serial communication. The strings are received through the Balanduino serial port and a built in function in the Balanduino script converts the string to two float values, x and y offsets. The Balanduino then tries to change its position to match the offsets. Sending several control signals per second translates these positional offsets to acceleration and turning. Large positive y offsets result in increased forward acceleration and backwards acceleration for negative values. Positive x offsets result in turning left and turning right for negative values.

IV. Results

Getting any decent measurements from steering the Balanduino is made very hard by the way the control system is implemented. The steering is done in two-dimensional positional offsets which are measured with the accelerometer and wheel positions. This makes the steering very blunt and sending the duplicate control signals can yield different results.

With the old software layout of three threads the program sent 5-8 control signals per second to the Balanduino. This was drastically improved to 48-56 control signals per second with the new layout of one video stream thread and one main thread.

The performance objective of the system was determined to be the speed which the robot gets around the track and how well it follows the path. A way to quantify this performance of the system was to record the position 1 offset, the speed and time during a lap around the track. Three tests were done with different speed settings. First one had constant speed, second one had high maximum speed and low minimum speed and the third one had high maximum speed and high minimum speed.Below in Table I the mean and the standard deviation were calculated for the position 1 offset. The plot below in Figure 9 shows the speed signals sent to the Balanduino for the same three test. Another test was done to compare the position 1 offset to the turn control signal during one lap. The result of this test can be seen Figure 10 and 11. Important to note here is that small values in mean and deviation are desirable.

TABLE I: Data is the position 1 offset (road to middle of picture, θ_1) from three tests with different speed settings. σ is the standard deviation and the mean is taken from the absolute value.

Test	1	2	3
Mean [cm]	1.38	1.42	2.08
σ[cm]	1.79	1.86	2.72
lap time [s]	17.56	17.97	16.48

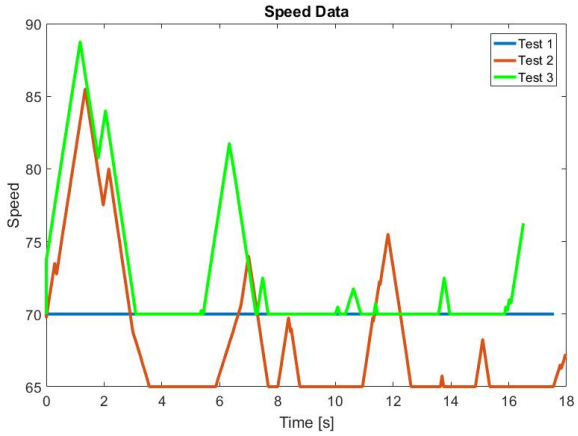


Fig. 9: Plot of speed references from the three tests with different speed settings over one lap. Speed has unit % of max-speed. Sampled: 50Hz.

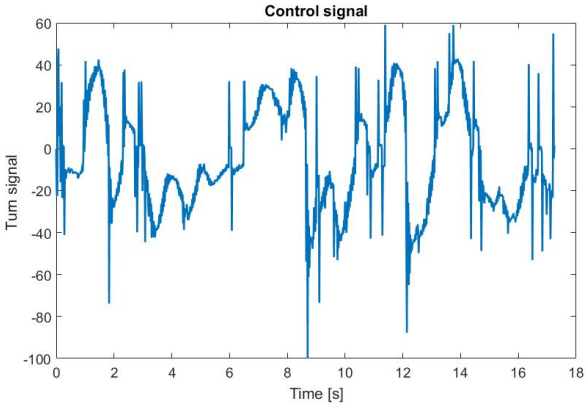


Fig. 10: Plot of control references from one lap. Control turn has unit % of max-turn. Sampled: 50Hz.

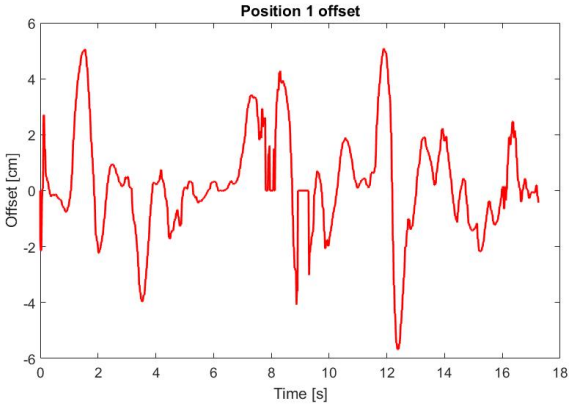


Fig. 11: Plot of position 1 offset from one lap. Sampled: 50Hz.

V. Discussion

As can be seen in Table I the faster the Balanduino goes the worse the line following becomes. This was also noted

when first testing the turning PID control. More weight on points further ahead meant it could go faster but also resulted in worse line following, sometimes taking shortcuts.

The plot in Figure 9 shows just how the speed control works. Previous failed tests also showed that constant speed was not bad because frequent speed changes affected the balancing which in turn affected the steering. Holding an even speed when turning was a hard problem to solve because the robot needs to recognize the exact curvature of the next turn to know the optimal speed. Instead a lower value was chosen that the robot was known to be able handle all turns on the path. The speed graphs might look a bit too nice, and they are, since this is the reference signals rather than the actual measurements, which are noisy.

Comparing the control signal and the position offset in Figure 10 and 11 a correlation can be seen. This is not surprising since parts of track with harder turns needs greater control signal which decreases precision. Sharp turns also make the robot turn ahead of time which increases the position 1 offset.

Since the color filtering for the computer vision depended on many different factors and had to be specifically designed for a setting it would be advisable in the future to design a calibration function where the Balanduino is placed along the track and analyses the tape color to create a filter. This calibration process seemed out of the scope of this project.

There have been several problems with the robot from the start that slowed down the implementation of the project. The serial port connection and a crystal component were loose and had to be soldered on again. One wheel was assembled the wrong way which made the balancing unstable since the gyro didn't sense the right rotation. The Bluetooth dongle placed in the Balanduino was defect which made it difficult to test how well the steering speed and turn worked. Mostly these problems were fixed or alternative solutions were found.

Another issue has been the unreliability of the robot in the way it interprets signals. The same signals can give two different outcomes on different runs. One reason being the amount of stored energy in the battery that power the motors. This has meant that parameter-tuning has been very hard so as to accomplish good control. Calibration has been important too since results could differ between two runs when nothing had been changed at all.

A. Further development

An early idea was to implement some kind of learning algorithm for the system. This was thought to be done by using ILC, or iterative learning control. The way this is done is by repeatedly driving the same track over and over, and feeding a learning algorithm with the input to the system as well as the error output. The algorithm thus learns what inputs yield the best minimization of error and converges to the optimal reference value.

This algorithm is formulated in the equations below.

$$y_k(t) = T_c(q)y_d(t) + T_c(q)u_k(t) \tag{9}$$

$$e_k(t) = y_d(t) - y_k(t) \tag{10}$$

$$u_k(t) = Q(q)[u_{k-1}(t) + L(q)e_{k-1}(t)] \tag{11}$$

Here T_c represents the closed loop system transfer function, q the time shift operator and Q & L the filters that need to be designed. The L_q filter can be chosen on a few different bases, one of them being the Model Based filter, which is basically the inverse of the closed loop transfer function. This however can cause an anti-causal function to arise, which means that the Q_q filter must be chosen as a low pass filter. However, this kind of algorithm is computationally and mathematically heavy. Therefore attempts will be made to implement this after this report is due.

A few different problems with implementing this is the variable nature of the process. Since the process turns differently and follows the track in other ways during two runs, the ILC can not solely rely on the previous runs' error values.

This might cause problems when trying to reach convergence for the algorithm.

References

[1] da Silva Jr Airton R. "Design and Control of a Two-Wheeled Robotic Walker". MA thesis. May 2014. URL: http://scholarworks.umass.edu/masters_theses_2/79.

[2] Vikram Balaji et al. "Optimization of PID Control for High Speed Line Tracking Robots". In: 76 (Oct. 2015). DOI: 10.1016/j.procs.2015.12.329.

[3] *How to make a line following robot.* Dec. 15, 2017. URL: https://diyhacking.com/make-line-follower-robot/.

[4] *Increasing Raspberry Pi FPS with Python and OpenCV.*. Dec. 5, 2017. URL: https://www.pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/.

[5] *Website for the Balanduino.* Dec. 18, 2017. URL: http://www.balanduino.net/.

[6] J. G. Ziegler and N. B. Nichols. "Optimum Settings for Automatic Controllers". In: (Nov. 1942). URL: goo.gl/Zj1Ckd.

APPENDIX

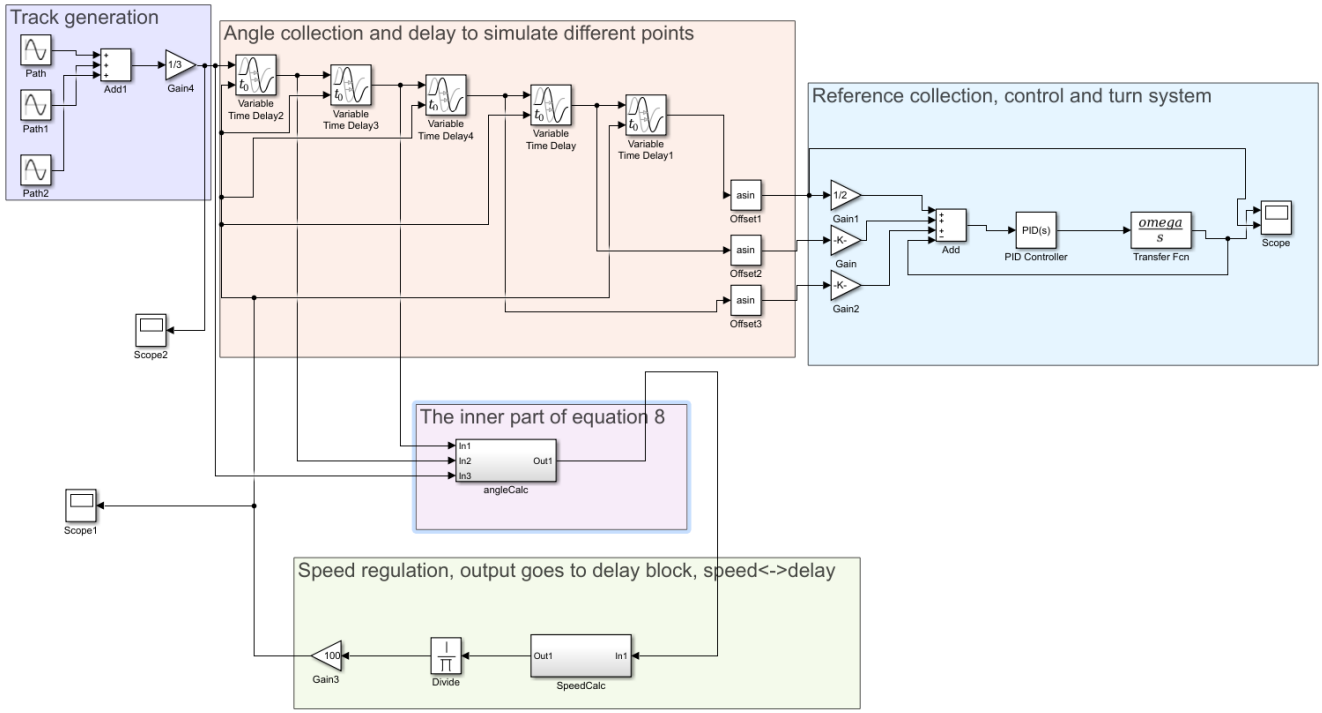


Fig. 12: Simulink model