Projects in Automatic Control 2021

Charlotta Johnsson
Editor

LUND
UNIVERSITY

Department of Automatic Control

Technical Report TFRT-7666
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 2022

Preface

The Department of Automatic Control at Lund University annually gives a project course in Automatic Control (FRTN40). The course is given at the advanced level (7.5 ECTS credits), and the students work in small teams to achieve a common goal. The projects typically involve a real-world estimation or control problem with relevance to industrial or other applications. In this course, the students get an opportunity to explore implementational aspects of concepts they have learned in previous control systems courses. With a faculty member or doctoral student as an advisor, the groups independently formulate an objective and an associate time plan. Subsequent activities typically involve modelling, controller design, implementation, documentation, and verification. The students present their work through two feedback seminars, an oral presentation, a demonstration session, and a written report. The reports of the 2021 edition of the course are presented in this booklet.

This year, the cohort consisted of 23 students, working in teams of 2-5 persons. The course included 6 projects; Panda robot, Crazyflie quadrotor, Slimdog car, Bluelining robot, Brain computing interface, and Ball balancing robot. Every single group managed to perform successful and satisfactory real-time experiments to generate the final experimental results ready in time for the demonstration in January.

Doctoral students Martin Gemborn-Nilsson, Julian Salt, Zheng Jia have served the course as project advisors, together with support from guest Tihomir Zilic and Professor Anders Robertsson. We would also like to thank our research engineers Leif Andersson, Anders Blomdell and Anders Nilsson who have supported the groups throughout their projects. Finally, we would like to thank Mika Nishimura for her help with student registration and related matters.

To find out more about the course, please visit http://www.control.lth.se/course/FRTN40.

Lund, January 2022
Charlotta Johnsson, Course responsible FRTN40

Contents

Object positioning with Panda Robotic Arm

Elisabeth Agerman[1] **Iván Arenas Kelbelova**[2] **Michelle Atlas**[3] **Jakob Bringman**[4]

Project Advisor: **Julian Salt Ducaju**

[1]el8232ag-s@student.lu.se [2]ivanarenask@gmail.com [3]mi7000at-s@student.lu.se
[4]ja5658br-s@student.lu.se

Abstract: The main objective of this project was to insert a piston into a hole using a collaborative PANDA Robotic Arm, starting at an arbitrary position. The robot was controlled with model predictive control (MPC) on a higher level and impedance control on the lower level. The hole was detected using a camera for computer vision and so that the robot could position itself sufficiently close to the hole. A search algorithm was then used to find the exact position of the hole and insert the cylinder. The result was a program for the robot arm that allowed the robot to start in an arbitrary position, except for some particularly difficult position or joint angles, and use MPC to calculate a path to the desired position above the hole and move the tool tip point to that position. In that position, the computer vision detects the circles and calculates the coordinates for the circle in the x and y axis and the robot then moves to a point slightly above the found coordinates. From that position, a search algorithm starts that makes the piston move in circles until the hole is found and the piston is inserted. The whole program is performed with a working impedance control. Further improvements and future applications are discussed as well as the choice of using Cartesian coordinates and why ROS was not used as first intended.

1. Introduction

This project aimed to control a panda robotic arm safely with impedance control while performing an insertion task. The task was to insert a cylinder attached to the robot manipulator in a cylindrical hole. The robot arm should be able to start from any arbitrary position and move it above a box with a cylindrical hole, identify and locate the hole with a search algorithm and finally insert a cylinder attached to the robot manipulator. Tasks like this are common in industrial settings. By implementing impedance control this could increase safety for tasks where human interaction with the robot is necessary. The control of the robot was largely based on the results in the paper "Model predictive control for real-time point-to-point trajectory generation"[3].

2. Modeling

For this application, it was important to use a control method that not only takes the current states into consideration, but also anticipates future events. Without this feature, the robot might collide with items in the room or itself. Generic PID controllers were therefore deemed inadequate since they are unable to represent the behavior of complex dynamical systems. Based on the results in [3], it was instead deemed appropriate to use model predictive control (MPC) to control the robot. The main advantage of the MPC is that it can optimize a timeslot while taking future slots into account. To use this method, it was necessary to define a model of the system, an objective function, as well as state and input constraints. Using a linear model simplifies calculations, however, robot movements are generally nonlinear. This trade-off was deemed reasonable for the applications and tasks of this project.

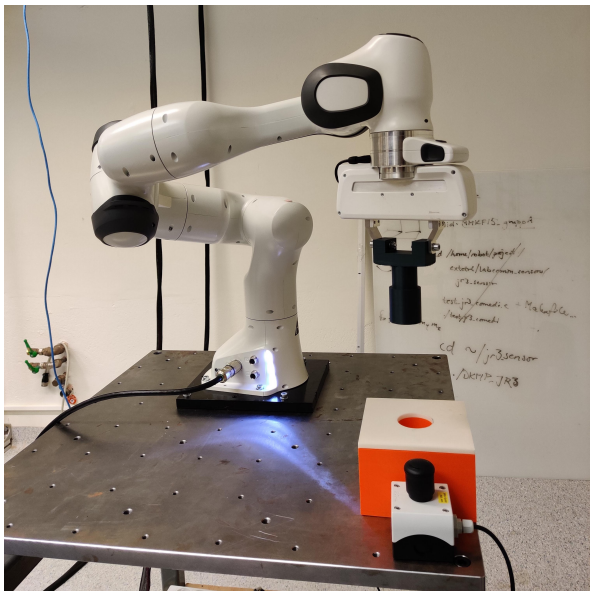

Figure 1. Setup of workstation and panda robotic arm with cylinder attached to the end effector.

The states of the system are defined as the Cartesian positions, velocities and accelerations

$$x = [x, \dot{x}, \ddot{x}, y, \dot{y}, \ddot{y}, z, \dot{z}, \ddot{z}]^T.$$

The final state was defined as the final and desired Cartesian position and zero velocity and acceleration: $\zeta = \zeta_f$, $\dot{\zeta} = 0$ and $\ddot{\zeta} = 0$. The derivative of the acceleration, $\dddot{\zeta}$, acted as the

control output only for calculation purposes.

2.1 Discretization

In the MPC framework, the space-state kinematic system was used to estimate future states. The continuous-time linear model used is a double integrator, see equation 1 and 2, being the matrices in equation 2 for each of the three axis. Since the trajectory is set in the three axis the system will result in one matrix for each one.

$$\begin{aligned} \dot{x}_c(t) &= Ax_c(t) + Bu_c(t) \\ y_c(t) &= Cx_c(t) \end{aligned} \tag{1}$$

The matrices A and B were defined as block diagonal matrices and are calculated as $A = \text{blkdiag}([\widetilde{A}, \widetilde{A}, \widetilde{A}])$ and $B = \text{blkdiag}([\widetilde{B}, \widetilde{B}, \widetilde{B}])$, where $A \in \mathbb{R}^{9\times9}$ and $B \in \mathbb{R}^{9\times3}$. The matrices were defined by the components,

$$\widetilde{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \widetilde{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2}$$

Assuming a sample period of h, the discrete-time system was obtained with the zero order hold (ZOH) sampling method which means that the output is constant until the next obtained sample, resulting in

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) \\ y(k) &= Cx(k) \end{aligned}. \tag{3}$$

The matrices for the discretized model concerning the controlled states can be calculated as block diagonal matrices as above, resulting in equation 4,

$$\widetilde{\Phi} = \begin{bmatrix} 1 & h & h^2/2 \\ 0 & 1 & h \\ 0 & 0 & 1 \end{bmatrix}, \widetilde{\Gamma} = \begin{bmatrix} h^3/6 \\ h^2/2 \\ h \end{bmatrix} \tag{4}$$

where $\Phi \in \mathbb{R}^{9\times9}$ and $\Gamma \in \mathbb{R}^{9\times3}$.

These matrices was used to estimate future states from each MPC calculation to decide which is the trajectory that minimizes the cost function.

3. Electro-Mechanics

All required hardware and parts were already built before the project start. The box as well as the piston had previously been 3D-printed. The main gadget in this project was the Panda robot, see Figure 1. The panda robot is a collaborative robot with seven degrees of freedom and torque sensors in all axes. It is specifically designed to assist humans but can accomplish programmed tasks on its own too. The robot arm has a path deviation that is less than 1.25 mm and has a collision detection time that is less than 2 ms. The robot uses positional encoders with 14 bit resolution and torque sensors with 13 bit resolution. The joint electronics operates at 1 kHz. The robot was already connected to a computer so no further setup was required.[7]

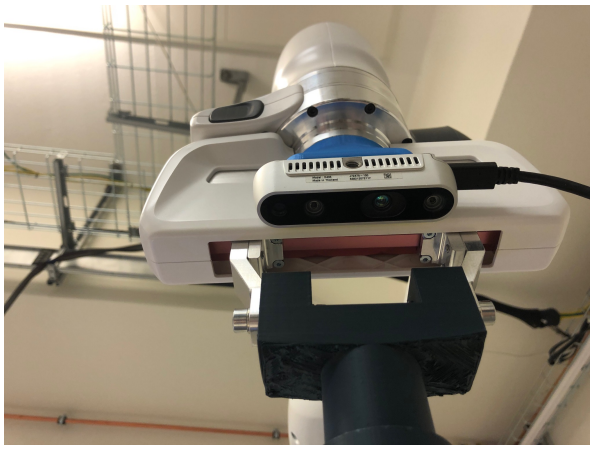

Figure 2. RealSense camera mounted to the robot.

An Intel RealSense Depth Camera D435 was also used, see Figure 2. The camera consists of two imagers, an IR projector and an RGB module. It has a relatively wide field of view, making it suitable for robot navigation and object detection. One disadvantage with the camera is that the minimum depth distance needed to accurately determine the distance is approximately 28 cm. [1]

4. Control

For the initial movement, the robot was controlled by a higher level Point-to-Point planning MPC and a lower level impedance controller. The MPC produces the desired position which is acted upon by the impedance controller to produce the torques to the joints of the robotic arm, see Figure 3. In the schematic, r_{goal} is reference signal of final position, $r_{desired}$ is reference signal of next desired position, e is the error between the next desired position and the current position, u is the control signal and y is the output signal. This control structure takes advantage of the optimal path produced by the MPC while allowing the robot to be designed as a mass-spring-dampener system. The impedance controller therefore allows for a safer interaction with the robot during operation from both a human-robot interaction and a robot-object interaction.

This section describes the controllers further and the strategy used for moving the cylinder into the hole of the box. The impedance controller was present during all movements for the reasons discussed above.

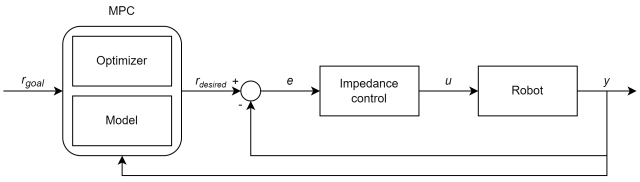

Figure 3. Schematic of point-to-point planning using MPC as higher level control and impedance control as lower level control.

4.1 State machine

The task of moving the cylinder, from its starting position into the hole of the box could be divided into a number of subtasks. Therefore, a state machine is considered the best solution for keeping track of each subtask, where each subtask is represented by a state.

A benefit of the state machine is that the states are decoupled and can be individually developed and tested. A representation of the state machine can be seen in Figure 4. The state machine consists of five states, determining what control strategy to be used. Impedance control is used as low-level control in all states in combination with different high-level controls.

The first state executes the combined MPC-impedance controller, see Figure 3, to produce an optimal trajectory from any starting position to a known position r_{goal}. Initially r_{goal} is a known position above the box. When this position is reached, a transition to the *Cam* state is triggered.

The *Cam* state uses the camera to locate and calculate the position of the hole. The reference signal r_{goal} is updated and a transition back to the *MPC-impedance control* state is triggered. When r_{goal} is once again reached, a transition to the *Approach* state is triggered.

The cylinder should now be placed above the box, close to the hole and is in this state lowered purely by impedance control until contact with the box is registered. The cylinder is considered being in contact with the box after a number of iterations where contact has been detected. The desired z-position is decreased in every iteration and therefore produces an error e_z along the z-axis.

The *Search* state is then transitioned into, where the search algorithm is performed until the hole is located. The hole is considered located when there is no contact with the box and the error e_z is bellow a threshold $e_{threshold}$. See Subsection 4.4 for an in-depth explanation of the algorithm used.

Once the hole has been located, a transition to the *Insert* state is triggered where the cylinder, like in the *Approach* state, is lowered with impedance control. If contact with the wall of the hole is registered, a small circular movement in the global xy-plane is also performed until there is no contact with the wall. When the cylinder makes contact with the bottom of the hole, the task is completed and the *end* state is triggered and the program is finished.

4.2 MPC

The model predictive control was being used to generate a Point-to-Point trajectory. In this case the MPC was used in an offline configuration to avoid the use of multi-threading, a method too complicated to implement when weighted against expected benefits. For any initial given position the state is read and the information is passed to the optimization solver to get the next point of the trajectory. This operation is repeated several times until the final fixed desired position is achieved. The position calculated by the MPC will be used as an input to the impedance control that will determine the values of each joint torque.

As it is explained in the modeling subsection, the MPC was working in Cartesian position coordinates (due to the increasing difficulty the orientation will not be solved with the MPC).

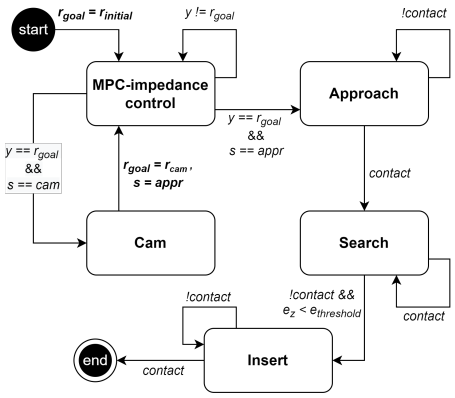

Figure 4. Schematic of the state machine.

Therefore the optimization equation and the restrictions had to be expressed regarding to this reference system.

Optimization The model predictive control is based on looking for the solution to an optimization problem in a fixed time horizon fulfilling a set of restrictions. A common form of the optimization function is a first term penalizing both the position error and the control signal for the first T time steps and a second term penalizing only the position error in the last steps to minimize this final position error. Following this structure the optimization problem has been structured as follows:

$$\text{minimize} \quad \sum_{t=0}^{T-Tf-1} (x_t^T Q x_t + u_t^T R u_t) + \sum_{t=T-Tf}^{T} (x_t^T Q_{final} x_t)$$

$$\begin{aligned}
\text{subject to} \quad & x_{t+1} = \Phi x_t + \Gamma u_t, \\
& |u_t| \leq u_{max}, \\
& |u_{t+1} - u_t|_\infty \leq S, \\
& |x_t(1)| \leq x_{max}, \\
& |x_t(4)| \leq y_{max}, \\
& |x_t(7)| \leq z_{max},
\end{aligned}$$

Being Q, R and Q_{final} weighting matrices; T and T_f steps intervals; $x(1)$, $x(4)$ and $x(7)$, the values of the Cartesian coordinates x, y and z, respectively; S the slew rate; and u_{max}, x_{max}, y_{max} and z_{max}, upper limits for the control action and Cartesian coordinates.

Regarding to the parameters in the optimization expression most of them were fine tuned. The optimization was set to 15 time steps in a fixed time of 4 seconds, between those the initial 12 penalized both the control action and the position error, and in the final three only the position error was penalized. The weighting matrices Q, R and Q_{final}; were fine tuned experimentally trying to get a smooth movement over the whole trajectory and a relatively small position error at the desired final position.

Regarding to the constraints, the first one is set in order to estimate the future next steps, this way the MPC is able to estimate the best path to achieve the final position. The second and third are restrictions to the control action maximum value and its change for one steps to another. Since the actual mechanical descriptions should be set on the robot joint absolute

positions, velocities or accelerations limits; there is not apparent accurate way to tune the second restriction. The same occurs with the third one, the slew rate constrain is mostly use to avoid discontinuities when applying completely different control actions in a small period of time so in this case is difficult to have a correlation between joint and Cartesian parameters. That being said this parameters were tuned as the ones in the minimizing expression, looking at the robot performance over different situations. The last three restrictions were set as spacial physical limits in the three axis x,y and z. These restrictions avoid that the trajectory of the robot goes out of a safety space delimited.

Constraints The use of Cartesian position was preferred because positional constraints are more straight forward to set. In this case the trajectory should be constrained within the cubic volume over the surface of the table. This is achieved as simple restrictions over the different axes. By using Cartesian coordinates, there is no way to constrain either the joint velocities or accelerations, since the controller is designed in Cartesian space. Another problem is restricting the robot from colliding with itself, which is a non-convex restriction. The MPC relies on quadratic convex optimization and these types of restrictions can therefore not be solved by the controller. The generated points from the MPC that are within a radius r from the center of the robot, are therefore moved to the circular arc around the robot in a perpendicular direction to the generated path.

4.3 Impedance Controller

The impedance controller was used to give the arm responsiveness against external forces that can appear within the work environment. In practice this means that impedance control can help to avoid rigid collisions since it gives a flexible response. This prevents collisions that could be dangerous for both the robot and the users around it. The impedance controller in the project has been set to calculate the desired joint forces for any of the states in the state-machine.

As the dynamic model of the robot was not being modified, using the software's default dynamic model for the arm; the cylinder attached to the hand is detected as an external force. Then, the impedance controller is compliant with this force and a precision error is being introduced. This error was solved by the search algorithm in the last stage.

4.4 Search algorithm

The goal of the search algorithm was to locate the hole. A simple spiraling movement was implemented, see Figure 5. This ensures that the hole is located from any starting position that is closer to the hole than the edges of the box. The spiraling movement is defined by increasing the radius proportional to the angle,

$$r = k\theta. \tag{5}$$

While executing the movement the robot is continuously pushing against the box by setting the desired position along the z-axis below the surface of the box. The normal force from the box is detected by the robot until the cylinder is placed above the hole. This was used as a condition for finding the hole.

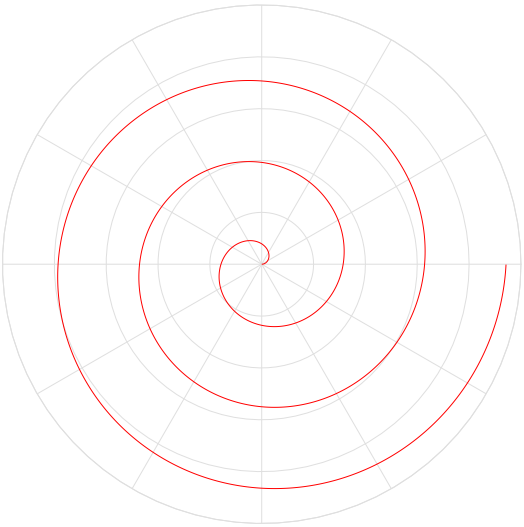

Figure 5. Spiraling movement of the search algorithm.

Since the cylinder may occasionally lose contact with the box due to the inaccuracy of the impedance control the positional error along the z-axis is also checked. When the calculated error is below a threshold and no contact with the box is registered, the hole has been found.

5. Implementation

The implementation of the different control systems were done separately and was intended to be put together using Robot Operating System (ROS), but was instead integrated in one single program due to ROS being too time consuming to initiate in relative to the benefits of using ROS. The state machine and impedance controller was implemented using the C++ library libfranka [4] with methods for controlling the panda robot. To avoid going back and forth between Cartesian and joint coordinates, all programs solely use Cartesian coordinates as input and output. The computer vision was written in Python and then translated to C++ and the libraries used for the computer vision were realsense and OpenCV.

5.1 MPC

The model predictive control returns a set $T_{horizon}$ of 15 points. In order to obtain a smooth trajectory in between points, linear interpolation was used to create multiple sub points. Since the movement should be completed in a fixed time T_{limit} of 4 seconds, the time between each point is 0.267 s and the number of sub points is calculated as

$$n_{steps} = f_{robot} * \frac{T_{limit}}{T_{horizon}} \tag{6}$$

where f_{robot} is the 1 kHz frequency which the joint electronics operates at.

This results in a total number of 267 sub points in between each point generated by the MPC. The interpolation is calculated and stored offline after each call to the MPC for reduced computation time in the control loop. These points were then used as reference signals to the impedance control.

5.2 Impedance

The impedance controller was implemented using functions provided in the libfranka library. Two main constants could be modified in order to obtain the desired collision behavior, the stiffness and the damping. The stiffness defines the input of force needed to achieve a displacement of the robot, and the damping is related to the variation of force needed depending on the velocity of the disturbance.

To control the behaviour of the panda robot there are two thresholds for collision behaviour: an "advice" threshold and an "error" threshold. In this case the values were set to max values in order to be responsive in a wider range of forces. This is true except for one of the advice parameters that will be used to detect the contact in the z-axis.

5.3 CVXGEN

The MPC was implemented in C using CVXGEN, a website that generates code for small, QP-representable convex optimization problems [2]. By inputting the estimated parameters, variables and constraints, previously modelled in Section 2. and defining the cost function as described in Section 4.2, CVXGEN outputs a solver for the control problem. The C code could then be linked to the main C++ code so that it could be used as a high-level Point-to-Point planner.

5.4 Computer Vision

To find a suitable starting position for the search algorithm, see Section 4.4, computer vision was used. Python was used to write the script, which includes the pyrealsense2 library [6] that handles the connection to the Intel RealSense D435 camera used and Open CV library [5] that is a library for computer vision. The depth and color stream from the camera was used to identify circles in the frame. To achieve this, the method Hough circle transform was used [8]. By applying the transform to gradient information of edges in gray-scale pictures, the circle parameters can be determined. To avoid incorrectly identifying other circles than the desired hole, the radius was limited so that only circles of the specified size would be detected. The detected circle and its center point was drawn in the image and can be seen in figure 6. The square shows the measurement points which the average distance to the box is based on. After identifying a hole, the coordinates for the hole was calculated by translating the number of pixels to a distance. At first, the distance to the box's surface was calculated using the cameras depth functionality but due to the camera being more inaccurate compared to the accuracy to the estimated position of the robot and the lack of need to be exactly on the surface of the box when starting the search algorithm, a fix distance was used instead. With this information, the robot should be able to position itself sufficiently close to the hole for the search algorithm to succeed.

5.5 ROS

Since different programming languages were used for different parts of the project, ROS was intended to be used for communication between the robot and the computer vision. ROS already includes software libraries and tools that aid in building robot applications. By building a package that included the program for the state machine, the MPC and the computer vision, it would not have been necessary to write all programs in the same language. ROS works as a multi-threaded program with built in communication since it can be used to launch a number of different nodes where each node run a program separately. The nodes can subscribe to each other and thus be notified when a message is sent from another node. This makes ROS a helpful tool when working on a project with different parts that needs to work together. However, the work to learn ROS and implement it was complicated and time consuming and it was realized that an easier option would work and have an equally good result due to the relatively low complexity of the project and thus it was decided to not use ROS. The option of using a Python C++ API to run the computer vision code (Python) in the code for controlling the robot(C++) was discussed but the choice was made to instead translate the computer vision code from python to C++ and call it from the controller code.

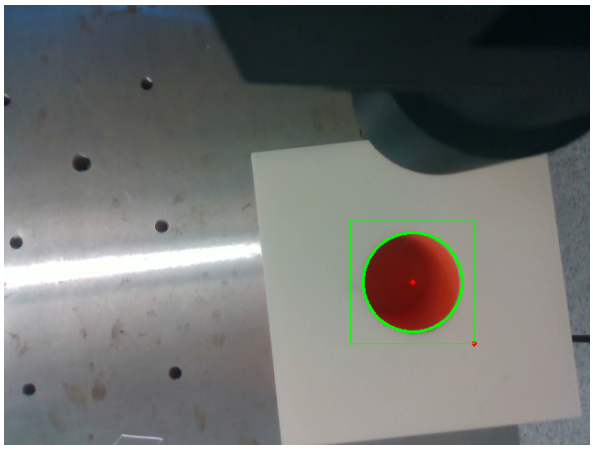

Figure 6. Image generated from the camera.

6. Results

Combining all of the control methods and implementations resulted in a program for the robot arm that allowed the robot to start in an arbitrary position, aside from some particularly difficult position or joint angles which leads to error due to exceeded joint limitation. This limitation could be derived from the use of Cartesian coordinates to control the robot, and the work to solve this would most likely be quite excessive and was thus excluded from the project. The program uses the designed MPC to calculate a path to the desired position above the hole and move the robot to that position. If the calculated path is inside the vertical cylinder designed to represent the robot body, the points within that cylinder is moved perpendicular to the path until it is outside of the cylinder to avoid the robot colliding with itself. This can be seen in Figure 7.

When the robot arm ends up in the end position of the MPC path, which is a point well above the box that is chosen to be a good position for the camera to be able to detect the hole, the computer vision starts. The program for the computer vision waits for a circle to be detected in the images from the camera, and uses the detected circle to calculate the distance from the robot to the circle in x and y coordinates. The robot is moved to the detected coordinates and to a z coordinate a short distance

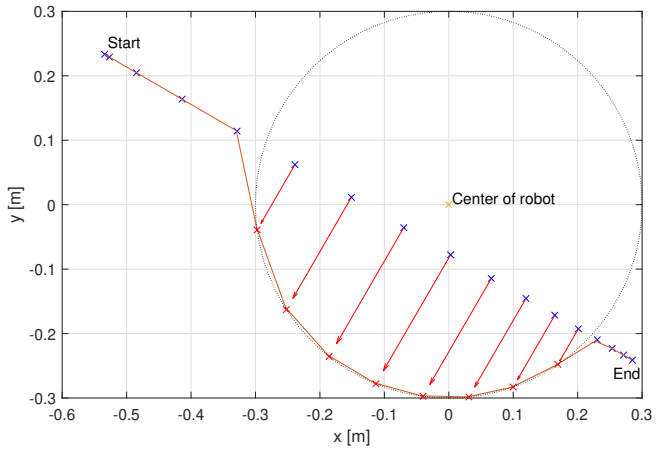

Figure 7. Points generated by the MPC in the xy-plane. Points within the circle, representing the robot, are moved to the arc (red points), perpendicular to the generated path.

above the box to ensure a good starting position for the search algorithm. From that position, a search algorithm starts that makes the piston move in circles until the hole is found and the piston is inserted, see Figure 8. The whole program is performed with a working impedance control which protects the robot and the surroundings from harm due to collision. The impedance control is also central in the search algorithm to detect when the hole is found. Depending on the robot's starting position, the program will take slightly more than a minute to execute, with the search algorithm being the most time consuming task, see figure 9.

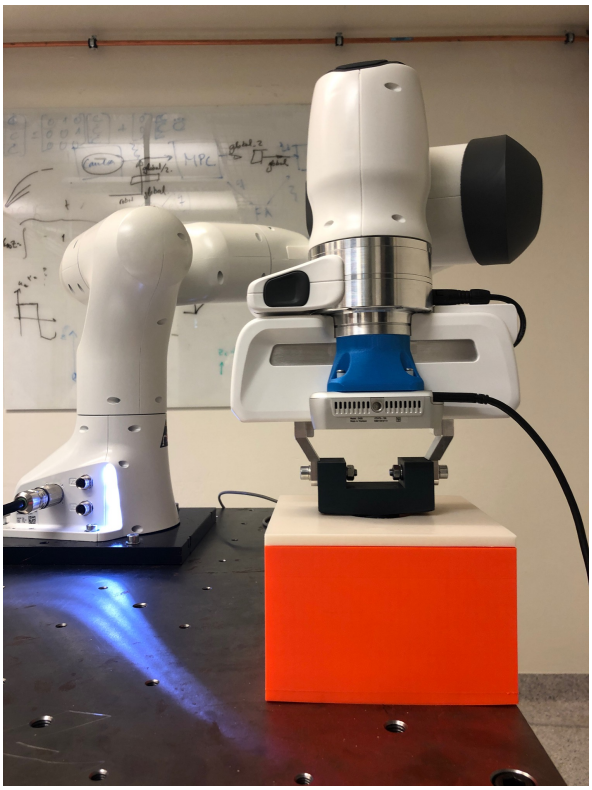

Figure 8. Inserted piston.

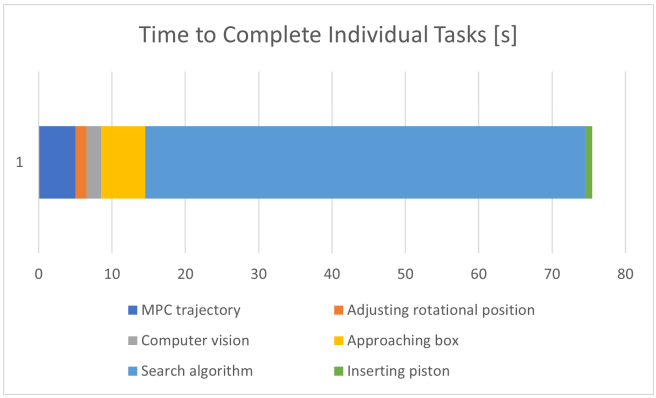

Figure 9. Time table of individual tasks.

7. Discussion

While the project was successful, some further improvements could have been made. In the initial phase of the project it was discussed whether to use Cartesian coordinates or joint coordinates in the MPC. To simplify the setting of restrictions, the Cartesian coordinates were chosen. It was then possible to for example limit the movements in the z-direction to prohibit the robot from moving beneath the table. However, this resulted in complications regarding the joint limitation and when controlling the orientation of the tool tip. Since joint limitations were not considered when calculating the path using the MPC, this resulted in the robot sometimes aborting due to exceeded joint limitations. This could for example happen if the robot's joint angles in the start position were already close to their limit, and the closest way to reach the end position was to continue to rotate the joint until the limit was reached. This problem would have been avoidable if joint coordinates were used instead. Another disadvantage of the use of Cartesian coordinates is that it is only the tool tip position and not the tool tip rotation that is considered in the MPC. This limited the possible starting positions of the robot to the ones with an angle of the tool tip very close to the desired final tool tip position, which is with the piston in a vertical downward position. This was necessary to make sure the field of view of the camera covered the circle and the box. Small deviations from that rotational position could be fixed after the robot had reached the end position but a better and more advanced algorithm would be needed to be able to adjust larger deviations. For future work this would be a suggested improvement.

Another possible area of improvement is the camera code which is good enough for this project in a lab setting when the position and field of view of the camera is known and there is no other circles with similar size that can be detected by the camera. To create a more robust code for the computer vision that could handle more advanced environments, a number of different frames could be used to get a mean value of the x and y coordinates.

The initial thought was to use the depth sensor of the camera, as well as the x and y coordinates detected by the computer vision to calculate the distance from the tool tip to

the circle in all three axes and move to that position. However, since the depth value was not very exact and was not accurate on a distance closer than 28 centimeter, the results were not very reliable. Since using the depth sensor led to a worse result than just using a fix distance based on the estimated end position, it was decided that that was a simpler and more robust solution. However, if a more exact depth sensor is used or if the actual end position of the robot differs a lot from the estimated position, one improvement could be to implement the usage of the depth sensor instead of the fix value to calculate the distance in the z-axis and not only the x- and y-axis.

Furthermore, to make the robot's path following more accurate, the specific robot parameters should have been modified in the code. For example, the weight of the camera was added to the tool tip since the difference between the actual and estimated torque needed for joint movements could otherwise have led to deviation from the final position. However, to get even more accurate results, the camera's cord should also have been taken into consideration, as well as the shifted centre of mass after mounting the camera.

Further improvements could also have been made to lower the time needed for calculations and movements. It was previously mentioned that a decision was made to avoid multithreading, which would have led to shorter computational time. It is possible that a better trajectory might have been generated if the MPC was allowed to run continuously.

Regardless of the possible improvements, the end goal was achieved. Especially given the limited time and resources, the project can be said to have been successful.

References

[1] *Depth camera d435*. URL: https://www.intelrealsense.com/depth-camera-d435/ (visited on 2022-01-05).

[2] *Example: model predictive control (mpc)*. URL: https://cvxgen.com/docs/mpc.html (visited on 2021-12-06).

[3] M. M. Ghazaei Ardakani, B. Olofsson, A. Robertsson, and R. Johansson. "Model predictive control for real-time point-to-point trajectory generation". *IEEE Transactions on Automation Science and Engineering* **16**:2 (2019), pp. 972–983. DOI: 10.1109/TASE.2018.2882764.

[4] *Libfranka*. URL: https://frankaemika.github.io/docs/libfranka.html (visited on 2021-12-06).

[5] *Opencv*. URL: https://opencv.org (visited on 2022-01-03).

[6] *Pyrealsense2*. URL: https://intelrealsense.github.io/librealsense/python_docs/_generated/pyrealsense2.html (visited on 2021-12-06).

[7] *The robot system*. URL: https://www.franka.de/robot-system/ (visited on 2022-01-05).

[8] C. H. Transform. *H. rhody*. URL: https://www.cis.rit.edu/class/simg782/lectures/lecture_10/lec782_05_10.pdf.

Quadcopter collision avoidance algorithm

Jon Swedberg[1] **Felix Ghosh**[2] **Fredrik Sjöström**[3] **Ricardo Almeida Henriques**[4]

Project Advisor: **Zheng Jia**

[1]jo8344sw-s@student.lth.se [2]fe6046gh-s@student.lth.se [3]fr8272sj-s@student.lth.se
[4]ricardo.a.henriques@tecnico.ulisboa.pt

Abstract:
Drones have a high variety of useful applications today. In this project the aim is to implement collision avoidance for an autonomous flying drone while flying in a circle. The drone dynamic can be divided into actuation system, which includes the motors and propellers, the motion system, which includes dynamics equations and kinematics and the sensors. In order to do collision avoidance while flying in a circle and a coordinate system was needed with the center of the circle defined as (0,0). To get the drones position an embedded Kalman filter was used together with a flow deck, and to detected obstacles a Multi-ranger deck was used. The collision avoidance algorithm calculates the radius of the object by measuring the distance to it with the front sensor and adds a safety margin to the estimated radius of the object and finally goes in a circle around the object with this radius. After the avoidance the drone continuous it's original circular path again. The collision avoidance algorithm was implemented in *python* and is was tested for a 2 and 3 obstacle course while analysing the accuracy of the radius estimation and the placement of the center of the obstacle.

1. Introduction

The applications for automated drones are evolving as technology improves. One application is to take photos to obtain images of areas that are hard to reach for non-autonomous vehicles. One example can be found in Japan, where they used drones to map the structure of a volcano in order to run simulations to calculate when the volcano would erupt[6]. Another application is autonomous delivery. Amazon has published that they are going to use automated drones to deliver packages up to 2.3 kg to their customers in near future[1]. In both cases the drones need to fly autonomously, meaning that they need some way of detecting obstacles and more importantly, avoiding them.

The aim of the project is to try to implement autonomous collision avoidance for a flying drone. The drone is programmed fly in a circular trajectory and be able to detect and avoid static obstacles placed on the trajectory, as seen in figure 1. After avoiding the obstacle it should also find it's way back to the circle trajectory. The drone that will be used to accomplish this task is a Crazyflie, a very small and light drone with user-friendly interfaces.

It is important to look at the applications of what you make. Drones can be used for helpful things to humanity as for example, helping to predict volcanic eruptions, but they can, and have also been used frequently in warfare. Of course this is a very small project, and will not create groundbreaking new algorithms to be used for any such extreme applications, but it is important to consider the ethical implications of potential uses of technology, when developing new innovations.

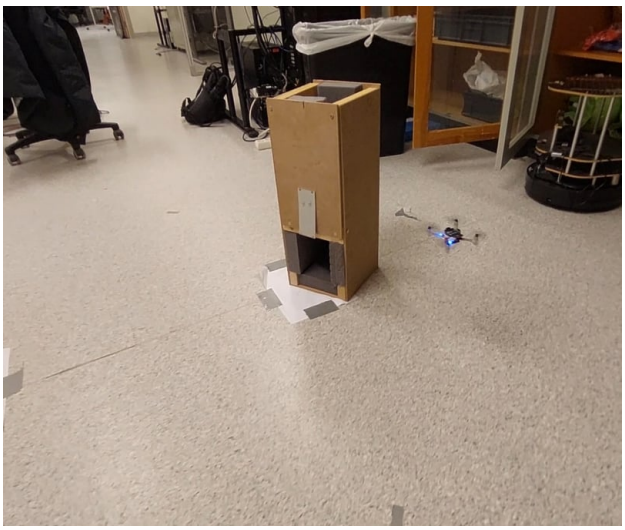

Figure 1. Project Outcome - Crazyflie running in the obstacle path

2. Modeling

The drone is supposed to follow a pre-defined circular path with a constant velocity, avoid obstacles and find its way back to the circle again. For the purpose of this project, the velocity used was $0.2m/s$ due to the fact that it was the velocity used in paper [3] which gave the best results without the drone becoming unstable. For the drone to find its way back to the same circle as before, a coordinate system is needed to keep track of both the drone itself, as well as the obstacle it has to avoid. In order to be able to do this, an estimate of the current position of the drone on the coordinate system is needed.

In addition to this we had to model the circular trajectory reference and determine a method for detecting and avoiding the obstacles.

2.1 Drone Dynamics

Before starting with the collision avoidance method we first identified the dynamics of the quadcopter. It can be divided into 3 subsystems [5]: the actuation system, the motion system and the sensors.

Actuation system In this system, the model of each of the four rotors consists of the motors, the propellers and the model of the allocation of control. The combination of forces and moment exerted by each one of the four rotors results in the thrust force, F_p, and the thrust moment, M_p, which are responsible for the movement of the drone. In this way, we consider the model of the system of actuation composed of:

Motors model: The motors are modeled as direct current motors, being controlled by the supply voltage V_m. The motor stator can be modeled as a circuit RL traversed by a current I, where the Joule losses are represented by the resistance R_m, the magnetic field losses are represented by the inductance L_m, and the electromotive force produced is represented by the multiplication of the angular velocity Ω by a counter-electromotive constant K_e. The rotor (moving part of the motor) rotates at an angular velocity Ω and has a polar moment of inertia J_m, the torque created is modeled as the product of the angular velocity by a torque constant K_t, and a friction force is modeled by the product of the angular velocity and a friction constant B_m. Thus, the equations that represent the engine model, where Q represents the moment associated with the propeller, are:

$$\dot{I}_n = \frac{1}{L_m}(V_m - R_m I - K_e \Omega_n) \tag{1}$$

$$\dot{\Omega}_n = \frac{1}{J_m}(K_t I - Q - B_m \Omega_n) \tag{2}$$

Assuming that the voltage received at each motor is equal to V_m, this can be related to the battery voltage, V_{bat}, by:

$$\delta_n = \frac{V_{mn}}{V_{bat}} \tag{3}$$

Where δ_n is the normalized voltage received by each motor. Since the movement of the motors is much faster than the movement of the drone, they can be modeled as a static gain. The normalized voltage will be used for modeling the motors through the following linear relationship:

$$\delta_n = K_\Omega \Omega_n \tag{4}$$

Propellers model: The system corresponding to each propeller is modeled using the force and momentum coefficients, K_T and K_Q, which multiplied by the square of the angular velocity of the propeller allow us to obtain the torque, T, and the momentum, Q , resulting from its rotational movement. These coefficients depend on the characteristics of the propeller used. Thus, with Ω_n being the speed of rotation of the propeller connected to the engine n, and T_n and Q_n the torque being the resulting momentum, respectively, the equations can be written as:

$$T_n = K_T \Omega_n^2 \tag{5}$$

$$Q_n = K_Q \Omega_n^2 \tag{6}$$

Allocation of control model: Considering the configuration in $\times$ of the quadcopter, and the different possible motion manipulations previously described, the resulting propulsion forces and moments can be obtained by the following equations:

$$F_p = \begin{bmatrix} 0 \\ 0 \\ -\sum_{n=1}^{4} T_n \end{bmatrix} \tag{7}$$

$$M_p = \begin{bmatrix} [(T_1 + T_4) - (T_2 + T_3)] L \cos 45 \\ [(T_1 + T_2) - (T_3 + T_4)] L \sin 45 \\ -Q_1 + Q_2 - Q_3 + Q_4 \end{bmatrix} \tag{8}$$

Motion system

Dynamics: The dynamics equations relate the forces exerted on the body with its acceleration and inertia. The translational dynamics, equation (9), and rotational dynamics, equation (10), of the drone are described by the following equations:

$$m\dot{v} = -\omega \times mv + F_p + F_a + F_g \tag{9}$$

$$J\dot{\omega} = -\omega \times J\omega + M_p \tag{10}$$

Where m and J are, respectively, the drone's mass and inertia matrix, v and ω are the linear and angular velocity vectors of the local frame relative to the inertial frame, expressed in the local frame. The external forces and moments acting on the quadri-rotor come from propulsion (vectors F_p and M_p previously described in equations 7 and 8, from aerodynamics (which include wind disturbances), F_a, and of gravity, F_g. There is also an aerodynamic moment M_a, however this is negligible compared to the propulsion moment M_p.

Kinematics: Kinematics equations relate the velocity of the body to its position. The linear and angular velocities of the quadcopter can be described in the inertial frame by:

$$\dot{p} = Rv \tag{11}$$

$$\dot{\Phi} = S\omega \tag{12}$$

where $p = [p_N, p_E, p_D]^T$ and $\Phi = [\phi, \theta, \psi]^T$ are the position of the center of mass and the attitude of the quadri-rotor relatively to the inertial frame, respectively. The matrix S is given by:

$$S = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi / c_\phi & c_\phi / c_\theta \end{bmatrix} \tag{13}$$

Where $s_\alpha = sin(\alpha)$, $c_\alpha = cos(\alpha)$ e $t_\alpha = tan(\alpha)$.

2.2 Coordinate system

All position data comes from the drone's sensors which means that we do not have any external information. This needs to be taken into account because the drone sensors do not give us absolute position and they are subject to errors. Therefore, the coordinate system and position of the drone has to be based on initial values and estimations. To do that we use the predefined Kalman filters that we obtained from the *cflib* which is the crazyflie library for python.

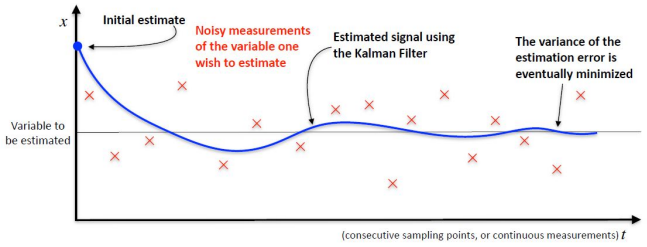

Figure 2. Kalman Filter Process [2]

2.3 Kalman Filter

The drone's sensors give us a measurement of the position of the drone. However, this one is subject to errors. Due to this, we used a Kalman filter which has the objective of minimizing the effect of the noise in the measurements by minimizing the tracking error. The principle of the Kalman filter is to gather information known about the system model with the information received by the sensors, in a recursive process, and it is verified that the value converges to the true value. The algorithm consists of two steps: first, in the prediction step, the Kalman filter estimates the states and its uncertainties, and in the next step, after receiving the next corrupted sensor measurement by noise, updates state estimates using a weighted average, as we can see in figure 2.

2.4 Circular path reference

The circular reference can be described by the following equation:

$$(x - x_0)^2 + (y - y_0)^2 = R^2 \tag{14}$$

where x_0 and y_0 are the center point of the circumference, x and y are the drone current position and R is the radius of the circumference.

2.5 Collision detection

To detect an object placed in the path we use the information gathered from the Multi-ranger deck. We assume that the first time the drone detect an object the edge is detected and that the obstacle can be represented as a circumference with radius R_{obj} and with the center placed on the trajectory of the drone. With these assumption when an object is detected we will have a situation as shown in figure 3, which can be seen as two triangles, one consisting of the following three points: The center point of the circular path the drone is taking, called C, the drone current position called D and the point where the obstacle is detected, called O. The other triangle is represented by points C, O and the center point of the obstacle, called OC. We want to determine R_{obj} which is the distance from the point OC to the point O.

We start with the first triangle. The angle at D is called α and is calculated by $90 - (desiredyaw - actualyaw)$ The angle at C is called θ and is calculated by first computing the inverse of the tangent of the y position of O divided by the x position of O, and then doing the same for D and taking the difference between them. Finally the angle at O is called ϕ and is equal to $180 - \alpha - \theta$.

For the second triangle we first make use of our assumptions. The assumption that the object is circular with center on

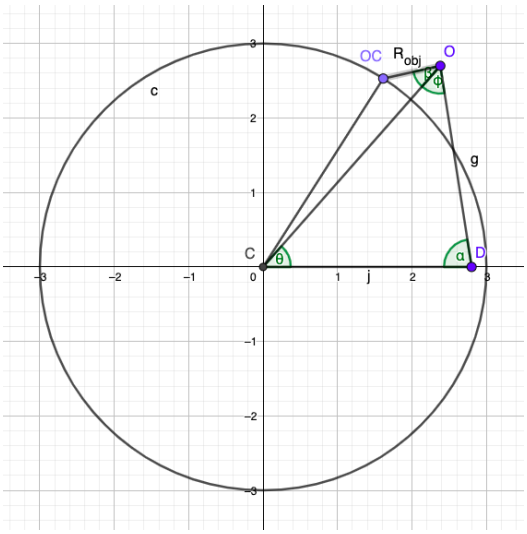

Figure 3. Collision Detection

the path the drone is taking means that the distance the point OC to the point C is R. The assumption that when we spot the object it is the edge of it means that the angle to reach the center is 90 degrees from the spotting angle, or if we in triangle 2 call the angle at O for β then $\phi + \beta = 90 <=> \beta = 90 - \phi$ With two distances and one angle know we can use the law of cosines [4] to calculate the remaining side. We get the following equation, where we call length OC,C for R and O,C for d_{obj}

$$R^2 = d_{obj}^2 + R_{obj}^2 - 2 * d_{obj} * R_{obj} * cos(\beta) \tag{15}$$

$$R_{obj} = \frac{d_{obj}^2 - R^2}{2} \pm \sqrt{(\frac{d_{obj}^2 - R^2}{2})^2 + 2 * d_{obj} * R_{obj} * cos(\beta)} \tag{16}$$

The equation 16 can give 2 solutions, but in our case we are only interested in the first, i.e. when the ± is negative. This gives finally:

$$R_{obj} = \frac{d_{obj}^2 - R^2}{2} - \sqrt{(\frac{d_{obj}^2 - R^2}{2})^2 + 2 * d_{obj} * R_{obj} * cos(\beta)} \tag{17}$$

Once R_{obj} is known, the center point of the obstacle, OC, can be found by treating the line between OC and O as a vector. Once the point is found the obstacle can finally be described by the following equation:

$$(x - OC_x)^2 + (y - OC_y)^2 = R_{obj}^2 \tag{18}$$

2.6 Collision avoidance

After detecting the obstacle we now have the equations 14 and 18 which are the equations of the trajectory and the obstacle.

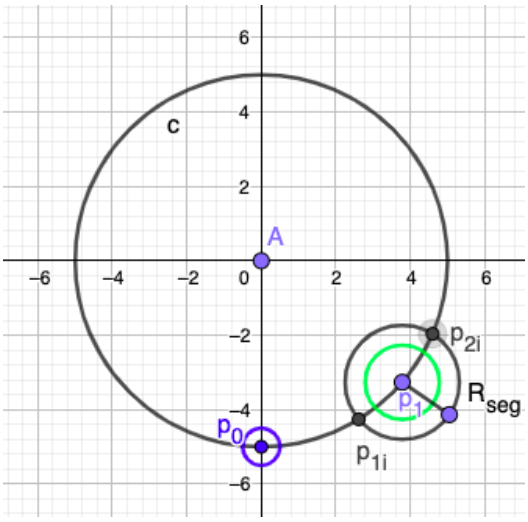

Figure 4. Collision avoidance

Figure 5. Crazyflie

With these two equations we can calculate the points where they intersect and we can avoid the obstacle by flying the drone between these two points using another trajectory. However, we need to have a bigger radius for the collision avoidance trajectory because there is a risk of collision between the drone and the object when it reaches the point where both circumferences intersect. We defined the variable R_{seg} which makes us able to define another circumference with the center on the obstacle center:

$$(x - p_{1x})^2 + (y - p_{1y})^2 = R_{seg}^2 \tag{19}$$

When the drone reaches the point p_{1i} it will start the new trajectory given by the equation 19. When it reaches the point p_{2i} it will return to the previous trajectory and with this we can avoid the collision as we can see in figure 4.

3. Electro-Mechanics

The drone used in this project is a Crazyflie 2.1, seen in figure 5, along with different decks mounted on it. The Crazyflie itself has an IMU with a three axis accelerometer and gyroscope. However, additional decks are needed for the collision detection and avoidance.

3.1 Flow deck

The flow deck is mounted underneath the Crazyflie and has optical sensors to both measure the distance to the ground as well as detecting movement relative to the ground. This deck is used to create an accurate model of the drone's movements. In practise it is used for the drone to keep a fixed distance to the ground, this makes for a easy way to keep the drone hovering in place, without too much configuration. Since the deck also has an optical flow sensor that can detect horizontal movement, the drone can, with just the flow deck, estimate how far the drone has moved in any direction, just based on data from the sensor.

3.2 Multi-ranger deck

The Multi-ranger deck, is mounted on top of the Crazyflie and measures distances to obstacles in five directions, front, back, right, left and up. Like the Flow deck the Multi-ranger deck also uses optical sensors to accurately measure distances to objects of up to 4 meters. The deck works by sending out laser straight out in the five directions and measure the distance to the point where the laser collides with an obstacle. Since the deck works by using lasers, it can only detect a single point in each direction, meaning that it has no field of view, and two different points of collision can only be detected at two different points in time. With this restriction we can not detect the entire outline of the obstacle, but have to estimate the size of the obstacle by only using a single point, and the distance from the drone to that point.

3.3 Lighthouse Positioning deck

The lighthouse deck uses base stations to give absolute and accurate positions of the drone while flying. Without the Lighthouse deck the position of the Crazyflie is estimated using data from the flow deck and a Kalman filter. However, the Lighthouse positioning deck will not be used in this project since the Multi-ranger deck is mounted in the same position, and the two decks can not be used together. Since the Multi-ranger deck is used to get the distance and direction to the obstacles, which is crucial to actually follow a set path and avoid obstacles. The position of the drone can be estimated in other ways, for this project, the Multi-ranger deck is chosen over the Lighthouse positioning deck.

4. Control

4.1 PI

Since the drone is supposed to fly in a circular trajectory, we found a function within the crazyflie python library, cflib, called *circle_left*. This gives the drone a constant velocity and constant yaw rotation. However, this function does not compensate for any kind of errors in the drone's trajectory. In order for the drone to be able to correct itself back to a given trajectory, two simple PI controllers were implemented. Since our trajectory is circular, there are only two variables that need to be regulated. The first is the distance to the origin point, denoted d_o, and the second is the drone's current yaw angle,

denoted y_c. The distance to the origin can be calculated using our current X and Y coordinates according to equation 20

$$d_o = \sqrt{x^2 + y^2} \tag{20}$$

Given this, and the radius of circular trajectory r_t we simply calculate the radius error e_r using equation 21

$$e_r = r_t - d_o \tag{21}$$

The second variable that needs to be regulated is the yaw of the drone, which is its rotation along the horizontal plane. The current desired yaw y_d can also be calculated using the X and Y coordinates using equation 22

$$y_d = arctan(y/x) + 90 \tag{22}$$

The yaw error e_y is the calculated using the current yaw value given by the drone y_c using equation 23

$$e_y = y_d - y_c \tag{23}$$

If the drone is now given a constant velocity in the x-axis (in the forward direction), we can use the radius error e_r and a PI controller to regulate the distance to the origin to be constant, using the drones lateral thrust, T_y, along the y-axis. If we then regulate the drone's rotation along the horizontal plane, R_y (the drone's yaw), using another PI controller and the yaw error e_y, we can have it match the current desired yaw. These two controllers will allow the drone to fly along a circular trajectory and keep the front of the drone facing along the tangent line of the circle. This gives us the following two controllers:

$$T_y(t) = K_p e_r(t) + K_I \int_0^t e_r(t) \tag{24}$$

$$R_y(t) = K_p e_y(t) + K_I \int_0^t e_y(t) \tag{25}$$

Integrating the error can be very complicated, so we substitute the integral with a Riemann sum of the error, R_e, which we calculate during the flight, giving us the final controllers:

$$T_y(t) = K_p e_r(t) + K_I R_{er}(t) \tag{26}$$

$$R_y(t) = K_p e_y(t) + K_I R_{ey}(t) \tag{27}$$

4.2 Program Loop

The collision avoidance algorithm was implemented in the following fashion. When the drone takes of from the floor, we hover in place for a little bit and normalize the drone's coordinates. We chose the starting position of $(0, -1)$, with the yaw value of 0 degrees, meaning that in our graphs the drone will start at the very bottom of the circle facing right.

In the main loop we define a Boolean value keep_flying, which is set to true as long as we wish the drone to continue flying. This value can be set to false either by triggering the drone's top sensor, or by user input from the keyboard, in order to terminate the program and land the drone, allowing us to easily abort flight using the drone itself, instead of relying on the computer. In the main loop of the program the drone will continuously fly along it's circular trajectory around the origin, logging the data from it's sensors and IMU so we can keep track of it's position using the Kalman filter. In each iteration of this loop we first log the above mentioned data, next we check if the front sensor has detected a value lower than our tolerance limit, and thus detected an obstacle on the path, and lastly if no obstacle was detected we keep flying around the origin. The reason we use a tolerance limit is to ensure that what we are detecting is an actual obstacle on the drone's trajectory and not simply the walls of the room.

If we have detected and obstacle we enter a method for avoiding the obstacle. In this method the drone stops and uses equations 15, 16, and 17 from section 2.5 to calculate the radius of the obstacle. It then finds the obstacle's center point and describes the obstacle by using 18. Now we define a new larger circular trajectory centered around the obstacle center point, and calculate the intersections between this circle and the original circular path using 14 and 18. Then we simply tune the PI controllers to regulate the distance and yaw angle to the center point of the obstacle in stead of the center of the circle, let the drone rotate a little towards it's new trajectory, and finally avoid the obstacle using the new circle. When the drone gets close to the second intersection point, it stops, rotates towards the original trajectory, sets the center point of the original circle as it's reference and finally returns to the main loop, moving along the original trajectory.

A code skeleton detailing the pseudo-code for the program can be found i figure 6.

```
void main(){
    tolerance_limit = 0.6;
    bool keep_flying = true;
    while(keep_flying){
        log_data();

        if(sensor.up() < 0.2)
            keep_flying = false;

        else if(sensor.front <= tolerance_limit)
            avoid_obstacle();

        else
            update_velocities(PI1, PI2);
    }
}

void avoid_obstacle(){
    stop();
    obstacle_radius = calc_obs_rad();
    p1i = calc_intersection_1(obstacle_radius);
    p2i = calc_intersection_2(obstacle_radius);
    rotate_right();
    while(distance_to(p2i) > 0.1)
        fly_around();
    stop();
    rotate_right();
}
```

Figure 6. Pseudo-code describing the collision avoidance algorithm

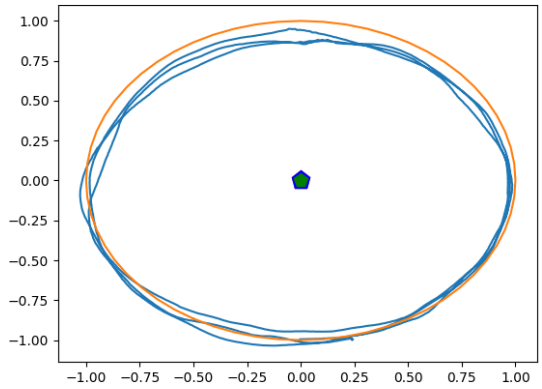

Figure 7. 3 laps without controller

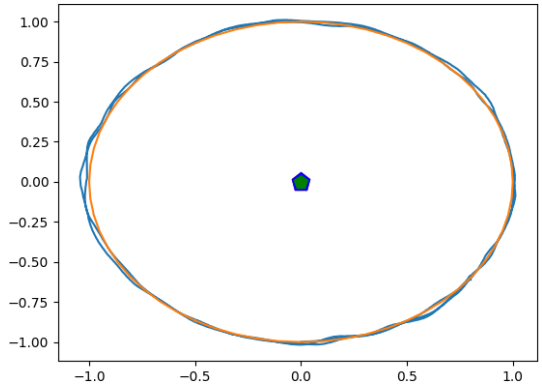

Figure 8. 3 laps with PI-controller

5. Results

5.1 Controller

Figure 7 and figure 8 shows the path the drone takes when flying in a circle for 3 laps, where the red circle is the path it is supposed to follow, and the blue line is the actual path the drone took. Figure 7 shows the result when using the *circle_left* function in the cflib library, and letting it runs undisturbed for 3 laps. In figure 8 the path is discretised, with correction to both velocity and yaw at each point on the circle. Comparing the two figures, it is clear that using a PI-controller with the desired circle as reference will keep the drone more precise on its path, in comparison to using the built-in circle function.

5.2 Obstacle Estimation

Radius Estimation An obstacle with a radius of 0.14 meters was placed on the circle the drone was supposed to fly around. The results in table 1 show the algorithm's estimation of the radius, based on the sensor data, of the obstacle for five different runs, as well as the error and proportion of error compared to the actual obstacle.

Position Estimation The position of the obstacle is also important for the drone to calculate a path around the obstacle without colliding with it. The table 2 shows the resulting estimations of the position of an obstacle placed with its center at the point (0, 1).

Estimated radius	Error	Error %
0.12018 meters	0.01982	14.1%
0.14958 meters	0.00985	6.8%
0.14139 meters	0.00139	1.0%
0.13527 meters	0.00473	3.4%
0.12769 meters	0.01231	8.8%

Table 1. Estimated radius of a 0.14 meter wide obstacle

Estimated position	Error X	Error Y
(0.17061, 0.975150)	0.17061	0.02485
(0.04476, 0.99602)	0.04476	0.00398
(0.12750, 0.98642)	0.12750	0.01358
(0.07869, 0.99563)	0.07869	0.00437
(-0.01392, 0.99570)	0.01392	0.0043

Table 2. Estimated positions of an obstacle placed at (0, 1)

5.3 Obstacle Avoidance

Since the drone only estimates the radius, and has no way on knowing how big the error is, a safety radius is also calculated to make sure the drone does not collide with the obstacle. In figure 9 two obstacles of different sizes were placed on the path of the drone. The blue line shows the path that the drone took, the red circles show the estimated circumference of the two obstacles, the green circles show the desired path of the drone around the obstacle, with the added safety radius, and the orange line shows the desired path the drone should follow in between avoiding obstacles. To test the accuracy of the method we also ran another test with 3 obstacles placed on the path, as we can see in figure 10.

6. Discussion

6.1 Project Results

After the analysis of the method and implementation, and concerning the results we obtained, we can ascertain that we

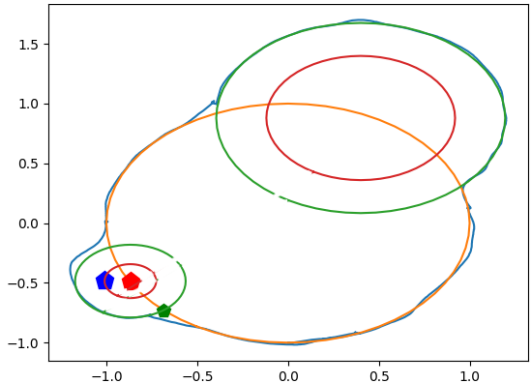

Figure 9. Flight path of drone avoiding two obstacles

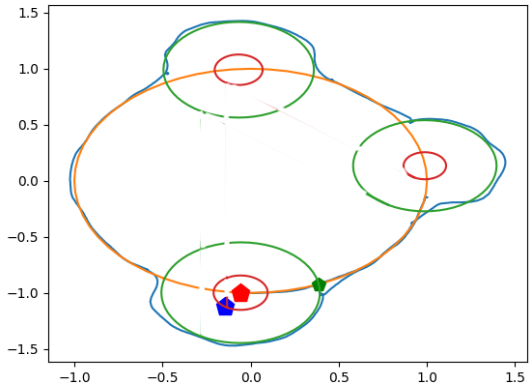

Figure 10. Flight path of drone avoiding three obstacles

met the goal of this project successfully. With this approach we can now detect and avoid obstacles with relatively good precision.

As seen in the graphs and tables from section 5. the drone can fly in a circular path, correct itself mid flight and avoid obstacles. We tested the method using a 2 obstacle and a 3 obstacle course. In both of them the drone could detect and avoid the obstacles and return to the previous path. It is clear from comparing figure 7 and figure 8, that using the controller implemented in this project keeps the drone more closely on its path compared to letting the drone fly without a controller.

However, the radius estimation and placement of any obstacle are based on a single measurement from the sensor on the Multi-ranger deck, and the assumption that all obstacles are placed with its center on the path. This makes it so that the accuracy of the estimations will vary depending on the obstacles shape and positions relative to the drone's trajectory. The fact that the drone does not follow exactly the circular path, but instead constantly has to correct it's velocity and angle, is a problem as well because we work on the assumption that the first time that we detect an obstacle it is on its edge and if the drone is not placed correctly on the trajectory, it may instead detect some other part of the obstacle, which in turn will produce errors in the radius estimation. However, we can see from table 1 and from the table 2 that we can estimate the radius and place it in the trajectory with relatively good precision. This approach was chosen since even though the estimation of the radius will be prone to errors, it should, for most shapes of obstacles, make it so that the drone calculates a path that is big enough to avoid the obstacle. Although the circle will sometimes be unnecessarily large, the drone would still clear the obstacle by a good margin.

6.2 Project Setbacks

First, we had some hardware problems due to the fact that the flow deck broke in one of the tests. We did not immediately identify the cause this problem, but only the effect, namely that the drone started to drift during all test flights. We performed some hardware tests and then we could identify that the flow deck was broken. Then we had to wait for it to be repaired and after that we proceeded with the measurements and the tests. This issue took us 1 week to solve and since we only had 6 weeks to complete this project, we unfortunately lost a substantial amount of time. However, we had taken into account that this type of setback would likely occur during the project, and had structured our project plan to include a buffer week, to be used for fixing bugs and errors. Because of this, we were still able to complete the project on time.

Another issue that we encountered is the flow deck sensor does not provide an absolute coordinate measurement. This made our task even more difficult due to the fact that our method is based on a set of coordinate systems. However, the Kalman filter measurements that were explained in section 2.3 were accurate enough that our approach still worked. The accuracy of the project could consequently be improved using an absolute position sensor such as the lighthouse position deck explained in section 3.3.

6.3 Final Remarks

While working on this project we realized the various amount of applications that quadcopters have and that it is a huge area of research. The next steps to this project would be trajectory optimization for following the circular path and then proceed to implement a collision avoidance algorithm for every possible trajectory and design an optimal controller to make the algorithm robust. Unfortunately, due to the limited time that we had we could not go further in the experiments.

References

[1] *Amazon prime air*. URL: https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011 (visited on 2021-12-06).

[2] M. A. Botto. "Optimal control - linear quadratic gaussian" (2020).

[3] C. Chadehumbe and J. Sjöberg. "Autonomous flight of the micro drone crazyflie 2.1 through an obstacle course" (2020).

[4] *Law of cosines*. URL: https://en.wikipedia.org/wiki/Law_of_cosines (visited on 2022-01-05).

[5] A. Moutinho. "Drones control and simulation" (2018).

[6] *Volcano disaster prevention*. URL: https://droneharmony.com/volcano-disaster-prevention/ (visited on 2021-12-06).

Path tracking with linear quadratic control using a GNSS receiver

Lars Brander[1] **Alexis Klotz**[2] **Vilius Koegst**[3] **Arian Maloku**[4]

Project Advisor: **Zheng Jia**

[1]la7853br-s@student.lu.se [2]alexis.klotz@insa-lyon.fr [3]vi6048ke-s@student.lu.se
[4]ar4172ma-s@student.lu.se

Abstract: The goal of the project was to implement path tracking on a small model car using model predictive control (MPC). Due to unexpected problems with the software to hardware compatibility this goal was not achieved. Instead, a less complex controller, one that does not require complex computation on board was implemented, namely linear-quadratic regulator (LQR). As the vehicle only has reliable data on position, a state estimator was used to extract directional data needed for path tracking. The path to follow was created by interpolation between input points using class Splinepath, which is also responsible for calculations of the error signals required by the controller and curvature of the path which was later used for feedforwarding. The non-linear system was linearised using Jacobian matrices around the linearisation point of straight path and stationary velocity of 1,5 m/s. The vehicle was then run many times to experimentally determine the tuning values for Q and R as well as for the feedforwarding term. The ultimate result is a vehicle that can follow a predetermined path well, without oscillations or significant over-damping or stationary error. The path tracking still works with velocities much higher than the 1,5 m/s in linearisation point, although the reduction in performance is clearly notable.

1. Introduction

1.1 Background

Recently there has been a significant push towards autonomous vehicles in the automotive industry and it is probably one of the most impactful innovations to look forward in the near future. The elimination of human error from traffic as well as reduced reaction time is expected to significantly increase safety. In fact, it is estimated than more than 95% of the road accident are caused by human error [1].

Even with the safety benefits aside, autonomous vehicles will allow an increase in efficiency as well. In fact, the majority of the distribution networks are based primarily on trucks. Currently, there are limits on how many hours a driver can work (for obvious safety reasons), which ultimately slows down the network. Therefore, the automotive industry is particularly well suited for automation as it would solve most of the current problems.

In this project the authors will familiarize themselves with one of the most important aspects of this upcoming automatisation, namely path tracking.

1.2 Project goal

The original goal was to model, develop and control a small electric car, called Slimdog, to follow a predefined path. The car can be seen in Figure 1. The initial controller wanted was a Model Predictive Controller (MPC). However, like it will be explained further on, the authors have used a Linear Quadratic Regulator (LQR). The Slimdog should avoid overdamping, oscillations, or even, significant constant error.

The car together with all the necessary hardware were all provided as well as a lot of software groundwork like actuation and hardware setup. The work left to be done was solely path tracking related.

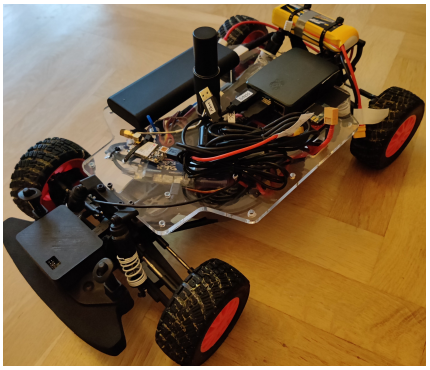

Figure 1. The Slimdog

2. Modeling

The model only has two states, the error in position (the distance between the car and the closest point on the path) d, and the error in heading (difference between the heading of the car and the tangential angle of the closest point on the path) θ_e. In this model the vehicle speed is assumed to be constant, only the steering is controlled see Figure 2. The controller aims to keep both states as close to 0 as possible, making the car follow the path.

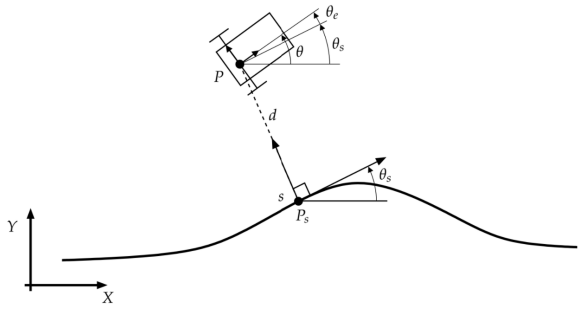

Figure 2. An overview of the model [2]

As one can guess, the resulting system 1 is non-linear and depends on external factors, like the curvature of the path at the current position. The v is the velocity of the car, the $c(s)$ is the curvature of the path at point s and u is the control signal.

$$\begin{pmatrix} \dot{d} \\ \dot{\theta_e} \end{pmatrix} = v \begin{pmatrix} sin(\theta_e) \\ u - \frac{c(s)}{1 - dc(s)} \end{pmatrix} \tag{1}$$

The system was then linearised around the point $\theta_e = 0$ and $c(s) = 0$ using Jacobian matrixes. This linearisation point assumes that the cars heading is at least close to the tangential direction of the path, which is reasonable. It also assumes a completely straight path which is perhaps less reasonable but, as shown in the result section, does not seem to impair the vehicles performance significantly. The linearised LTI system can be seen in equation 2. The linearisation was done by hand, while the discretization was carried out in matlab with Ts = 0,1.

$$\begin{pmatrix} \dot{d} \\ \dot{\theta_e} \end{pmatrix} = v \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} d \\ \theta_e \end{pmatrix} + v \begin{pmatrix} 0 \\ 1 \end{pmatrix} u \tag{2}$$

Table 1. Variable look-up table

d = Distance to the nearest point on the path
θ_e = Heading error
v = Velocity (assumed constant)
$c(s)$ = Curvature at point s on the path
u = Control signal (only steering)

3. Electro-Mechanics

3.1 Hardware Setup

The car is already built seen in figure 1 by four wheels with front-wheel steering, a Power bank that supplies the Raspberry Pi and a lithium battery that supplies the VESC module.

The hardware consists of four modules:

- A GNSS receiver
- A compass – an Arduino microcontroller (not used)
- An Xbee RF communication device (not used)
- A VESC motor speed controller

The hardware setup can be seen in Figure 3.
The RF communication device isn't used as it is simply not useful for our application, while the compass isn't used as it provides very unreliable data. Heading direction is instead extracted from the provided state estimator, using a Kalman filter.

The car uses the GNSS technology to know its position on the globe. In fact, the GNSS outputs are the latitude, longitude, and height. Since the authors consider being on a flat surface, the height won't be considered in the control of the car.

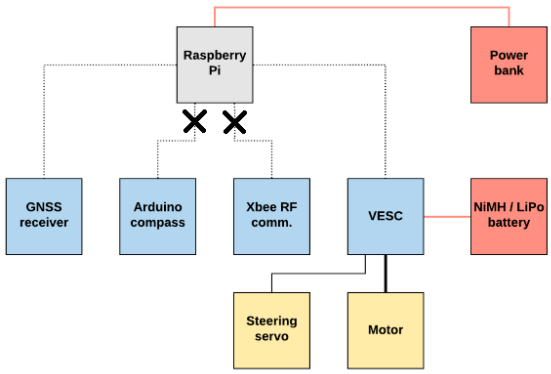

Figure 3. Overview of hardware modules and their connections

3.2 Software Setup

In Figure 4 we see the structure of how the different software's work together. The telemetry part is not used in this project as it handles the radio controlling hardware in the car. The GNSS part is where data receives continuously position and update it in the state_estimator were it store state information. In the controller part it performs control algorithm based on state_estimator and send the output to the VESC module. The VESC module then Communicates the desired velocity and steering position to the motor speed controller. The logger is a fairly simple class that logg:s all interesting data from the car to a separate file that later can be presented by a visualising software for analysis.

4. Control

As previously mentioned, the original goal was to control the vehicle using MPC. The plan was to use a library called "CasADi" to quickly and efficiently solve the cost function which is necessary for the MPC. Unfortunately the library proved to be incompatible with the Raspberry Pi and, after many attempts, we were ultimately unable to install it.

In order to still have working prototype a decision was made to implement path tracking using a simpler controller. An LQR controller was chosen, as it uses similar concepts as the MPC but instead, the cost function is non-changing and therefore only needs to be calculated once. This can then be

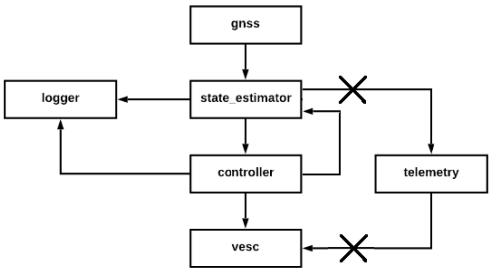

Figure 4. The overlaying software structure of the Slimdog.

done externally outside the vehicle, removing any need of a specialized library on board.

4.1 State Estimator

The estimator's function is to provide us with a more reliable state data such as our heading and position. It does that by taking the raw data from the GNSS positioning hardware and combining it with the output of the car to filter out noise in our position signal.

Before the car starts to move, it initializes the coordinate system. In practise, it means that it stands still for a number of samples and then calculates the average.

Once the calibration is done, a local coordinate system is created with the position calculated as its origin. Without this, the path would have to be coded in global coordinates which would introduce new problems and need to be recalculated for every new test location.

It has to be noted that the estimator takes a short while converge, meaning that the estimation of the heading is inaccurate in the beginning. This will be clearly visible in the plots in the result section of this report, as the error is noticeably larger near the origin where the car starts.

The state estimator was already implemented and provided to us, as the compass was shown to provide unreliable data. All the control methods described use both, the positional and the directional data, from the estimator.

4.2 Splinepath

The Splinepath class has a function that returns a 2-dimensional coherent path. This 2- dimensional path will be created from a Nx2 matrix containing x and y coordinates, these coordinates can be collected by the cars GNSS receiver and extracted from the log file, alternatively manually selected to create the Nx2 matrix.

This class provides important input information to the controller, such as the distance between the car and the path as well as the normal and tangential angle for any point of the path. Also, this class provide the curvature for any point of the path, allowing for better control using feed-forwarding. These functions allow us to calculate the d and the θ_e error signals necessary for the controller.

4.3 LQR

Linear-quadratic regulator (LQR) is a controller where the feedback constants are decided by minimizing a cost function.

$$J = \sum_{t=0}^{\infty} x_t^T Q x_t + u_t^T R u_t \tag{3}$$

Where x is the state vector, u is the control signal vector and Q and R are diagonal tuning matrices. The idea is that the controller is trying to minimize both, the error (the states) and the control signal (the steering). The Q and R matrices are there to weigh the different states and control signals, the higher the value the more aggressive the controller will be in trying to minimize that state/control signal in relation to others. In this case, there are only 2 states so the Q has a size of 2x2 see 4. and, since we only control steering, R is a scalar.

$$Q = \begin{pmatrix} q_1 & 0 \\ 0 & q_2 \end{pmatrix} \tag{4}$$

The system was then discretised and the feedback gains were calculated with the Matlab function lqrd(A,B,Q,R,Ts) where A and B matrices are from the linearised system in equation 2. Ts was chosen to be 0.1 seconds.

Finally to determine adequate Q and R, the car was run multiple times and the values were adjusted until the performance of the car was satisfactory.

$$\sum_{t=0}^{\infty} q_1 d_t^2 + q_2 \theta_{e,t}^2 + R u_t^2 \tag{5}$$

4.4 Control law

The control law is calculated by the following equation:

$$u = K_1 d + K_2 \theta_e + c_1 curv(s). \tag{6}$$

Where K_1 and K_2 are the feedback gains, c_1 is a tuning constant and curv(s) the curvature of the projected point s on the path. The curvature term is added to control law as feed-forwarding for better reference tracking. The constant c_1 was determined experimentally. The u is then sent to the actuator class which translates the angle into a voltage for the steering servos as well as saturates it for safety.

5. Results

5.1 LQR optimization

From extended testing of different values of Q and R yielding different K values, an optimum was found at the values in Table 2.

5.2 Test results

The result is based on a number of test runs on different predefined paths, the position data is continuously stored during the run to be presented in the following graphs as an orange line, comparing it to the desired predefined path presented as a blue line. The starting point for all tests is in the origin, (0,0).

Table 2. Optimal values found from testing. This table shows all the relevant values for our controller, including the curvature coefficient c_1.

Q =	[1 0 0 4]
R =	5
v =	1,5 m/s
c_1 =	0,1
K_1 =	0,4062
K_2 =	1,2134

The result is a controller that has one output, steering angle. It can only drive at a constant predefined speed optimized at 1.5 m/s, the handling of this controller will now be shown. First, the authors tested two different LQR controller setup, one that accounts for the curvature of the path closest to the Slimdog and one that does not account for the curvature. These can be seen in Figure 5 and 6.

Figure 5 shows the test run with the LQR controller and curvature accounting, c_1 is set to 0.1. The blue line is the reference path and the orange line is the cars actual path. The scale in both x- and y- direction is in meters. The speed is set to 1.5 m/s.

Figure 6 shows the test run with the LQR controller without curvature accounting, c_1 is set to 0.1 The blue line is the reference path and the orange line is the cars actual path. The scale in both x- and y- direction is in meters. The speed is set to 1.5 m/s.

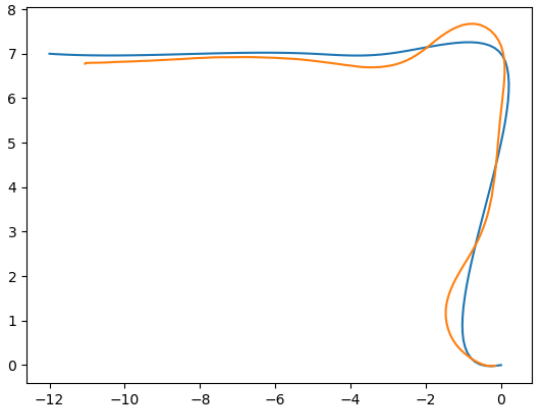

Figure 5. Plotting of the reference path (blue line) and the followed path (orange line). The LQR controller and the curvature have been used. c_1 is set to 0.1 and the speed to 1.5 m/s.

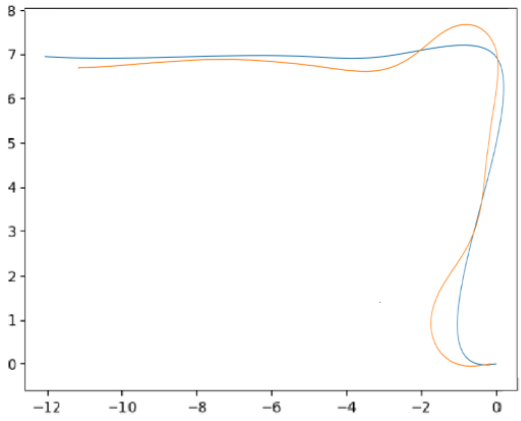

Figure 6. Plotting of the reference path (blue line) and the followed path (orange line). The LQR controller without the curvature have been used. c_1 is set to 0.1 and the speed to 1.5 m/s.

By comparing the figures, a small improvements can be seen when including the curvature into the controller. However over adjusting the output based on the curvature leads to a worse performing controller as seen in Figure 7.

The Figure 7 shows the test run with the LQR controller that is over compensating for curvature, c_1 is set to 0,3. The blue line is the reference path and the orange line is the cars actual path. The scale in both x- and y- direction is in meters. The speed is set to 1.5 m/s.

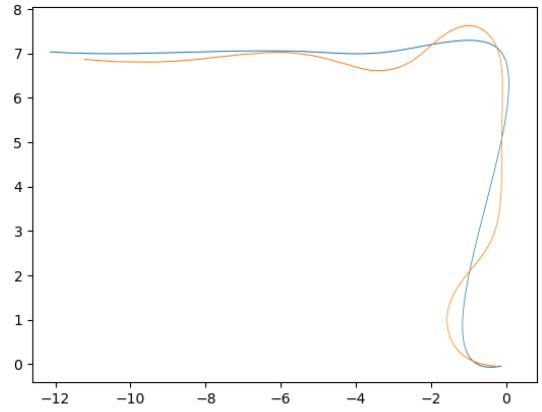

Figure 7. Plotting of the reference path (blue line) and the followed path (orange line). The LQR controller is over compensating for curvature. c_1 is set to 0.3 and the speed to 1.5 m/s.

A more complex path shows the Slimdog:s handling at different speeds, although the LQR controller is optimized for 1.5 m/s at all the different speeds. This can be seen in Figure 8, 9 and 10.

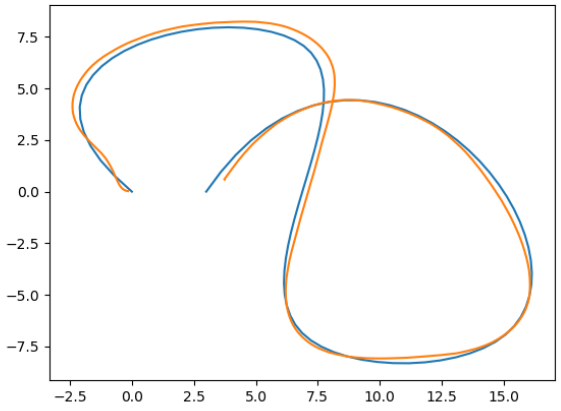

Figure 8. Slimdog's handling at 1.5 m/s.

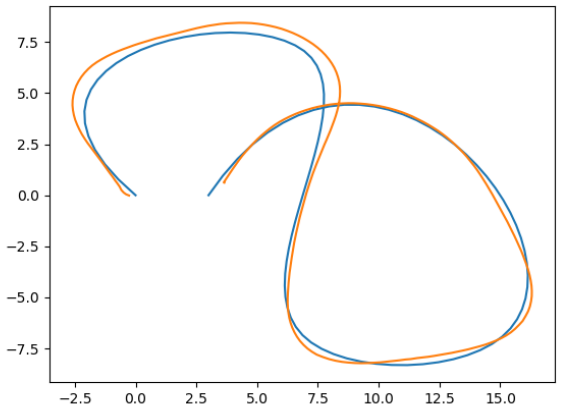

Figure 9. Slimdog's handling at 2.5 m/s.

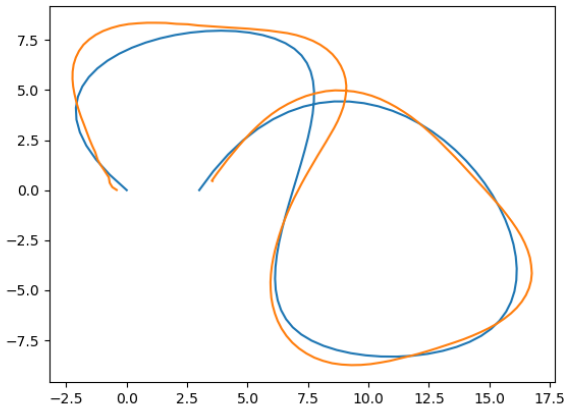

Figure 10. Slimdog's handling at 4 m/s.

From these Figures, the conclusion that can be drawn is that the Slimdog works best at the lowest speed in which the controller where optimized, but is still functioning at greater speeds although struggling to keep close to the path in the narrow corners.

One undesirable feature of this controller which we encountered is that it only works in close proximity to the path, if the distance is to great between the car and the path the controller will make the car go in a circle, a solution to this is therefore necessary to increase the robustness of this controller.

To conclude the results, the Slimdog manage to meet the updated project goals of a LQR controller. The goals were a controller that could follow a predefined path while avoiding overdamping, oscillations, and significant constant error.

6. Discussion

6.1 Preexisting code

The authors have really realised the importance of having a well documented program code. The majority of the time spent on this project was probably trying to understand what happens in the provided code and how to manipulate it, as it wasn't very well commented. This led to a lot of software troubleshooting that was quite time demanding, because every time we would change something and wanted to test it, we would have to take the car outside. This however also is a skill that needs to be practiced and most likely will prove useful future projects. Unfortunately, due to time constraints, the code we wrote ourselves isn't well structured but some comments has been added.

6.2 Improvements to be made

There are some improvements that can be made to the current controller, firstly the issue with the Slimdog driving in circles when it is too far from the path. There are ways to solve this, one such solution would be to switch controller when the distance goes beyond a certain distance to direct pursuit controller that simply would drive straight to the closes point on the track. Another alternative could be to implement a non-linear controller that would guarantee global stability, although that would require switching the controlling method close to the path as well. As the prototype was in a functioning state only towards the very end of the schedule, we are confident that the performance of the vehicle could still be improved by better tuning.

Another improvement to the current controller could be to actively control the speed, from Figure 8 to 10 we can see that taking a corner slowly effectively decreases the distance error to the path. Therefore, a speed that is dependent on the current curvature could be appropriate. The reason for this outcome could be a combination of factors, not only the fact that the controller only is optimized at 1,5 m/s, but also the handling of the Slimdog. Examples of factors that play into the handling is the friction between the tires and the surface, ultimately setting the limit for the maximum speed while maintaining a given curvature. Since the Slimdog has a suspension setup, weight transfer of the vehicle entering a corner is a highly relevant

factor of the handling as well. Future investigation and testing are necessary to optimise the speed control.

Also, currently the car stops by its own when it comes as close to 0.5 meter from the finishing point. The problem by doing this is that if the finish point is the starting one, the car won't start since it thinks it has arrived. This problem could be address by stopping the car once Splinepaths orthogonal projection places the car at the last point.

We had made a mistake by using a library that we were not completely sure if it would work on the Raspberry Pi. We believe though that we have a decent understanding of the code and how the Slimdog operates now, so that the implementation of MPC would not be too difficult but maybe given more time for it. The work left to be done would be to write some code that could predict future state of the system (possibly with a Runge-Kutta method) and find a library that could effectively solve multiple cost functions on board. These calculations will most likely be the bottleneck in our sampling frequency which, if made too low, could compromise the system stability.

6.3 Testing difficulties

Testing the car was very time consuming as it has to be done outside. Indoor locations were either too small for a reasonable path, or blocked the the GNSS signal. This was made even worse by a lot of bad weather as the project was done under November/December. Most of the hardware lays bare on top of the Slimdog, so the testing was often cancelled or limited to quick 2-3 minute sessions because of rain or snow. This occurred very often, so we would strongly recommend on either finding a suitable indoor testing location or weatherproofing the vehicle (perhaps with a 3D printed case) if the project is to be carried out during this time of the year. For the tuning of the controller parameters, a simulation could be used instead removing some outside testing requirements. This however would heavily depend on the quality of the model and would only reduce the tuning time.

6.4 Conclusion

When looking back to the starting point, we have made great progress. The importance of having a clear project plan was proven. The Gantt chart was updated during the whole extent of the project and was generally helpful for keeping the deadlines. It was difficult to find a time where all the group members could meet to work on the project, but with the distribution of the workload and division into smaller groups, it worked out in the end. It was a lot of work, but with a lot of time dedicated from everyone in the team, a working prototype was completed in time. An overview of the final Gantt chart can be seen in Figure 11.

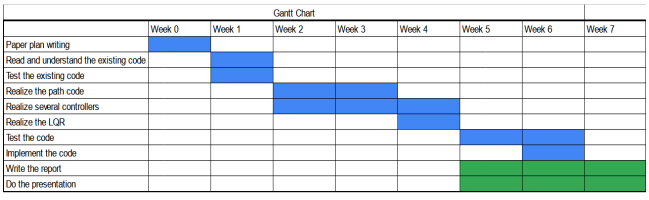

Figure 11. The final GANTT chart for the project

References

[1] N. H. T. S. Administration. *Traffic safety facts*.

[2] E. Frisk. *Ground vehicle motion control*.

Riemannian Geometry for Transfer Learning in a practical BCI experiment

Emil Bergström[1] **Mitra Malekpour**[2] **David Sandell**[3] **Andre Rath**[4]
Juan de la Peña[5]

Project Advisor: **Martin Gemborn Nilsson**

[1]em7751be-s@student.lu.se [2]mi5345ma-s@student.lu.se [3]elt15dsa@student.lu.se
[4]an5700ra-s@student.lu.se [5]ju1111de-s@student.lu.se

Abstract: Brain-Computer-Interface (BCI) devices are an area of extensive research and potential for future advancement. In order to make BCI more accessible for large scale applications, the time and effort needed to get started using BCI needs to be lowered. One approach to solving this issue is with so-called transfer learning (TL). In the case of BCI, data and previously trained models from earlier sessions can be used, possibly also from different subjects and experiment setups. The goal of this project is to implement a near-real-time process pipeline from an Electroencephalography (EEG) device (the Muse-S headband) to a machine learning (ML) algorithm giving results from the data. For this purpose, TL methods in ML based off a method known as Riemannian Procrustes Analysis (RPA) [27] are to be used to minimize calibration time. Simulating the calibrations of transfer learning on pre-recorded data shows promising results. An integrated Muse to experiment to ML pipeline gives data that can be used for classification, with certain results giving around 60-70% accuracy. The best applied Riemannian methods finally gave an accuracy of 75-80 % on any single dataset, proving a viable pathway to transfer learning.

1. Introduction

Brain-Computer-Interface (BCI) devices are, as the name suggests, devices that allow humans to directly control or in some other way interact with technology using only neurological activity. One way to achieve this is with noninvasive electroencephalography (EEG), the process in which surface electrodes placed on the scalp are used to record electrical activities generated by the neurological system [28].

BCI devices have proven to be quite promising in many areas [20, 17]. Cochlear implants are perhaps the most well established BCI device. They are a type of auditory-sensory neuroprosthetic that directly stimulates the auditory nerve to provide a sense-of-sound for people who are either hard-of-hearing or profoundly deaf [10]. Recently, invasive BCI devices have even been used to record motor-cortex activity, enabling the fine motor control of upper-limb prostheses - with individual finger motion [13].

A large issue preventing the widespread commercial usage of EEG BCI devices in industry is poor reliability and robustness [28, 20]. Even in ideal conditions within a single measurement, EEG-based BCI devices suffer from nonstationary signals, a low signal-to-noise ratio and a high sensitivity to variations in environment factors e.g. location [20, 28, 12]. This is further compounded by variations when multiple measurements are attempted or when the BCI device is generalized to different individuals [6, 29, 20, 19]. An EEG-based BCI device must therefore be continuously re-calibrated to each new session, either with the same or a new individual. This requires minutes of calibration time before the BCI device is capable of functioning optimally [20]. Therefore, prior EEG-based BCI devices are mostly constrained to laboratory conditions.

One promising avenue to improve upon the inter-session data variability issue is to utilize so-called Transfer-Learning (TL) methods within the field of Machine-Learning (ML). TL entails using some method that can leverage knowledge from a related dataset to improve the performance or the speed of convergence for a new dataset [22]. In essence, the ML algorithm 'generalizes' knowledge across a wider problem area to enable robustness to context changes. Applied to EEG-based BCI devices, TL principles can thus be used to enable reuse of data recorded in past sessions or from sessions with different individuals and sessions [20].

The overall goal of this project is to examine an implementation of a TL method called Riemannian Procrustes Analysis (RPA) [27] in a real-time laboratory experiment to demonstrate the possibility of minimizing calibration time. In essence, to combine a commercially available EEG headband with a practical experiment and TL, showcasing the possible decrease in calibration time. For this purpose, we will be using a Muse S EEG headband (Figure 1) combined with a real-time experiment formulated to trigger the characteristic P300 neural response pattern, which will then be classified using a TL method functioning in parallel and in near-real-time.

This project is based off a challenge hosted by NeurIPS known as 'Beetl AI': "this competition aims to stimulate the development of transfer learning and meta-learning algorithms applied to a prime example of what makes the use of biosignal data hard, EEG data"[21].

Figure 1. The EEG 'Muse S' headband.

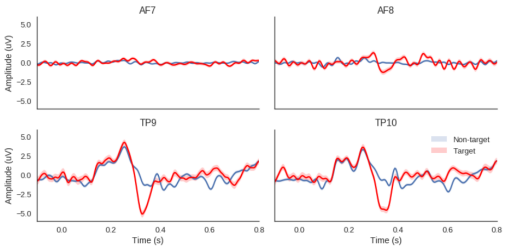

Figure 2. Detection example of a P300 response [3].

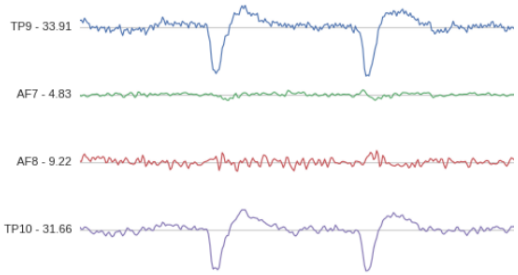

Figure 3. Visualization of streaming data using Muse-LSL [18]

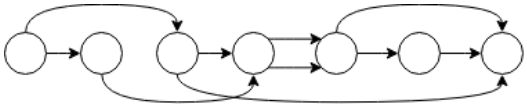

Figure 4. An example directed acyclic graph (DAG). Circles represent nodes, arrows are edges. The connection points between nodes and edges are called ports. Information flows from left to right, at a frequency defined by the graph rate. [7]

2. Neurological background

P300 stands for "Positive 300 millisecond response" and is an event-related potential (ERP) which is a large waveform that can be generated from an EEG using different paradigms and stimuli. Furthermore, P300 utilizes one of the brain's reactions to surprising and unexpected stimuli [26]. A stimulus might be, for example, a short beeping sound. This evokes the P300 response in humans and even animals' brains. The brain always categorizes something new and unexpected as something interesting, so if a random sound occurs, it initiates a certain voltage/time pattern in the brain. This is also known as the oddball paradigm. In summary, we expect that 300 milliseconds after an event the EEG headband would pick up a positive peak in amplitude, see Figure 2.

3. Data acquisition and preparation

A big part of the project consists of obtaining the raw EEG data from the band and processing it into a usable form; in our case, making it compatible with the input data that the ML algorithm expects.

Regardless of the final use of the EEG signal, data is to be collected following specific steps. However, this is dependant on the operating system and development environment to which the headband is connected to, and with which the experiment is meant to be performed. In this project, BlueMuse [15] was installed and used together with Muse-LSL [18]. This allowed the band to connect and set up a lab streaming layer (LSL stream), see Figure 3, which is a system for the unified collection of measurement time series in research experiments that handles both the networking, time-synchronization, (near-) real-time access as well as optionally the centralized collection, viewing and disk recording of the data [14].

As mentioned, raw EEG signals have a poor signal-to-noise-ratio [20, 28, 12] and poor information density. In addition, we know that the P300 response is known to occur specifically with a 300 *ms* latency after the stimulus. [26] It is thus pertinent to focus attention into that appropriate time-interval when searching for the characteristic EEG signal. Moreover, filtering undesirable frequencies in the signal also helps in reducing noise. These and other tasks are possible to manage using Timeflux [7], which is a free and open-source framework for the acquisition and real-time processing of biosignals.

3.1 Timeflux implementation

The Timeflux framework works with *.yaml* files where one or more directed acyclic graphs (DAG) are defined and may interact with each other. This graphs independently execute a certain *process* sequentially at a specific adjustable rate. Every *process* consist of nodes and edges that connect them, where information flows in a given direction, without any loops. Some of these nodes are predefined in the Timeflux environment that can be used directly, whereas others need to be described or slightly modified from an existing implementation, depending on the specific goal at hand. Nodes expect a data input in a specific format: either a numpy structure or datetime-indexed Pandas DataFrame. The second option is preferred as it easily handles timestamps that allow to keep track of when data is generated, processed, and saved.

The experiment uses six graphs, each of them responsible for a different task (see Figure 5).

- Broker: the only purpose of this graph is to allow exchange of information between any other two graphs.
- Reading EEG: makes the acquisition of data from the Muse S band possible, using LSL protocol. There is a series of nodes in the graph, dedicated to collecting events in the same LSL protocol. These events are related to the presentation of stimuli on the screen and it is necessary to keep track of them in this way to match them with the real-time data from the headband. This is done by comparing timestamps from the experiment and the real-time data.
- Visualization: creates an interface to visualize real time data on the screen. Uses a local server with a web GUI

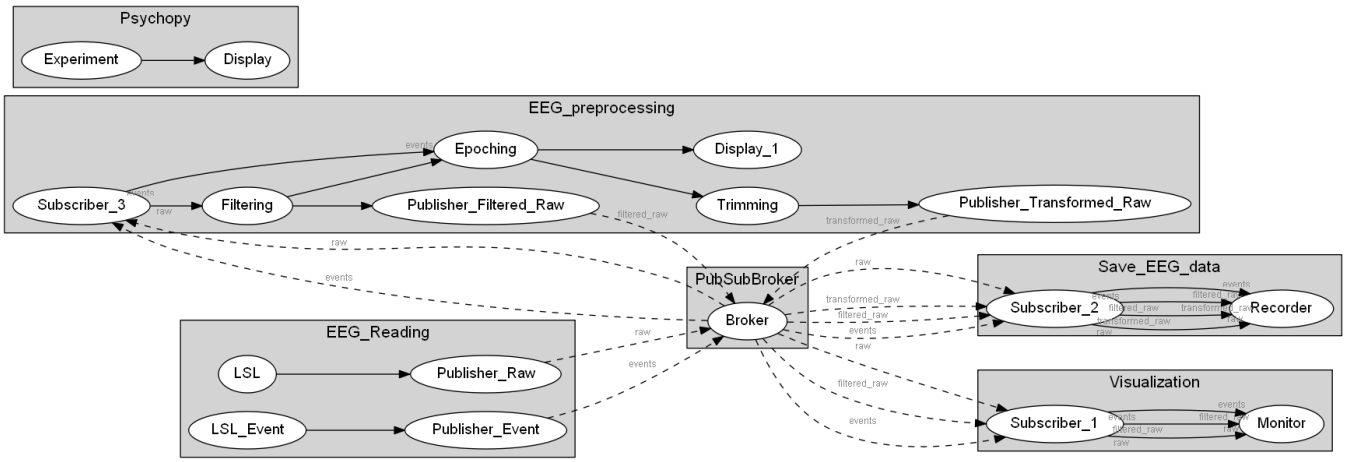

Figure 5. Visual representation of the .yaml file used in the Timeflux environment

incorporated to plot the specified data. It has been programmed to show the raw data (directly from the headband), as well as the filtered data and the presentation of stimuli.

- Save data: this is the module responsible for saving data on request. Upon a keyboard user input, it starts recording the EEG data in to a csv file for later use and analysis.

- EEG Processing: This graph prepares the data for the ML and TL modules. The first node, *Filtering*, uses an IIR band pass filter (see Figure 6) with cutoff frequencies of 1 and 30 Hz to eliminate linear trends such as signal drift or high frequency noise. The *Epoching* node splits the data so that we end up only with a desired time window. In relation to stimuli onset, this node must be in sync with the presentation of the stimuli to the subject doing the experiment. Finally, the *Trimming* node ensures that the epoched data has a constant size everytime, as there might me small deviations otherwise.

- PsychoPy: [25] The PsychoPy graph is responsible for the interaction between patient and experiment. There is more information on this topic in a specific section (see Psychopy: The experiment GUI in section 4.2).

The amount of information to be analyzed is defined in the *Epoching* node described before, where the range property refers to the amount of time before and after the exact time the event triggering the node is detected. Since the P300 response is found roughly 300 ms after the stimuli is presented to the patient, it has been decided to collect 0.8 seconds of data as in reality there might be some deviations from the theory.

4. Experiment setup

4.1 Experiment design

The experimental setup consists of the Muse S headband and the experiment itself. In this experiment, it is desirable to record the brain activities and analyse it when it reacts to

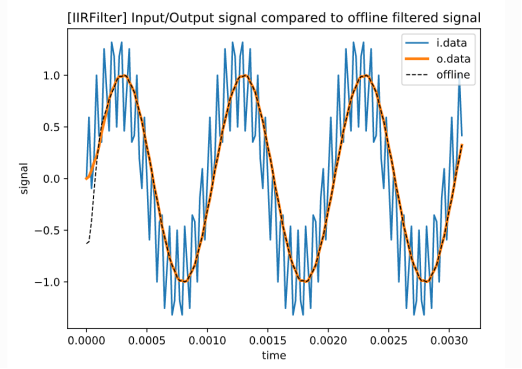

Figure 6. IIR filter example [7]. Visual explanation of how data is filtered. The difference between the online (*o.data*) and offline response is explained by the fact that the input is received in real time, and hence the filter cannot use later data-points to construct the response at the beginning of the filtering process.

something unexpected and surprising. (see Neurological background in section 2).

4.2 Psychopy: The experiment GUI

To implement the EEG experiment, Psychopy is used [25]. Psychopy is an open source software package which allows you to run experiments in the behavioral sciences e.g. in neuroscience, psychology and linguistics. The experiment is based on two different categories, animals and furniture. The reasoning for this choice of categories is to ensure that the brain is presented with stimuli that are both easily distinguishable and easily recognized [9]. The experiment starts with a welcome text followed by the trial, where several pictures from the two different categories are displayed for the user. The pictures are displayed randomly for 5 seconds and the user is asked to press space on the keyboard whenever a dog is displayed on the screen. The experiment goes on for approximately 80 seconds. The structure of the experiment can be seen in figure 6. The experiment starts with a welcome screen, shortly presenting the task to the tester and waiting until the tester gives an input. Thereafter the experiment starts and loops through

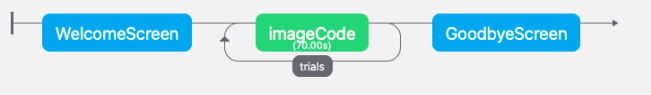

Figure 7. PsychoPy structure

Figure 8. Example of dog and chair images

images of either dogs or furniture for 70 seconds. The experiment ends with a goodbye screen, thanking the tester for the participation.

It is important to know precisely when the P300 response was perceived by the patient, as it is key to perform a good analysis in the Timeflux and ML modules. Therefore, as previously explained, the Psychopy python file needs to include some instructions to send a signal using the LSL protocol with basic metadata (*True* if dog, *False* if chair, and exact time of presentation) to the Timeflux program, which it can read and recognize.

5. Machine Learning

5.1 Theory

Riemannian geometry of SPD matrices

Covariance matrices are Symmetric Positive Definite (SPD), and therefore have a number of constraints on what values the matrix can have. We can therefore define a space of SPD matrices of some dimensionality n, defined as $\mathbf{P}_n := \{X \in \mathbb{R}^{nxn} | X = X^T, X > 0\}$ [30], composed of symmetric strictly positive eigenvalued matrices. The manifold of possible $n \times n$ covariance matrices for a given EEG dataset recorded on an EEG system with n sensors is thus on the manifold $\mathbf{P}_n$. This manifold is Riemannian, and therefore we have a tangent space $\mathbf{T}_X\mathbf{P}_n$ in every point of this space [30].

Well-defined methods in the literature allow for various metrics and algorithms to be utilized on Riemannian manifolds. This includes distance metrics such as the Affine Invariant Riemannian Metric, and also allows for optimization on the Riemannian manifold [29]. This enables the usage of ML methods in combination with Riemannian geometry. Recently, the family of signal processing methods based on these covariance matrices on the Riemannian manifold of SPD matrices and the Riemannian metrics applicable thereof have proven promising as a basis for ML and TL in EEG devices [29, 8].

EEG decoding in Tangent space

One common method for using Riemannian geometry is to project the data onto the tangent space located at the geometric mean of a set of trials. The features are then the coordinates of the data after projection onto the tangent space [4].

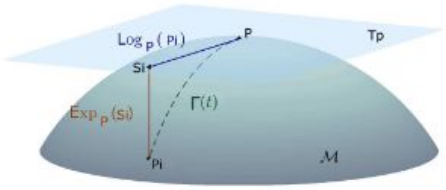

Figure 9. Tangent space of point P on the manifold M. S_i is a vector in the tangent space of P. P_i is the projection of S_i from the tangent space to the manifold M. $\Gamma i(t)$ is the geodesic between P and P_i" [4, Fig. 1]

As an example, a 4x4 covariance matrix given by an EEG headset with 4 sensors can be projected onto a tangent space, with the feature vector having 10 parameters. As the tangent space is a hyperplane, the use of classification algorithms that are based on projections onto hyperplanes is possible.

A pipeline using Riemannian geometry for ML classification for EEG signals could therefore consist of three steps: first, the covariance matrix is estimated for each epoch corresponding to an event. Next, the tangent space of the geometric mean of the trials is obtained, and the covariance matrices of the events are project onto it. Finally, the transformed feature vectors are used with classical ML classifiers e.g., Gaussian Process Classifiers (GPC), Support Vector Machine classifiers (SVC), etc.

Riemannian Procrustes Analysis for TL

In this project, we implement a method for TL using the Riemannian manifold of the covariance matrices: Riemannian Procrustes Analysis (RPA) [27]. In essence, it fits Riemannian-geometry-aware geometric transformations onto a source dataset of covariance matrices to transform it to a target dataset, with the goal of obtaining a statistical distribution as similar as possible between the two.

The advantage of this approach is twofold: geometric transformations enable the usage of previously discovered ML methods on $\mathbf{P}_n$ manifolds, while geometric transformations are not computationally intensive, and therefore can be used in real-time BCI experiments.

5.2 Implementation

Packages, environment

Alongside the standard array of python packages useful for scientific processing, there are a number of packages that are used to implement the machine-learning pipeline;

- Scikit-learn, a highly general and versatile ML package, provides the general ML pipeline and the overall codebase for data handling [24].
- EEG-notebooks [1], more of a development environment than a single package, it provides a general collection of EEG experiments and packages that greatly reduce the amount of package management work required. Built on brainflow, MNE and PsychoPy; all of which are therefore used in the ML environment [23, 11, 25].

- pyRiemann [2], a package built on MNE and Scikit-learn that implements appropriate Riemannian geometry methods for SPD matrices; most relevantly, tangent-space projection and covariance matrix calculation.
- RPA-01 [27], is the code/package for the proposed method of Riemannian Procrustes Analysis that this project extends to a laboratory experiment.

Simulated experiment

With methods discussed, the two following pipelines can be created for the ML code: one with transfer-learning and one without. The pipeline and abberviation for these are as follows:

- COV-RMTS-SVC pipeline: this is the 'prior-free' method that ignores previous experiments, and purely uses the data from the actual session. It consists of calculating the covariance matrices (COV), then projecting the data to the tangent space of the data mean with Riemannian methods (RMTS), and finally using the obtained hyperplane vectors with a Support Vector Machine Classifier.
- COV-RPA-RMTS-SVC pipeline: this is the method that implements transfer learning. It is identical to the COV-RMTS-SVC method except for the TL step. Between the COV and RMTS steps, the data from the current session is augmented by using prior data from past sessions. Before augmentation, the past data goes through an RPA step for its distribution on the Riemannian manifold to fit the current data.

To examine feasibility under more ideal laboratory circumstances, exploratory testing is first done by taking an existing database with multiple recorded sessions, and by 'simulating' an online experiment by feeding in the new session data in increasing chunks and recording cross-validation accuracy on data reserved from 'later' in the session. This is done on the motor-image dataset collected and used by the original RPA paper [27] in their git repository.

6. Results

6.1 Timeflux

Snapshots can be shown to prove that it is in fact working, and that data can be observed in the monitor (see Figure 10). One of the most challenging parts was to integrate Psychopy into Timeflux and being able to send an event when an image is presented in the screen so that it is recognized by the epoch node in the EEG preprocessing graph. This can be seen in the prompt (see Figure 11). To show in a more realistic way what Timeflux is doing, a more representative graph has been created offline using real collected data (see Figure 12).

6.2 Simulated Real time convergence in ML

Though performance is variable, preliminary experiments in comparing the near-identical TL and non-TL pipelines of COV-RMTS-SVC and COV-RPA-RMTS-SVC seem to show what we hoped to see; that when we have a low number of measurements in the actual session, transfer learning allows for better overall performance. This can be seen in Figure 13.

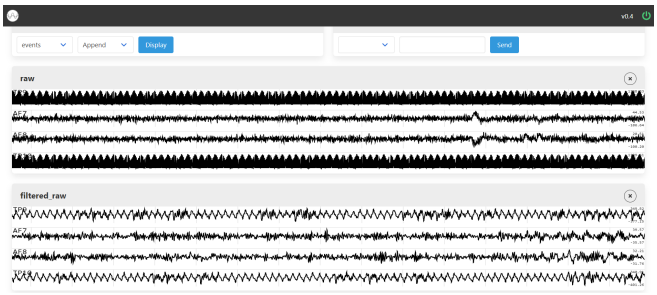

Figure 10. Monitor showing real time data: the top plot represents the raw initial data, and the bottom one after being filtered with a band-pass filter

```
Event pushed from psychopy <pylsl.pylsl.StreamOutlet object at 0x000001E9E92A4C88> ['start_epoching', '1']
LSL Inlets:  [['start_epoching', '1']] [1641586805.929295]
Epoch matches:                                    label data
2022-01-07 20:20:05.929295063  start_epoching     1
2022-01-07 20:20:05,965 DEBUG      epoch_1      35916     EEG_preprocessing There was an event match
2022-01-07 20:20:07,014 DEBUG      debug        35916     EEG_preprocessing
                                        TP9          AF7          AF8         TP10
2022-01-07 20:20:05.852093697 -139.986518 -233.718488  354.157763  -50.316878
2022-01-07 20:20:05.854125023 -130.825839 -222.740879  407.273764  -12.132997
2022-01-07 20:20:05.858031273 -118.995723 -193.520704  433.637649   16.279158
2022-01-07 20:20:05.861937523 -109.187182 -159.048450  431.665520   27.593165
2022-01-07 20:20:05.865843773 -103.979635 -134.102118  411.001363   22.209418
...                                    ...          ...         ...         ...
2022-01-07 20:20:06.913562536 -357.206632 -200.526260    7.411646  242.570998
2022-01-07 20:20:06.917468786 -352.474753 -185.916843   20.037709  293.006672
2022-01-07 20:20:06.921375036 -325.913117 -170.639483   42.126723  341.027710
2022-01-07 20:20:06.925281286 -281.208323 -158.475937   61.292564  372.994818
2022-01-07 20:20:06.929187536 -223.653888 -150.274252   65.051380  382.827501

[276 rows x 4 columns]
```

Figure 11. Snapshot from the prompt, showing how an event is sent from the experiment node (Psychopy program) and recognized by the LSL event receiver, as well as the epoch node – which prints the data corresponding to slightly more than one second.

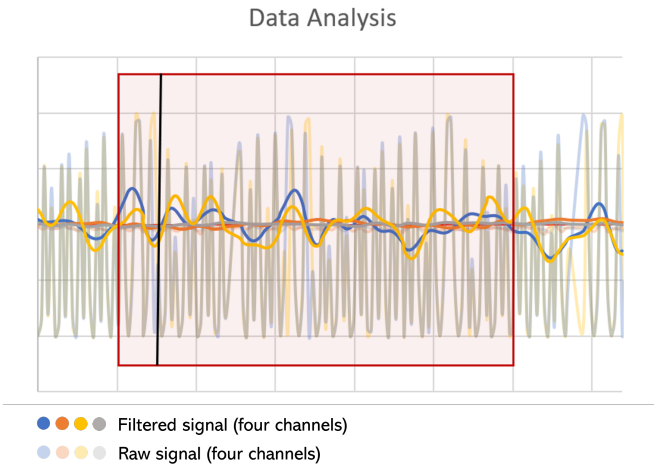

Figure 12. In bold, the filtered signal. In the same colors, but lighter, the raw data corresponding to that period of time. The red box represents the epoched time window, and the black line the instant in which the stimuli is presented. This is a representative image, so the P300 response might not be visible.

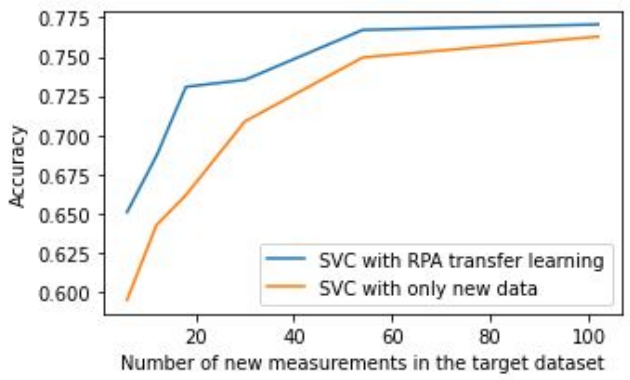

Figure 13. Simulated convergence using motor neuron data when using the same COV-RMTS-SVC pipeline without any prior data and the TL-based COV-RPA-RMTS-SVC pipeline that does utilize prior data. This simulated experiment shows the advantage of TL for a motor imagery dataset

As the number of new measurements increase, this lead slowly shrinks.

Note, however, that other datasets have not yet been investigated than the Motor imagery data from the cited paper and our own dataset [27].

6.3 Machine Learning applied to the experimental setup

Riemannian machine learning was successfully made to work with the data recorded from our experimental setup. This was done by taking two datasets, each containing 60 5-second BCI recordings using the experiment setup detailed in Section (4). Then the corresponding label of Furniture or Dog was added to each BCI recording, allowing for classification. Finally, the ML algorithms presented in Section 5 were applied to this data. Using standard covariance matrix calculation, they proved incapable of producing any results meaningfully above random chance, giving results of around 46-54%, irrespective of whether transfer learning was utilized or not. To test if the non-functional results were due to the method used or poor quality of data, alternative methods to Riemannian learning were also considered, such as the Random Interval Spectral Forest [16]. Though not originally part of the project plan, this resulted in a classification accuracy of around 60-75%, which demonstrates that the data itself is useful. As a next step, alternative means of covariance matrix calculation were investigated. This included using a special form covariance matrix dedicated to ERP processing, described in the paper [5]. This essentially enabled Riemannian methods to work properly for the data, obtaining a classification accuracy of 70-80%.

Therefore, we can draw the conclusion that for the purposes of the Muse-s in the designed experiment, the Riemannian ML is a viable pathway to enabling TL.

7. Discussion

7.1 General considerations

As expected, one of the tasks proving to be the most difficult is the merger of all the disparate parts of the project into a cohesive whole. (PsychoPy, Timeflux, and the ML and TL algorithms). Although theoretically possible, the integration involves dealing with unexpected problems. Mistakes were made early in the project when setting up python environments. Even though the team has difficult time running the project according to the planned schedule for various reasons, the project is completed. It has come to realization that working with a frontier field, such as BCI, makes the planning and estimating quite difficult.

7.2 Machine learning

From preliminary results on simulating the experiment with ML and prerecorded data, the TL approach using the COV-RPA-RMTS-SVC pipeline seemed promising as compared to having no TL. On the first attempt results obtained on motor image data in a clinical environment did not generalize well to our experimental setup using the Muse S headband and P300 data. Riemannian methods failed at gaining any meaningful classification accuracies above random chance in our experiments, with or without transfer learning being included. This issue was then overcome by using alternative methods to cal

In conclusion, we can see that the experimental setup itself is viable. Ultimately, this means that the ML methods tested in Riemannian transfer learning when combined with timeflux is a realistic path to experiment with reducing calibration time. Of course, considering how relatively new the field of Riemannian TL for EEG data is; the issues and details for implementing it for specific use-cases are not well known.

Furthermore, alternate methods for non-TL machine learning such as Random Interval Spectral Forest also seem to show results, so alternative paths to achieve TL can also potentially improve results.

7.3 Outlook and ethical aspects

In view of the ethics behind BCI in general and in our project, there are as presented in the introduction many positive use cases for this technology, including hearing aids and prosthetic limbs. It is also possible to use BCI for wheelchair control, giving more independence for the user. There are many more possible use cases for people in need that can be developed in this area. With a non-invasive BCI method similar to ours but with better instrumentation than in our case, it is possible not only to differ between dog and chair but to control a computer mouse or for even more. BCI could even potentially be used in the future for more day-to-day use cases like social media and gaming.

On the other hand, the potential of malicious usage of BCI can not be overlooked. This might include non-voluntary manipulation and mind reading, potentially even as a component in totalitarian rule. In conclusion, there are many positive use cases for BCI technology but also a risk for exploitation.

References

[1] B. Alexandre, B. Hubert, M. Dano, S. Ben, G. John, E. Amanda, M. Kyle, T. Jadin, and B. Erik. “Scikit-learn: machine learning in Python”. *BrainHack Ontario* (2020). URL: https://neurotechx.github.io/eeg-notebooks/index.html.

[2] A. Barachant. *Pyriemann v0.2.2*. https://github.com/alexandrebarachant/pyRiemann. 2015. URL: https://doi.org/10.5281/zenodo.18982.

[3] A. Barachant. "P300 with muse eeg headband". In: 2017. URL: http://alexandre.barachant.org/blog/2017/02/05/P300-with-muse.html.

[4] A. Barachant, S. Bonnet, M. Congedo, and C. Jutten. "Multiclass brain–computer interface classification by riemannian geometry". *IEEE Transactions on Biomedical Engineering* **59**:4 (2011), pp. 920–928.

[5] A. Barachant and M. Congedo. "A plug&play P300 BCI using information geometry". *CoRR* **abs/1409.0107** (2014). arXiv: 1409.0107. URL: http://arxiv.org/abs/1409.0107.

[6] S. Brandl, J. Höhne, K.-R. Müller, and W. Samek. "Bringing bci into everyday life: motor imagery in a pseudo realistic environment". In: *2015 7th International IEEE/EMBS Conference on Neural Engineering (NER)*. IEEE. 2015, pp. 224–227.

[7] P. Clisson, R. Bertrand-Lalo, G. V.-T. M Congedo, and J. Chatel-Goldman. "Timeflux: an open-source framework for the acquisitionand near real-time processing of signal streams". In: 2019. DOI: 10.3217/978-3-85125-682-6-17. eprint: https://timeflux.io/assets/pdf/Timeflux_GBCIC2019.pdf. URL: https://timeflux.io/.

[8] M. Congedo, A. Barachant, and R. Bhatia. "Riemannian geometry for eeg-based brain-computer interfaces; a primer and a review". *Brain-Computer Interfaces* **4**:3 (2017), pp. 155–174.

[9] P. E. Downing, A. W.-Y. Chan, M. V. Peelen, C. M. Dodds, and N. Kanwisher. "Domain Specificity in Visual Cortex". *Cerebral Cortex* **16**:10 (2005), pp. 1453–1461. ISSN: 1047-3211. DOI: 10.1093/cercor/bhj086. URL: https://doi.org/10.1093/cercor/bhj086.

[10] J. M. Gaylor, G. Raman, M. Chung, J. Lee, M. Rao, J. Lau, and D. S. Poe. "Cochlear Implantation in Adults: A Systematic Review and Meta-analysis". *JAMA OtolaryngologyHead & Neck Surgery* **139**:3 (2013), pp. 265–272. ISSN: 2168-6181. DOI: 10.1001/jamaoto.2013.1744.

[11] A. Gramfort, M. Luessi, E. Larson, D. A. Engemann, D. Strohmeier, C. Brodbeck, R. Goj, M. Jas, T. Brooks, L. Parkkonen, and M. S. Hämäläinen. "MEG and EEG data analysis with MNE-Python". *Frontiers in Neuroscience* **7**:267 (2013), pp. 1–13. DOI: 10.3389/fnins.2013.00267.

[12] S. Gudmundsson, T. P. Runarsson, S. Sigurdsson, G. Eiriksdottir, and K. Johnsen. "Reliability of quantitative eeg features". *Clinical Neurophysiology* **118**:10 (2007), pp. 2162–2171. ISSN: 1388-2457. DOI: https://doi.org/10.1016/j.clinph.2007.06.018.

[13] G. Hotson, D. P. McMullen, M. S. Fifer, M. S. Johannes, K. D. Katyal, M. P. Para, R. Armiger, W. S. Anderson, N. V. Thakor, B. A. Wester, et al. "Individual finger control of a modular prosthetic limb using high-density electrocorticography in a human subject". *Journal of neural engineering* **13**:2 (2016), p. 026017.

[14] C. Kothe, D. Medine, C. Boulay, M. Grivich, and T. Stenner. *What is lsl?* https://labstreaminglayer.readthedocs.io/info/intro.html. 2019.

[15] J. Kowaleski. *Bluemuse, version 2.2.0.0*. https://github.com/kowalej/BlueMuse. 2021.

[16] J. Lines, S. Taylor, and A. Bagnall. "Time series classification with hive-cote: the hierarchical vote collective of transformation-based ensembles". *ACM Trans. Knowl. Discov. Data* **12**:5 (2018). ISSN: 1556-4681. DOI: 10.1145/3182382. URL: https://doi.org/10.1145/3182382.

[17] D. J. McFarland, J. Daly, C. Boulay, and M. A. Parvaz. "Therapeutic applications of bci technologies". *Brain-Computer Interfaces* **4**:1-2 (2017), pp. 37–52.

[18] D. Morrison, alexandre barachant, J. Kowaleski, hubertjbanville, S. Chevallier, U. Shaked, xloem, E. Bjäreholt, J. J. T. Tresols, H. Banville, S. Jawhar, pdpino, stepheninx, and T. Zhao. *alexandrebarachant/muse-lsl: Muse LSL v2.1 with adaptive timestamp correction, hci-tool workaround, configurable device presets, and more bugfixes*. Version v2.1.0. 2021. DOI: 10.5281/zenodo.4540829.

[19] C. Mühl, C. Jeunet, and F. Lotte. "Eeg-based workload estimation across affective contexts". *Frontiers in neuroscience* **8** (2014), p. 114.

[20] C. S. Nam, A. Nijholt, and F. Lotte. "Brain-computer interfaces handbook: technological and theoretical advances". In: 2018.

[21] NeurIPS. *Beetl competition*. https://beetl.ai/challenge. 2021.

[22] S. J. Pan and Q. Yang. "A survey on transfer learning". *IEEE Transactions on knowledge and data engineering* **22**:10 (2009), pp. 1345–1359.

[23] A. Parfenov. *Brainflow*. https://github.com/brainflow-dev/brainflow. 2021. URL: https://brainflow.org/.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: machine learning in Python". *Journal of Machine Learning Research* **12** (2011), pp. 2825–2830.

[25] J. Peirce, J. R. Gray, S. Simpson, M. MacAskill, R. Höchenberger, H. Sogo, E. Kastman, and J. K. Lindeløv. "Psychopy2: experiments in behavior made easy". *Behavior research methods* **51**:1 (2019), pp. 195–203.

[26] J. Polich and T. Bondurant. "P300 sequence effects, probability, and interstimulus interval". *Physiology & Behavior* **61**:6 (1997), pp. 843–849.

[27] P. L. C. Rodrigues, C. Jutten, and M. Congedo. "Riemannian procrustes analysis: transfer learning for brain–computer interfaces". *IEEE Transactions on Biomedical Engineering* **66**:8 (2018), pp. 2390–2401.

[28] J. R. Wolpaw. “Brain–computer interfaces”. In: *Handbook of Clinical Neurology*. Vol. 110. Elsevier, 2013, pp. 67–74.

[29] F. Yger, M. Berar, and F. Lotte. “Riemannian approaches in brain-computer interfaces: a review”. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* **25**:10 (2016), pp. 1753–1762.

[30] P. Zanini, M. Congedo, C. Jutten, S. Said, and Y. Berthoumieu. “Transfer learning: a riemannian geometry framework with applications to brain–computer interfaces”. *IEEE Transactions on Biomedical Engineering* **65**:5 (2017), pp. 1107–1116.

Project in Automatic Control
Ball-E

Samuel Gunnarsson[1] **Oskar Nilvéus Olofsson**[2] **Alexander Persson**[3]
Johan Siwerson[4]

Project Advisor: **Martin Gemborn Nilsson**

[1]sa8664gu-s@student.lu.se [2]os2323ol-s@student.lu.se [3]alexandermikael@gmail.com
[4]jo8447si-s@student.lu.se

Abstract: The project's main objective is to implement proven control algorithms and prior control and robotics knowledge in order for a ball-balancing robot to stabilize around an upright equilibrium point. The robot consists of a basketball and a tower like structure. The tower like structure and the basketball are in contact through three stepper motors and omni wheels. The omni wheels in combination with the basketball allows for movement in any horizontal direction. The system is modeled using the Euler-Lagrange method [3] in two vertical planes and it is assumed that there is no coupling between them. System simulations and linearization of the model has been performed in MATLAB. It is by sending control signals to the motors, that enables movement and balance. For controlling, LQR is used and the system receives user input through a Bluetooth module. Results show that the robot is able to balance on top of the basketball, however, only for around a minute.

1. Introduction

The goal of this project is to utilize our prior knowledge and further explore the field of control. The main objective is to make a robot balance around an equilibrium point on top of a basketball, with the help of three stepper motors and A4988 drivers, an inertial measurement unit (IMU) and an Arduino Mega 2560. The secondary objective is to make the system cordless by implementing a built in energy source i.e. a battery. Furthermore, if time allows, the group will try to make the robot be able to move around in its environment. Throughout the project several control algorithms will have to be evaluated to find the optimal solution for the problem. The first and final version of the robot can be seen in Figure 1.

2. Modeling

A pendulum that wants to be stabilized in it's upmost position on top of a movable cart is a well known control problem [5]. Although a body on top of a ball may look different at first glance, theory from the standard inverted pendulum should be able to be utilized to control this system as well. One of the biggest differences between the inverted pendulum on a cart and a body on a ball is the increase of dimensions, going from two to three dimensions. With this in mind a simplified physical model, modeling the system in 3 independent 2D planes (XY, XZ and YZ), based on differential equations describing torques and forces acting on the system were decided as the way to model the system instead of grey or black box models based on data. Although some dynamics of the complete system are not captured with 3 independent 2D planes they were considered negligible for this project. Views of the YZ and XY planes can be seen in Figure 2. Torques applied on

(a) First version **(b)** Final version

Figure 1. First and final versions of the self-balancing robot.

the blue virtual wheels in Figure 2 plane can be mapped into stepper motor torques with the transformation in Equation 1 using the angle γ the motors are mounted at in relation to the flat surface of the robot.

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} -\frac{2r_w}{3r_k \cos\gamma} & \frac{r_w}{3r_k \cos\gamma} & \frac{r_w}{3r_k \cos\gamma} \\ 0 & -\frac{\sqrt{3}r_w}{3r_k \cos\gamma} & \frac{\sqrt{3}r_w}{3r_k \cos\gamma} \\ \frac{r_w}{3r_k \sin\gamma} & \frac{r_w}{3r_k \sin\gamma} & \frac{r_w}{3r_k \sin\gamma} \end{bmatrix} \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \quad (1)$$

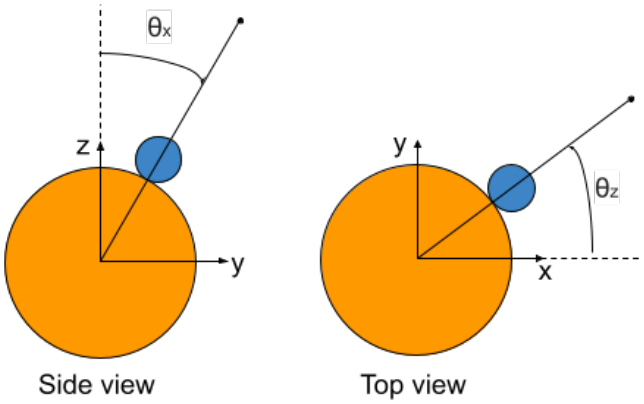

Figure 2. Definitions of the YZ and XY planes.

2.1 Equations of Motion

The equations of motion can be seen in Equation 2. It consists of four matrices, with three of them depending on the current state of the robot (i.e. position, angle and their velocities) described in Equation 3, as well as physical constants of the system. These are derived using the Lagrangian method. The matrices stands for the following in the system: M for masses and inertias, C for the coriolis forces, G for the gravitational forces and E for external forces, such as the forces from the stepper motors. All relevant and important parameters of the system can be seen in Table 1. [2]

$$M(q_x)\ddot{q}_x + C(q_x, \dot{q}_x)\dot{q}_x + G(q_x) = E\tau_x \tag{2}$$

$$q_x = \begin{bmatrix} y \\ \theta_x \end{bmatrix} \tag{3}$$

Rewriting the equations of motion and solving $\ddot{q}$ gives Equation 4. The state space form of this can be seen in Equation 5.

$$\ddot{q}_x = -M^{-1}C\dot{q}_x - M^{-1}G + M^{-1}E\tau_x \tag{4}$$

$$\begin{bmatrix} \dot{y} \\ \dot{\theta}_x \\ \ddot{y} \\ \ddot{\theta}_x \end{bmatrix} = \begin{bmatrix} 0_{2\times2} & I_{2\times2} \\ 0_{2\times2} & (-M^{-1}C)_{2\times2} \end{bmatrix} \begin{bmatrix} y \\ \theta_x \\ \dot{y} \\ \dot{\theta}_x \end{bmatrix} - \begin{bmatrix} 0_{2\times1} \\ (M^{-1}G)_{2\times1} \end{bmatrix} + \begin{bmatrix} 0_{2\times1} \\ (M^{-1}E)_{2\times1} \end{bmatrix} \tau_x \tag{5}$$

where $X_{n\times m}$ represents a matrix with n rows and m columns.

Equation 5 represents the non-linear system for the YZ plane (analogously for the XZ plane with different variables) and was used to simulate the system in Simulink and linearize the system around an equilibrium point to obtain a linearized system to control, used for control synthesis.

2.2 Linearization

As previously mentioned the model is analogous in the XZ and YZ plane, so the same linearization can be applied to both of the planes. In this subsection, only the latter is presented. Linearizing Equation 5 around $x_0 = \begin{pmatrix} y & \theta_x & \dot{y} & \dot{\theta} \end{pmatrix}^T = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^T$, where the index x means that we are in the YZ plane, yields:

$$\dot{x} = Ax + Bu \tag{6}$$

Description	Parameters
Mass of ball	m_k
Radius of ball	r_k
Inertia of ball	I_k
Mass of body	m_a
Body inertia	I_x, I_y, I_z
Radius of omni wheels	r_w
Intertia of omni wheels	I_w
COM height body	l
Gravity constant	g
Zenith angle	γ
External torque	τ_x, τ_y, τ_z

Table 1. Description of parameters used for modeling. Observe that COM stand for Center of Mass and that the external torque represents the torque that will be applied on the blue virtual wheels in Figure 2. The zenith angle describes the angle the motors are mounted in relation to the horizontal surface of the robot.

where

$$A = \begin{pmatrix} 0 & I \\ -M(x_0)^{-1}\frac{\partial G(x_0)}{\partial x_x} & 0 \end{pmatrix}, B = \begin{pmatrix} 0 \\ M(x_0)^{-1}E \end{pmatrix} \tag{7}$$

and

$$x_x = \begin{bmatrix} q_x \\ \dot{q}_x \end{bmatrix} = \begin{bmatrix} y \\ \theta \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{8}$$

From now on we will ignore the position states, $x, y, \dot{x}$ and $\dot{y}$, since the main objective is to balance the robot in the upright position.

2.3 Converting torque to motor speed

The output of the control algorithm consists of three torque values, representing the necessary torques in each plane to balance the robot. These values need to be converted into angular velocities that can be given to the stepper motors. This is done using the equation for angular momentum, as in Equation 9, where I is the moment of inertia and ω is the angular speed, and the relation between angular momentum and torque, see Equation 10. The combination of these equations will give the formula needed to convert torque to angular speed for the stepper motors, as seen in Equation 11.

$$L = I \times \omega \tag{9}$$

$$\int \tau dt = L \tag{10}$$

$$\omega = \frac{L}{I} = \frac{\int \tau dt}{I} \tag{11}$$

Another easier, but not as accurate, approach to translate a torque to an angular velocity for stepper motors, is to assume a linear relationship between the two as in Equation 12

$$\omega = J_k \times \tau \tag{12}$$

where J_k is a constant tuned experimentally. This approach will be used initially to get a functional prototype and Equation 11 will replace it if deemed necessary.

Item	Qty	Cost/Item
Stepper motor driver	3	40
Stepper motor	3	500
Omni Wheels	3	100
Arduino Mega 2560	1	400
Copper plate	1	50
IMU (LSM9DS1 Breakout)	1	300
Battery (Li-Po 3S 11V)	1	250
Battery voltage reader	1	50
Bluetooth chip (HM-10 BLE Module)	1	150
Basketball	1	200
Total:		3320

Table 2. List of equipment, with the corresponding quantity and cost per item in SEK. The prices are estimated from various sources online and smaller equipment such as wires and tools are not listed. Qty is an abbreviation for quantity.

3. Electro-Mechanics

The hardware used for the robot can be seen in Table 2, along with bolts, nuts, three threaded rods, three angle brackets, two acrylic plates. The first version of the robot, that was supplied during the start of the project, was a simple version that only contained the necessary component such as the Arduino, the IMU and the stepper motors and their driver (this version can be seen in the left part of Figure 1). So to expand the system and increase its adaptability, a battery, a battery voltage reader and a Bluetooth chip were installed on the system. The circuit on the breadboard was redesigned and soldered on a smaller copper plate to eliminate some of the otherwise necessary wires. See Figure 1 for a comparison of the given and updated robot and Figure 3 for the redesigned circuit.

The main advantage of using stepper motors for this project was the need for precise movement. This causes them to have good speed control and precise positioning, with the only big disadvantage being that it happens that they miss stepping because of heavy load. The way to control the motors is to send a digital pulse to the stepper motor drivers, which in their turn will control so that the connected motor operates correctly in regard with direction and the amount of stepping to do. The drivers also allows to set the precision of the steps. The motors themselves then converts digital pulses into mechanical shaft rotation. The configuration of the stepper drivers can be seen in Figure 3, where the main point is that the pins for setting microsteps are all set to high, meaning that the stepper motors will move one sixteenth of a step for every pulse sent. The effect of this is to have the highest precision possible for the movement of the stepper motor.

3.1 Hardware and Software implementation

The relation between hardware and software can be seen in Figure 4. The modules used in this project are the following: Wireless interface, IMU controller, Stepper controller, EEPROM interface and finally LQR controller. Each one of these modules will handle a specific part and communicate directly with the Main module to update values or perform some action. Following paragraphs will in more detail explain the content and application of each module.

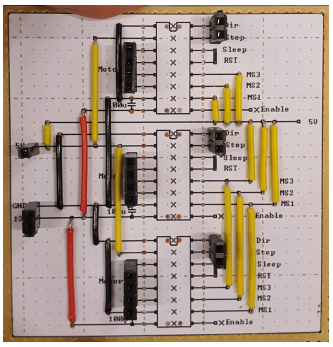

Figure 3. The redesigned circuit on the copper plate.

The Main module handles the setup for all the other modules and then runs a loop that for each iteration lets the other perform their action. The important content of the Main is the current values read from the IMU, the LQR parameters and the current torque values that are to be applied on each motor.

To easily handle setting and getting different parameters, the EEPROM chip on the Arduino will be used to store these values. The EEPROM module will handle these operations, together with reading all the parameters at startup.

Modifying what values to set can then be done using Bluetooth communication, through a mobile phone app called ArduinoBlue [1]. The Wireless module will be in constant communication with the Bluetooth chip, and may receive and send data to be used in the robot. Together with the EEPROM module this will facilitate tuning the parameters, as well as perform actions such as turning on or off the motors.

The IMU chip will be handled by the IMU module, where the values from the gyroscope, accelerometer and magnetometer will be processed. These three sensors will be polled approximately every 10 ms and then sent to a complementary filter (described at the end of Section 4.2) to reduce the amount of noisy data. These values will then be used as input for the LQR module, and will thus be sent to the Main module to update the current state of the robot.

The LQR module will handle the control algorithm, further described in Section 4.1. Using the sensor values from the IMU module, the LQR will output the torque that needs to be applied from the motors.

Stepper Control is used to control the three stepper motors through PWM signals. An important aspect of this module is the ability to calculate the correct speed to apply, based on the torque calculated from the LQR module.

4. Control

4.1 Control design

A system of an inverted pendulum like ours (linearized around an equilibrium point) is suitable with linear quadratic control [5]. This minimizes an accumulative cost function J quadratic in states x_k and control signals u_k

$$J_t = \sum_{k=t}^{t_{final}} x_k^T Q x_k + u_k^T R u_k \tag{13}$$

where Q and R are positive definite *state* and *control* matrices respectively. Since we want the robot to balance on top of

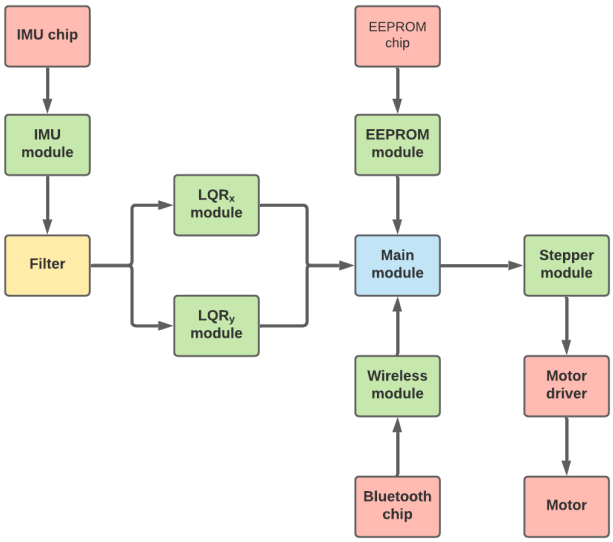

Figure 4. A flow chart for the hardware and software in the project. The green squares represent modules in the Arduino code (i.e. software), the yellow square represents filter and the red squares represent the hardware.

the ball indefinitely, $t_{final} = \infty$. To implement computerized digital control, the dynamic equations (see Equation 6) are discretized using zero-order-hold as $\Phi = e^{A\Delta t}$, $\Gamma = \int_0^{\Delta t} e^{As} ds\ B$ and the digital LQR control is then given by:

$$u_k = -R^{-1}\Gamma^T P x_k \equiv -K x_k \tag{14}$$

where P is a steady state solution obtained by solving the discrete-time algebraic Riccati equation:

$$P = \Phi^T P \Phi - (\Phi^T P \Gamma)(R + \Gamma^T P \Gamma)^{-1}(\Gamma^T P \Phi) + Q \tag{15}$$

Since the linearization accurately represents the system in the equilibrium, the LQR control will be optimal with respect to stabilizing the plant close to the equilibrium.

4.2 Measuring states

Calculations of the tilt angles of the robot, θ_x and θ_y can be seen in Equation 16

$$\theta_x = \arctan \frac{y}{\sqrt{x^2 + z^2}}, \theta_y = \arctan \frac{x}{z}, \tag{16}$$

where x, y and z corresponds to the raw data from the IMU.

These measurements were considered noisy and therefore combined with the gyroscope in a complementary filter [4], essentially low pass filtering the accelerometer data and high pass filtering gyro data to obtain a more accurate estimation of the angles while avoiding drift caused by integrating values from the gyroscope. This can be seen in Equation 17:

$$\theta = \alpha(\theta + \dot{\theta}_{gyro} \cdot \Delta t) + (1 - \alpha) \cdot \theta_{acc} \tag{17}$$

where α typically is some values close to 1, i.e. 0.98, to get the correct filter characteristics.

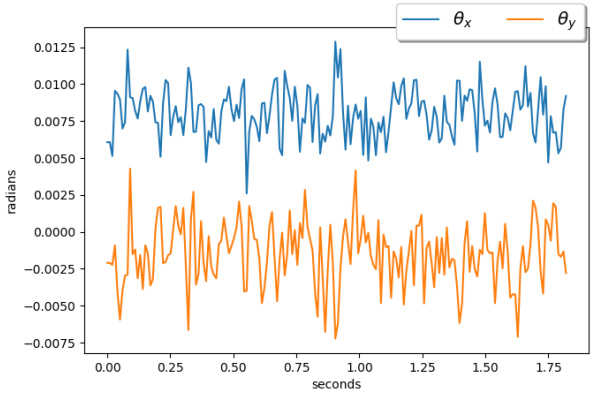

Figure 5. Plot of roll and pitch angles while manually assisting the robot to stand in the upright position. Used to estimate the offset of the accelerometer.

4.3 Calibration of sensors

Looking at the raw data from the accelerometer and the gyroscope while standing still in an upright position rose suspicion that the sensors were not calibrated and outputed values with a somewhat constant offset. In order to compensate for this, two experiments were performed.

Accelerometer offset To calibrate the accelerometer the robot was put upon the ball in a position where a human barely needed to support the robot for it to stand upright. While in this position, readings of θ_x and θ_y were collected. These are plotted in Figure 5. To compensate for the offset in software the mean value of the data points was calculated offline once and subtracted from every θ_x and θ_y calculation. The obtained mean values can be seen in Table 3.

Gyroscope offset Since a gyroscope relies on relative movements instead of an absolute position (like the accelerometer) the robot was placed on the ground and measurements from the gyroscope were collected. These are plotted in Figure 6. A mean value for $\dot{\theta}_x$ and $\dot{\theta}_y$ were calculated once offline and directly subtracted from the sensor readings in software. The obtained mean values are available in Table 3.

θ_x	-0.01331 (rad)
θ_y	0.01949 (rad)
$\dot{\theta}_x$	-0.00846 (rad/s)
$\dot{\theta}_y$	0.011557 (rad/s)

Table 3. Offsets for the accelerometer and gyroscope.

4.4 Motor filter

Although compensation of the sensors' offsets helped to balance the robot, fluctuations in the readings and vibrations in the system caused torque spikes in the motors. These spikes induced even more vibrations into the system and to counteract this a motor low-pass filter were implemented before sending the control signals to the motors. The details of this can be seen in Equation 18:

$$\omega_{filtered} = \beta\omega + (1 - \beta) \cdot \omega_{filtered} \tag{18}$$

where β is some value close to 0.

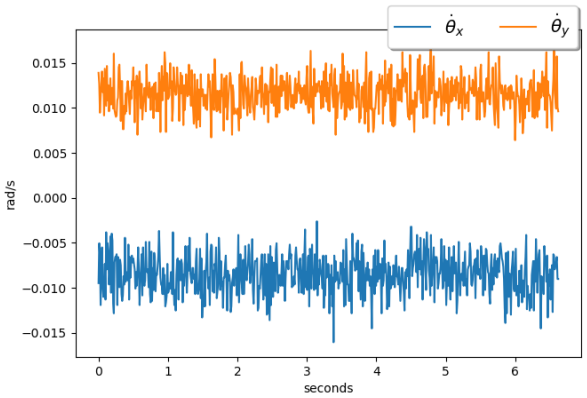

Figure 6. Plot of $\dot{\theta}_x$ and $\dot{\theta}_y$ while stationary. Used to estimate the offset of the gyroscope.

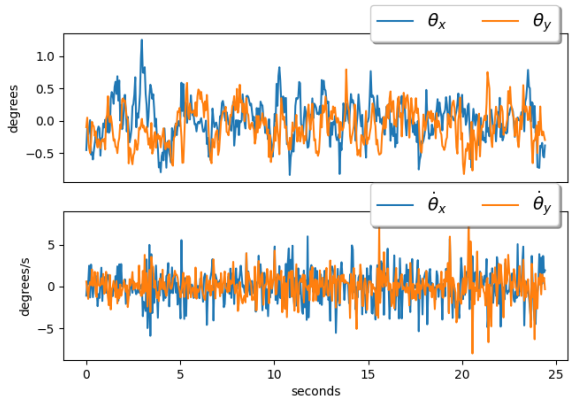

Figure 7. Plot of θ_x, θ_y (from the complementary filter), $\dot{\theta}_x$ and $\dot{\theta}_y$ while balancing on top of the ball.

5. Results

5.1 Control design parameters

To obtain a LQR controller that balances the robot in the upright position is an iterative procedure and most of the tuning of the Q and R matrices were done with the help of the simulation of the non linear system in Simulink. Using $Q = diag[0.6, 8]$ and $R = 0.05$ gave satisfying results in the simulation. However, when trying these values out on the real system, it resulted in a too weak control signal and the values were increased by 50%. These parameters resulted in a control law, K that looked like:

$$K_x = K_y = [k1, k2] = [0.1903, 0.4871] \tag{19}$$

5.2 Real world results

Using the parameters in Table 4, starting the system in it's upright position gave the plots in Figure 7.

The time axis of Figure 7 is cropped in order to show the angles and their velocities in more detail. On average, the robot manages to balance on top of the ball for approximately 40 seconds before it falls off.

6. Discussion

6.1 Result analysis

As Figure 7 shows, the robot managed to balance on top of the ball in a desired way. The tilt angles are kept low and the system quickly reacts to changes, keeping all states $\theta_x, \theta_y, \dot{\theta}_x$ and $\dot{\theta}_y$ close to zero. By observing the robot, the main reason to why the robot does not balance on top of the ball forever seems to be due to vibrations caused by the frame not being stiff enough, causing behaviours that the model does not capture.

6.2 Software implementation

After connecting the IMU chip to the Arduino it was observed that the values for θ_x and θ_y were quite fluctuating, changing with a $\pm 1°$in a rather short period of time. Thus a complementary filter was implemented to keep the values more stable, using the formula found in Section 4.1.

$k1$	$k2$	J_k	α	β	γ
0.204	0.492	40	0.98	0.02	45°

Table 4. Final parameters used on the system

Another problem that occurred during development was that the speed of the motors did not quite reach a high enough velocity for the robot the be balance. The main reason for this was that the IMU module read the position too often, which took too much time and did not allow the Arduino to control the three stepper motors. The solution was then to limit the amount of reads, to once every 10 ms.

One of the main points of discussion during the projects course was on how to convert the torque value from the control algorithm to a motor speed value to apply for each motor. The easy way of solving this is to assume a linear relation between torque and speed, and thus only multiply the torque with some constant. This constant could be found by doing some rather simple experiments, in other words try out different scenarios until a satisfied value has been found. But the more correct way of doing this conversion is described in Section 2.3. This theoretical relation between torque and speed seemed like the better way, but as a start the first method was implemented to get the robot working. Unfortunately the project ran out of time before the theoretical relation could be implemented, thus this method remains untested.

6.3 Sources of error

The basketball provided with the project was old and not completely spherical. This was not modelled and made it hard for the robot to balance on top of the ball for certain ball positions. Another source of error was that the structure of the robot was not very rigid. The model didn't take this into account and gave rise to unwanted vibrations in the structure which the control algorithm did not handle well. Finally the different intertias and the center of mass of the system was not measured. Instead they were approximated due to lack of measuring equipment.

6.4 Future work

Because of the time limitation a 3D printed case could not be designed and printed. Therefore one possible improvement for this project is to design a good case, as to both boost the appearance but to also have a better weight placement,

to have more mass near the z-axis (origin of the XY plane). This would result in a smaller uncertainty of the center of mass. Making the robot more rigid with a better structural construction, would also make the system easier to control due to reduced vibrations.

Another optional task that was not implemented during the run of this project was the ability to make the robot move in the room. This could for example be achieved with the model described in Section 2. and by adding sensors measuring the relative position of the robot to the environment and another control algorithm for position control. However, due to the short period of time assigned for this project, we focused on achieving a reliable stabilization of the robot instead.

A problem with the stepper motors was that they are dependent on the capability of the Arduino. If the Arduino is operating with a too high CPU load then the motors will decrease in velocity, as they are controlled with pulse-width modulation (PWM). Thus if the pulses are delayed the speed will decrease. A way to avoid this is to control each stepper motor with their own MCU, to prevent the PWM signals from being dependent on what other processes the CPU is performing. A simple form of communication can then be used between the Arduino and each MCU, e.g. letting the Arduino send velocity commands to the MCU.

Other control approaches such as nonlinear control, PID control or some sort of machine learning control could also be interesting to test and compare with.

6.5 Final words

This project has been a great exercise in control theory, robotics and simulation. While also covering important topics such as project planning, group dynamics and critical thinking. The project has not simply been a straight path, but rather a path of overcoming difficulties and obstacles, as it should be. However, it has been quite rewarding seeing the project come together bit by bit, finally reaching a final product of a balancing robot. Obstacles have been overcome by debugging, finding new solutions and by consulting with our supervisor.

References

[1] *Arduinoblue: an ios/android app for controlling arduino over bluetooth*. URL: https://sites.google.com/stonybrook.edu/arduinoble/ (visited on 2022-01-18).

[2] K. van der Blonk. *Modeling and control of a ball-balancing robot*. URL: https://essay.utwente.nl/65559/1/vanderBlonk_MSc_EEMCS.pdf (visited on 2021-12-01).

[3] *Lagrange equations (in mechanics)*. URL: https://encyclopediaofmath.org/index.php?title=Lagrange_equations_(in_mechanics) (visited on 2022-01-18).

[4] Pieter-Jan. *Reading a imu without kalman: the complementary filter*. URL: https://www.pieter-jan.com/node/11 (visited on 2021-12-13).

[5] I. Siradjuddin, B. Setiawan, A. Fahmi, Z. Amalia, and R. Erfan. "State space control using lqr method for a cart-inverted pendulum linearised model". **17** (2017), pp. 119–126.

Tracking of a high precision robot

Abdullah Shahin[1] **Vinay Venkanagoud Patil**[2]

[1]sve15ash@student.lu.se [2]vi0507pa-s@student.lu.se

Abstract: The project is centered around the evaluation of sensor fusion with the help of a filtering algorithm to filter out noise from the process and the measurement of a high precision robot that is being built for MAX IV, the filtered position estimate is used to track the robot in the environment. The filter in focus is the extended Kalman filter (EKF). This is a continuation of the work done by the author, Abdullah Shahin and Vinay Venkanagoud Patil, the collaboration will use the dual robot (omnidirectional/delta) that has been built by Vinay and implement the filters that Abdullah worked on in the localization node, the main focus is to find the measurement model of the IMU and Encoders. Further, the project will work on the inverse kinematics of the robot, so that the control signal units correspond to the measurement units, the final result has been tested at MAX IV, the robot is equipped with 3 omni wheels and 3 actuators that has a delta configuration. The results of this project was a maximum radius error of 3.5mm, this is more than a 10 fold improvement in the precision of the robot as our most accurate sensor has a 4cm resolution.

1. Introduction

In this project, a fine-tracking mobile robot for high-precision positioning and localization is implemented and used at the beam-line laboratory MAX IV in Lund, Sweden. The requirement comprises of the positioning and marking of relatively exact points on concrete floor where the beam line equipment and machinery will be positioned. Currently, there are 16 funded beam-line experiments and 6 are being installed. All of the beam-line equipment has been placed manually by construction workers. This machinery needs to be placed very accurately since the radiated beam itself is highly sensitive to deviations along its path and will have a direct impact on the resolution this beam has. Therefore, a poor placement will interfere with the experiments and measurements. A computer based blue lining system is being used to map the construction area in order to help the engineers to place the aforementioned equipment. Thus, the task is repetitive and physically demanding for the workers due to the goal precision of ±60 microns they need to reach for each mark using the current equipment. Consider that there are a couple of hundred points to be marked. Another drawback is the time the worker spends to reach each point because of the natural inaccuracies of the human hand. Thus, it takes several tries to reach the position. It is so a highly accurate robotic system with advanced control techniques is required to accomplish this task more efficiently and in less time.

Previously, the authors (Patil V., Carrera L.) [2] have implemented a dual robotic system for accurate positioning consisting of a Delta-configuration robot over an omnidirectional mobile robot. This system features the coarse navigation (omnidirectional) and the fine positioning (Delta) in two consecutive stages, once the omnidirectional robot has reached a reasonable ±2 centimeters in radius from its target the Delta robot goes into action by fine positioning the end effector to the target with an estimate of ±300 microns of error from the actual target. Although the robot is capable of a considerable high accuracy on its positioning it does not perform at its full potential due to the control system employed. Alternatively to this work, the author (Shahin A.) [4] has developed the control software intended to solve the blue lining task using advance control techniques with outstanding localization and navigation features. The objective was to investigate, via simulation, an EKF (Extended Kalman Filter), an UKF (Unscented Kalman Filter) and a PF (Particle Filter) over a car-like steering vehicle and its corresponding state-space model. This work concluded that the filters have potential to navigate the MAX IV robot with accuracy, the simulation had both process and measurement noise applied to it. Therefore, and in order to take advantage of the full potential of the authors work, the authors have studied the adaptation of these filters on the omniwheel/delta dual robot and investigated with potential implementation in this dual robot for later testing on-site at MAX IV, as this robot will run the algorithm on a raspberry pi, the PF is computationally demanding, the UKF a more advanced version of the EKF and thus is more demanding on the group to implement. The objective is therefore to reach a fine positioning suitable enough to perform the blue lining process with great accuracy and cheap but reliable components with an EKF algorithm. The book [5] is used extensively during the previous project, that was investigated by (Shahin.A) and has been continued, with a real world robot with more sensors and actuators than the simulated robot, the chapters studied will mainly be chapter 1,2,3,7 and 8.

2. Modeling

The present project has a list of key components in which the functioning is based on. It is described from the hardware components to the subsystems they form. Also, the essential math background is described for each subsystem so the whole model can achieve the localization and navigation process.

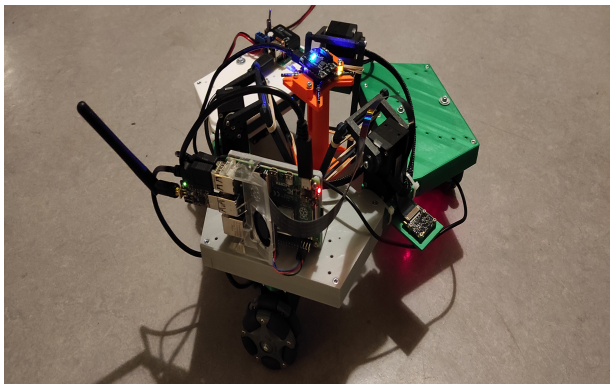

Figure 1. Dual robot (omnidirectional/delta) used in this project

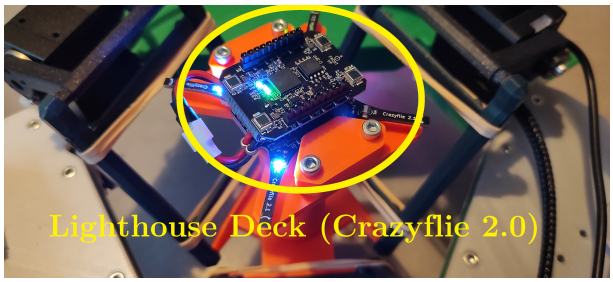

Figure 2. Crazyflie sensor (Lighthouse Deck)

IMU sensor

Since the dual robot uses relative position and rotation measurements, the considered sensor for this setup is the 6-DOF (Degrees Of Freedom) IMU (Inertial Measurement Unit) which is mounted on the center of the omnidirectional robot frame.
The crazyflie drone has been retained to compute the inertial measurements and use the sole measurements from the on-board accelerometer and gyroscope. This sensory data has been logged using the crazy-radio(RF interface to control and log data from the crazyflie drone). The sensory data logged from the crazyflie drone will consist of X and Y position using the lighthouse deck, acceleration along X and Y using the accelerometer. However, more accurate and reliable IMU's could be mounted on the robot in the future

Wheel Encoders Along with the IMU sensor, wheel encoders are used to extract wheel velocities which are in turn transformed to the robot body velocities using the forward kinematics of the robot. The encoder data is polled from the dynamixel motor [1] every instance a control signal is sent in. Additionally, the encoders on the motor provide us a resolution of 4096 ticks per revolution which corresponds to 0.00153radians.

Leica Absolute Tracker The Leica Absolute Tracker is a laser based device which uses a laser beam directly pointed to a reflector to estimate with metrology-grade accuracy the 3D-position, see Figure 3. In the dual robot, the Leica reflector has been installed in a position near the IMU sensor in order to obtain the position of the robot with respect to the Leica tracker. This enables the robot to know its initial position

Figure 3. Leica Absolute Tracker

in the space and compute the trajectory to the target. The important role of the Leica Absolute tracker in this project is to provide us with absolute position in space during the initial calibration and also help measure the performance of the localization on arrival to the target.

Crazyflie Lighthouse deck

The lighthouse deck is one of many decks developed by Bitcraze. This deck is custom designed to acquire position data using the htc vive lighthouse base stations. The base stations used produce a light signal that sweeps in the horizontal and the vertical plane with a unique frequency which is quite similar to a conventional lighthouse we see at the sea shores. The flydeck is equipped with 4 mirrors that are photo sensitive that sense the lighthouse signals to compute its position from the lighthouse.

Omnidirectional Robot

The considered model is the omnidirectional setup shown in Figure 4.

An omnidirectional platform is used since it can perform translations in any direction without the need to reorient. Moreover, due to its symmetric construction, the omnidirectional platform can also rotate about its Z axis seamlessly. These features are achieved by using the resulting velocities of the omnidirectional wheels that are controlled dictated by its kinematics. The command to the robot is in form of $u_t = (V_x, V_y, \omega_z)$ which is the body velocities in X and Y, and the angular velocity along Z. Thus, the robot kinematics equation will correlate the command vector variables with the actual controllable variables in the omnidirectional platform i.e. the wheel velocity of each wheel $(\omega_1, \omega_2, \omega_3)$. This relation is derived from the inverse kinematic transformation of the robot.

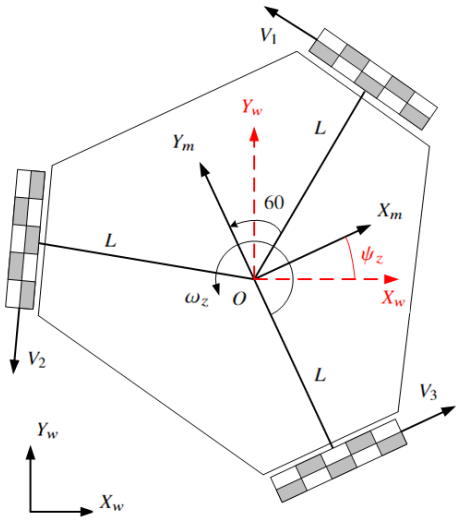

Figure 4. Model of the omnidirectional robot

2.1 State Space Model

First, the state transition model for this setup is described in Eq. 1

$$x_t = g(u_t, x_{t-1}) + v_t \tag{1}$$

Where, x_t is the state vector, the control vector u_t, v_t is a Gaussian process noise in the form of a Gaussian vector.

The state vector x_t is given by (Eq. 2),

$$x_t = (x_{world}, y_{world}, \psi_z)^\top \tag{2}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi}_z \end{bmatrix} = \begin{bmatrix} (u_1) * cos(\psi_z) - (u_2) * sin(\psi_z) \\ (u_1) * sin(\psi_z) + (u_2) * cos(\psi_z) \\ u_3 \end{bmatrix} \tag{3}$$

Where x and y are the positions in the world coordinates, u_1(linear velocity along X), u_2(linear velocity along Y) and u_3(angluar velocity along Z) are the control velocities, ψ_z and ω_z are the angular position and angular velocity respectively.

The control vector is given by (Eq. 4),

$$u_t = (V_{xref}, V_{yref}, \omega_{zref})^\top \tag{4}$$

Where u_t is the control signal containing the body velocities in the x,y and ψ direction respectively. This is used to predict the estimated state in EKF algorithm.

Measurement Model

From Eq. 5 and the state vector in Eq. 2 the measurement model is then given by the vector,

And the mobile robot measurements,

$$y_t = h(x_t) + e_t \tag{5}$$

Where e_t is a Gaussian measurement noise in Eq. 1 and Eq. 5 accordingly.

The y_t is the measurement vector containing the measurements form the IMU sensor, Light house position data and the encoder data, the IMU sensor outputs data from the accelerometor which is the measure of a_x, a_y this data has been fused with the Light house data which outputs the position for X_{world}, Y_{world} and ψ_{body} and the encoder data V_x,V_y and w_z to correct the estimated state from the control signal. This has been done in the final step of the EKF algorithm called the correction step.

$$\begin{bmatrix} y_t \end{bmatrix} = \begin{bmatrix} x \\ y \\ \psi_z \\ V_x \\ V_y \\ \omega_z \\ a_x \\ a_y \end{bmatrix} \tag{6}$$

3. Extended Kalman filter

As it was mentioned before, an EKF (Extended Kalman Filter) has been employed to preform the localization of this robot. Such algorithm relies on two steps sequence: **Prediction step** and the **Correction step**. The updates for the state and the covariances are performed in Eq. 1 and 7 respectively,

$$P_{t+1|t} = F P_{t|t} F^\top + Q \tag{7}$$

Where $\hat{x}_{t|k}$ is intended to be the estimate of x at time t given the control signal up to time k. Now, when the state space model is linearized to F the covariance update is possible. Further, the Kalman filter equations for the measurement update is seen in 8,

$$K_t = P_{t|t-1} H_t^\top (H_t P_{t|t-1} H_t^\top + Q_e)^{-1} \tag{8}$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t (y_t - h(\hat{x}_{t|t-1})) \tag{9}$$

$$P_{t|t} = (I - K_t H_t) P_{t|t-1} \tag{10}$$

3.1 Predict for EKF

The prediction for the EKF is preformed by equation 1 and 7.However, the prediction in this project differs from the previous in the following way, we have a more states in our state space model, where our control signal is updating the position states and the orientation states. The acceleration states are updated using the accelerometer readings. Thus, there are in total 5 sates as can be seen in equation 2. To predict the covariance matrix it is then required to linearizer the state space model with a Taylor series expansion, this creates a 5x5 matrix which is the Jacobian matrix, the Jacobian matrix has been used to predict the covariance of the system.

3.2 Correct for EKF

After the prediction is complete, the correction of the predicted state is facilitated from the measurements on the robot, the measurements are two IMU readings i.e. the accelerations along X axis and the Y axis, the position data from the HTC vive lighthouse deck and three encoder readings, by taking the difference in the innovation seen in equation 9, the Kalman gain calculated will help to weigh the different measurements form the sensors and the final estimated state is reached, this is

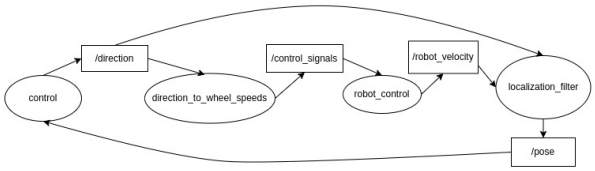

Figure 5. ROS graph

where the 'sensor fusion' is completed. The corrected estimate is then sent to the controller as input. for the next iteration the correction of the covariance matrix is preformed and is fed back to the predicted step along with the new control signal and the corrected estimated state, the cycle continues until the final desired position is achieved.

4. Implementation

4.1 ROS

The software for the robot to control and estimate its position is done using ROS. ROS that stands for Robot Operating system is an open-source robotics middle-ware. Although it is not an actual operating system but a collection of software frameworks for robot software development. It provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. The operating processes in ROS can be represented using a ROS graph where the operations happen in the nodes that may receive, send or multiplex sensor data and other messages such as control signals and set points. The other features that ROS provides is suite of debugging tools that enable us to plot data from the ROS topics in real time and check for inconsistencies. In this project ROS is used as framework for multiple python nodes that perform different tasks within the robot application. The ROS graph in the following figure 5 is the ROS graph that shows all the nodes and how they communicate with each other and the hardware using the respective ROS topics.[3]

4.2 Message passing

Since the body velocity obtained by the wheel velocities is one of the sensory data used in the EKF, we need to extract wheel speeds from the motors at every iteration. The ROS-node we initially used didn't accommodate for timing and this led to the robot loosing its control over the motors every time we read the sensor data. This was a result of the bus used to read and write to the motor being a shared variable between the reading and the writing functions. To fix this conflict, we introduced a ROS service which whenever invoked would obtain a lock over the bus to read the sensor data and release it whenever it is not using it thus removing the conflict between the 2 processes trying to access the bus.

4.3 Control signal

From the above section we can see that the control node in the ROS system, subscribes to the $/pose$ and the $/setpoint$ topics. It computes the errors along X, Y and ψ. It then runs the error through a PI controller and computes the corresponding correction body velocities that are fed to the robot as control signals in the form (V_x,V_y,ω_z). This body velocity is then converted to wheel velocities and then to motor PWMs in the direction_to_wheel_speeds, and the control_signals nodes respectively.

4.4 Kinematics

As we are using a 3-wheel omnidirectional setup, we need an inverse kinematic model to transform the linear and angular velocities of the body to wheel speeds. In our case, d is the distance of each wheel from the centre of the robot, r is the radius of the wheel and in a general case, angle α_i is the angle between the axes of the wheels. The angle θ is the angle of the first wheel from the X axis of the robot body frame. In our robot, the values of θ, α_1, α_2 and α_3 are 30 deg,0 deg, 120 deg, and 240 deg respectively. The corresponding transformation can be represented using the following matrix.

$$R_{IK} = \begin{pmatrix} -sin(\theta) & cos(\theta) & d \\ -sin(\theta+\alpha_2) & cos(\theta+\alpha_2) & d \\ -sin(\theta+\alpha_3) & cos(\theta+\alpha_3) & d \end{pmatrix} \tag{11}$$

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} = R_{IK} \cdot \begin{pmatrix} V_x \\ V_y \\ \omega_z \end{pmatrix} \tag{12}$$

This matrix was used in the direction_to_wheel_speeds node to convert the robot velocity from the control node to wheel velocities that is further used to move the robot in the desired way.

Additionally, a rotation matrix (Eq. 13) is applied to the motion body frame to fully describe the velocities V_x, V_y and angular velocity ω_z from the wheel velocities V_1, V_2 and V_3 (Eq. 14). Due to the symmetric nature of the robot, the forward kinematics can be computed by directly inverting the Inverse Kinematic transformation from (Eq. 11).

$$R_{FK} = R_{IK}^{-1} \tag{13}$$

$$\begin{pmatrix} V_x \\ V_y \\ \omega_z \end{pmatrix} = R_{FK} \cdot \begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} \tag{14}$$

This matrix was used to compute the body velocities from the wheel velocities in the robot_control node which is further used in the localization_filter.

4.5 Calibration of encoder measurements

The measurements from the encoders had to be calibrated for the controller, as the EKF algorithm is running at 10000 Hz the encoder measurements had to be scaled to compensate for the frequent update of the EKF. Thus, at first the encoder values where too small in the algorithm, the controller would run the control commands more frequently than necessary, the encoder measurement where multiplied by a thousand and a destination of 10cm was given to the robot, the robot would reach around 3.1cm which was then compensated for by dividing the 1000 by 3.19 to reach the desired scaling factor which would reach the 10cm mark with a ±1mm error.

4.6 Tilt Compensation

Due to the unevenness of the ground, the robot would have a tilt error form the IMU reading, that would result in the robot control to incorrectly compensate for this tilt. This is implemented to remove any effect of the acceleration due to gravity appearing on the other axes namely x and y. In an ideal scenario, when the robot is stationary on a completely flat surface, the acceleration across z should be -g and 0 across x and y axes. In order to achieve tilt compensation, we acquire the accelerometer data along x,y and z. We then compute the orientations of the IMU in the X-Z plane and the Y-Z plane. Upon obtaining these angles, we know that the accelerometer orientations in the space. We then project the raw values onto the corrected axes. This is well described in the following equation 15-18.

$$\phi_x = tan^{-1}(acc_x^{raw}/acc_z^{raw}) \tag{15}$$

$$\phi_y = tan^{-1}(acc_y^{raw}/acc_z^{raw}) \tag{16}$$

$$acc_x^{compensated} = acc_x^{raw} \cdot cos(\phi_x) \tag{17}$$

$$acc_y^{compensated} = acc_y^{raw} \cdot cos(\phi_y) \tag{18}$$

4.7 Path planning

An algorithm was implemented to automate the motion of the robot in a continuous loop of points chosen. This algorithm keeps track of the error i.e. the euclidean distance between the current position and the destination. When the error is smaller than the resolution of the robot, the next destination is automatically selected and the robot starts moving to the next position. The path planned is a square with the sides of length 10 cm.

4.8 Extended Kalman filter

The pseudo algorithm seen in algorithm 1 represents the EKF algorithm.

Algorithm 1: Extended Kalman Filter Algorithm

```
while currentSimulationTime < simulationTime do
    Calculate the state estimate with the control
      commands from the previous time step;
    Calculate the error covariance with the Jacobian
      of the model;
    Calculate the measurement residual;
    Calculate the Kalman gain;
    Correct the state estimate with the Kalman gain;
    Correct the error covariance;
end
```

As in the previous project the group relied on the help of [5, pp. 203–220]. for the completion of the EKF algorithm, as well as the experience the group members gained from the previous projects.

Predict algorithm Equation 21 shows the Jacobian matrix of the state space, the Jacobian is required to predict the covariance matrix.

$$a = -dt * ((u_1) * sin(\psi_z) + (u_2) * cos(\psi_z)) \tag{19}$$

$$b = dt * ((u_1) * cos(\psi_z) - (u_2) * sin(\psi_z)) \tag{20}$$

$$F = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \tag{21}$$

Correct algorithm As our observation model needs to be weighed for the correction step its Jacobian is shown in equation 22, as previously mentioned the output of the update step is again fed into the Predict State and the cycle goes on until the final position is estimated with in a satisfactory frame.

$$H_t = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ dt & 0 & 0 \\ 0 & dt & 0 \\ 0 & 0 & dt \\ 0.5 * (dt * dt) & 0 & 0 \\ 0 & 0.5 * (dt * dt) & 0 \end{pmatrix} \tag{22}$$

5. Results

The EKF algorithm with sensor fusion from the three encoders is finalized and have the same units as the control signal values, the IMU units are adjusted to the control signals units as well as the data from the HTC VIVE light house. As well as the re-tuning of the PI controller to accommodate the new control signal units. The ROS network and the inverse kinematics have been adjusted to the project, debugged and are running as expected. The MSE error for the localization was computed and the values stayed between $6.53 * 10^{-8}$ to $7.4 * 10^{-8}$ along X and $5.23 * 10^{-8}$ to $5.59 * 10^{-8}$ along Y when run for 15 minutes in a square trajectory.

5.1 Repeatability and Reliability

From the path planning node the robot had an automated loop that moved in a square with sides of length 10cm, at every corner of the square a point is made, this test is done to visually see the error distribution as the robot moves along the square in an infinite loop, the position estimate of the EKF as well as the measured light house measurements are plot and can be seen in figure 6 where the units for the x and y axis are in meters, the EKF position estimate is in red and the light house position measurement is in blue. Further, the light house measurement was blocked to test the effect of the light house measurement, the results of that test can be seen in figure 7, the light house measurement is the dominant measurement in the sensor fusion done by the EKF, the light house was blocked in the right top corner of the square the robot then moved around 17cm in the x direction 7.4cm in the y direction, the EKF flowed the Light house. In figure 8 the markings on the floor have a error radius distribution of maximum of 3.5mm

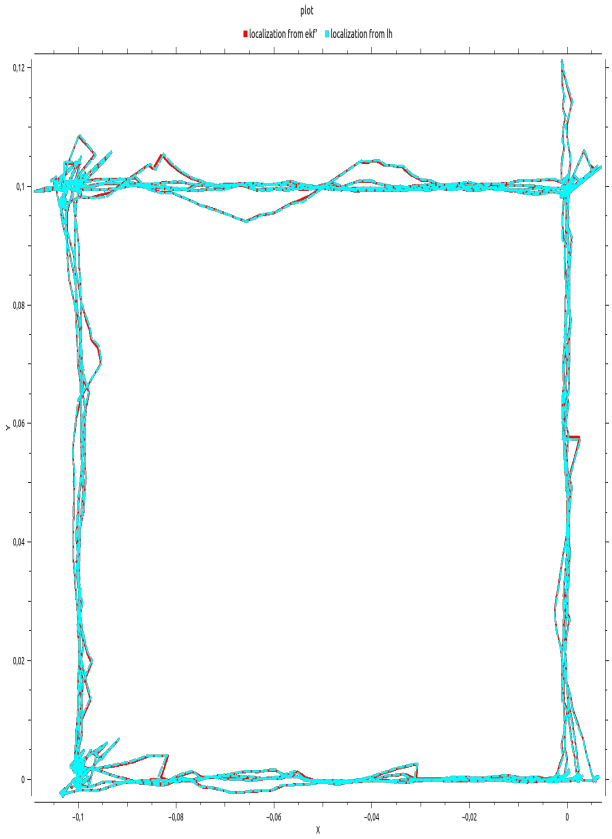

Figure 6. EKF estimate VS Light house measurement

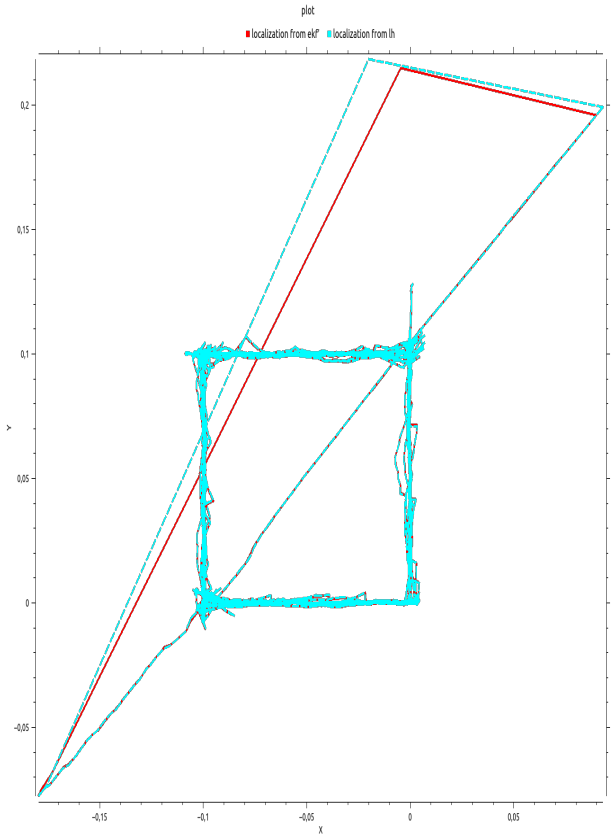

Figure 7. EKF estimate VS Light house measurement blocked

this is measured after the robot has moved clockwise and the over shoots are mostly seen on the y axis, the robot is then moved anti clock wise and a different corner have the largest error radius to be 3mm as seen in figure 9. In figure 10 and 11 where the robot reaches the 10cm destination with an error radius of around 2.5mm for both the x axis and the y axis respectively.

6. Discussion

6.1 Results

The testing on the 8th of December at MAX IV, resulted in acknowledging that more work needed to be done on the network for the message passing to the motors. It was also observed that the environment was dusty where the robot would be working, thus error would arise from the encoder measurements. Further the leica laser could not be relied on for measurement values as jitters where present in the leica laser measurements, the jitters are due to the fact that the leica laser is not capable to update the measurements of a moving target as well as it does for a stationary one, thus different covariances was tuned in the EKF for when the robot is moving and when it is stationary. As the robot should trust the IMU measurements more when the robot is moving and trust the encoders as well as the leica laser more when stationary, the covariance matrix must be tuned accordingly. As every sensor has disadvantages and advantages when in operation the sensor fusion is not only beneficial for the operation of the system but necessary. The testing of this EKF design could not be done in due time,

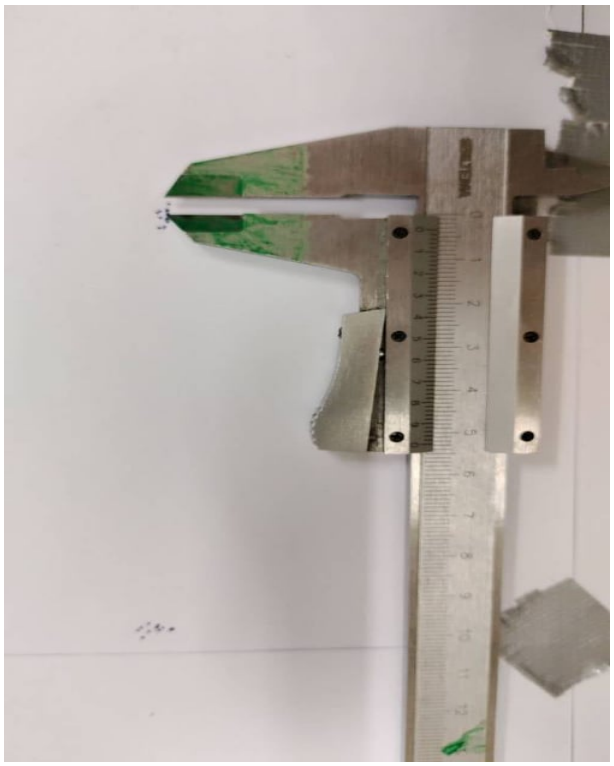

Figure 8. 3.5mm radius error distribution when running the robot clockwise

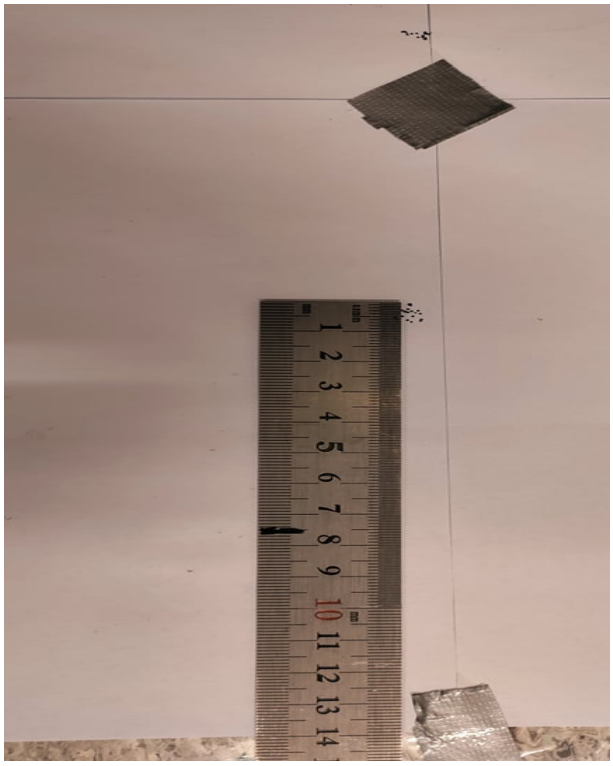

Figure 9. 3mm radius error distribution when running the robot clockwise and anti clock wise

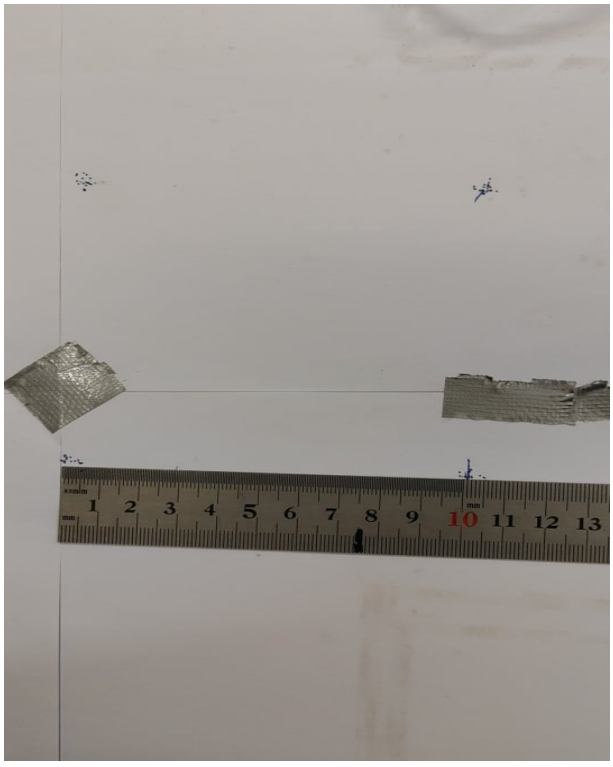

Figure 10. horizontal total distance and error when running clock wise and anti clockwise

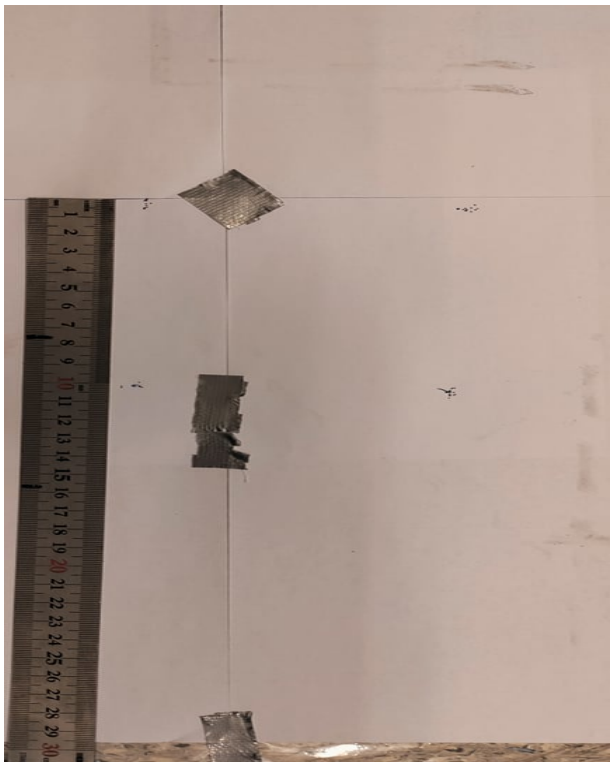

Figure 11. Vertical total distance and error when running clock wise and anti clockwise

as the measurement engineer Alina did not have a possibility to meet with us during the holidays. Thus, the team replaced the Leica measurements with the Light House measurement, the light house measurements are less accurate with 4cm but is more reliable when it comes to moving target, as seen in 5.1 the resolution of the position estimate relative to the resolution of the Light house measurement resulted in a 10 folds improvement form 4cm to around 3mm. The final set up of the algorithm and results were presented on the 14th of January to the faculty members and students at LTH.

6.2 Project dynamics

The dynamics of the group has been under pressure, as one of our group members decided to drop the course. Further, the testing of the robot was scheduled to be on the 8th of December, thus the group was required to be 4 weeks ahead of schedule, this resulted in long night (the longest night lasted until 5 in the morning) in the lab to get everything debugged and working. Results of the work were investigated on the 8th and reported soon after. Finally, the group missed one week of supervisor help due to the robot lab week, which resulted in the investigation of quaternion which was not necessary to the project, the python package built by Anders Blomdell that is responsible for controlling Dynamixel motors did not account for negative values in the register. Thus, some hours were spent to understand and process the negative values from the Dynamixel motors. Due to the circumstances and the nature of a project was proposed by the authors. Resulted in the authors having to be more self reliant, compared to other projects.

6.3 Project outcome

The project was a challenging endeavour for the time assigned to it, many over time hours were invested into the project for the final results, the majority of the time was investigating and debugging the code implementation for better results. As the project has not been done before many unknown unknowns where realised as the project developed, this project contains information form the majority of the courses taken previously by the authors and some courses that haven't been taken at all. The investigation in this project will further develop in the authors thesis work, that will be worked on in the spring.

References

[1] *DynamixelSDK*. https://github.com/ROBOTIS-GIT/DynamixelSDK. Accessed: 2021-12-1.

[2] V. V. Patil and L. D. Carrera. *OmniWheel Robot*. The Faculty of Engineering at Lund University, 2021-06-01.

[3] *ROS*. https://ros.org. Accessed: 2021-12-1.

[4] A. Shahin. *Tracking of a high precision robot*. The Faculty of Engineering at Lund University, 2021-06-01.

[5] S. Thrun. *Probabilistic robotics*. Vol. 45. 3. ACM New York, NY, USA, 2002.