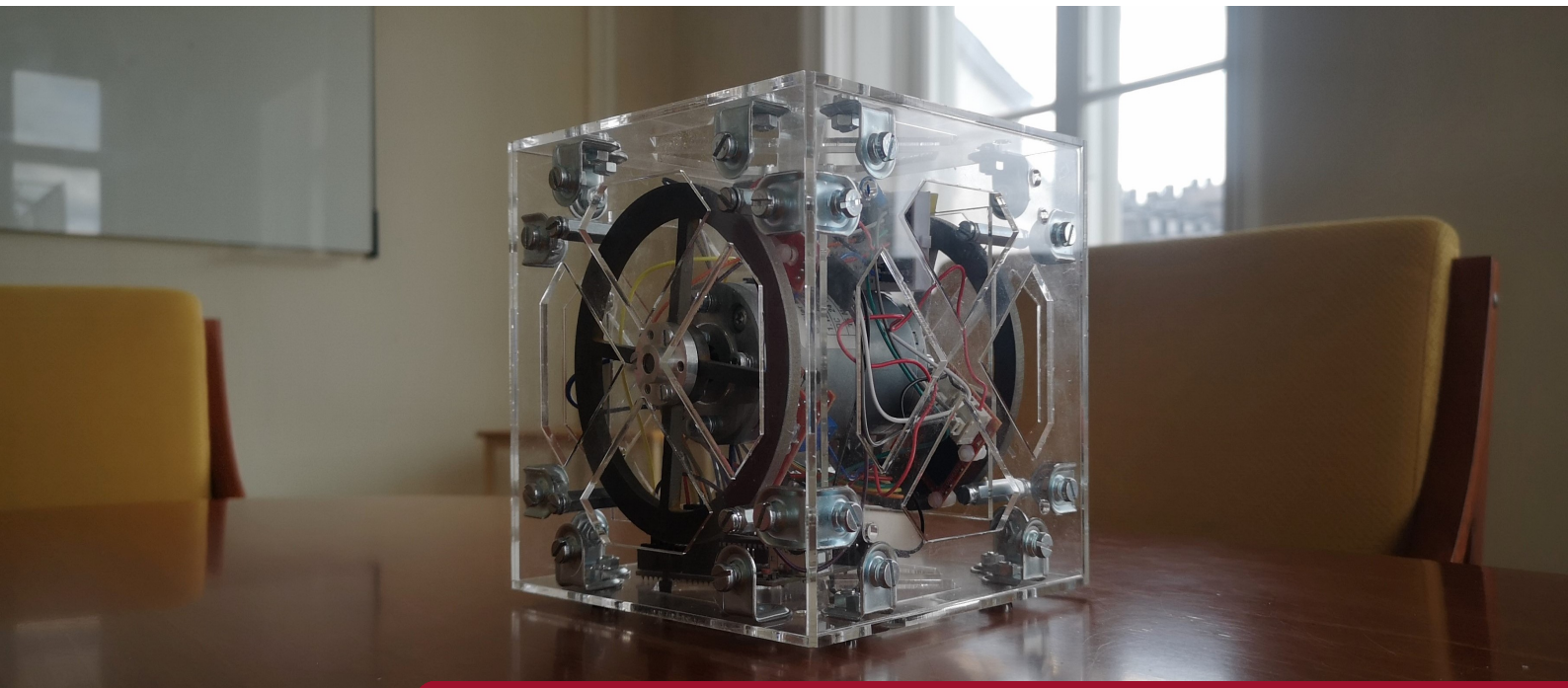
DEGREE PROJECT IN ,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, *SWEDEN 2020*

PID Regulated Balancing Cube
PID-reglerad Balanserande Kub

SEBASTIAN BRANDMAIER

DENIS RAMSDEN



KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

PID Regulated Balancing Cube

Bachelor's Thesis in Mechatronics

SEBASTIAN BRANDMAIER, DENIS RAMSDEN

Bachelor's Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2020:33

Abstract

In this project a cube was constructed with the intent for it to balance on one of its edges by regulating the speed of a reaction wheel. To be able to do this, a research was done to understand the mechanics of the system and to know what components were required for the project. A brushed DC motor was used with one reaction wheel on each end of its shaft, using the moment of inertia to convert the torque from the motor to an angular acceleration of the cube. A control system was implemented to regulate the speed of the motor depending on the angular offset of the cube. This control system was chosen to be a proportional–integral–derivative (PID) controller, and a number of different tuning methods were used to determine the parameters of said controller to create a stable system. Despite the different methods used, the cube did not successfully balance for a longer period of time.

Keywords: Mechatronics, PID-control, Reaction wheel, Arduino, Cube.

Referat

PID-reglerad Balanserande Kub

Under detta projekt har en kub konstruerats med syfte att balansera på en av sina kanter genom att reglera hastigheten på ett reaktionshjul. En undersökning genomfördes för att få en bra förståelse för det mekaniska systemet samt för att ta reda på vilka komponenter som var väsentliga för projektet. En borstad likströmsmotor användes tillsammans med ett reaktionshjul på varsin sida av motorns drivande axel vars tröghetsmoment utnyttjades för att överföra momentet från motorn till en vinkelacceleration på kuben. En regulator implementerades för att styra hastigheten på reaktionshjulen beroende på vinkelavvikelsen från jämviktsläget. Regulatorn som valdes var en proportionell, integrerande och deriverande (PID) regulator och flera olika metoder användes för att bestämma regulatorns parametervärden som avgör dess egenskaper. Trots att olika metoder prövades lyckades inte kuben balansera under en längre period.

Nyckelord: Mekatronik, PID-kontroll, Reaktionshjul, Arduino, Kub.

Acknowledgements

We would like to thank our supervisor Nihad Subasic that have given us continuous feedback during our work and Seshagopalan Thorapalli Muralidharan that have help us to manufacture all the necessary parts and guided us through this project.

Sebastian Brandmaier, Denis Ramsden
Stockholm, May 2020

List of Abbreviations

PID - proportional–integral–derivative
DC - direct current
CAD - computer aided design
IMU - inertial measurment unit
MCU - microcontroller unit
PWM - pulse width modulation
GPIO - general-purpose input/output

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Control systems are used in almost any kind of engineering industry due to the ability to automate regulation of dynamical systems. It could be from adjusting the flow of water through a pipe to control satellites circulating the Earth.

The project that inspired this thesis the most is the *Cubli* [15], a cube that uses a control system to balance on a corner with help of reaction wheels. It balances by accelerating the reaction wheels with a motor creating torque with the moment of inertia making the cube counteract its own weight. It does not only demand a good control system, but also a lightweight design that can contain a lot of electrical components.

The aim of this project was to construct a similar product, a cube that balances on one of its edges.

The theory of constructing a balancing cube could benefit similar projects in the future. Examples where control theory has been applied is when constructing a *Ball balancing robot* [23], a *Riderless self-balancing bicycle* [19] or when *Implementing Gyroscopic Stabilization* [14]. These are other theses that have used different kind of control systems and are great examples of what control systems are able to accomplish.

1.2 Purpose

The fact that the design of the physical product is very simple, but in reality is based on a complex control theory problem was the fundamental idea for the project. Although the final product will not solve any everyday problems, the technology is important as it provides insight into how control theory can be applied in practice. The experience of controlling motors and handling signals can benefit

later projects since similar problems exists in many areas. This study will mainly focus on answering following questions:

- What is a suitable way to construct a cube with the necessary hardware to make it balance on an edge using a reaction wheel?
- Is it possible with the resources available to balance the cube with a PID controller and what tuning method should be used?

1.3 Scope

The focus of this project is to design and construct a fully functional prototype of a product. This with the limitation of a budget of 1000 SEK and a project period stretching over five months. There are different ways to design a control system, but in this project a PID controller will be analyzed.

1.4 Method

The project started with the cube being constructed and assembled with all necessary components. Software was later written to communicate with the electric components including motor and sensors with an Arduino Uno as a development platform. All manufacturing of necessary components took place at the KTH Prototype Center. Mathematical models of the system was then made and analyzed to get better understanding of the system that helped implementing the PID controller. After that the parameters for the PID controller was tuned to create a stable system and several experiments were made on the cube to answer the thesis's questions.

Chapter 2

Theory

This chapter established all the gathered theory that was needed to get a better understanding of the mathematical systems and the functionality of the components in this project. The cube was designed according to the limitations and possibilities of the theory and helped answering the researched questions of this thesis.

2.1 System Dynamics

To be able to stabilize the cube on its edge a relationship between the different affecting factors were needed and a model of the system dynamics was set up. M_{max} is the maximum torque needed to counteract the construction's own weight at a horizontal position and is given by equation 2.1. From Figure 2.1 a mechanical equilibrium gives a relationship between the cube's mass m, gravitation g and the horizontal distance to its center of gravity from one of its corners $\frac{l}{2}$ according to

$$M_{max} = mg\frac{l}{2}. \tag{2.1}$$

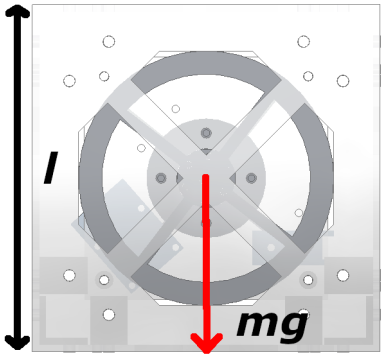


Figure 2.1. Gravitational force acting on the cube in horizontal position. Made in Solid Edge.

This sets the requirement for needed torque from the motor. An important aspect is to make the construction mechanically balanced because it will set a fundamental stability to the construction that gives a good starting point for the automatic control later. This is done by positioning the center of mass in the middle of the cube.

2.1.1 Reaction Wheel

The most suitable shape of a reaction wheel is a ring because it provides the greatest moment of inertia per mass that will result in a decreased acceleration for the same torque. The theoretical moment of inertia for a ring I_{ring} can be calculated as

$$I_{ring} = mr^2 \tag{2.2}$$

where m is the ring's mass and r is its radius. [6] When the motor accelerates the reaction wheel generates torque. Due to Newtons third law, a reaction torque that acts on the cube in the opposite direction is also generated. This torque can be described as

$$M_{motor} = I_{ring}\dot{\omega}_{motor}.[6] \tag{2.3}$$

where I_{ring} is the wheels theoretical moment of inertia and $\dot{\omega}_{motor}$ is the angular acceleration of the reaction wheel. [6] By adjusting the speed and acceleration of the motor the angle between the cube and the surface can be controlled. In this project a Newtonian model was used that can be described as

$$I_{tot}\ddot{\theta} = mglsin(\theta) - M_{motor} \tag{2.4}$$

where θ is the angular offset perpendicular to the surface below.

A requirement for the wheel is to be able to store enough energy that it takes for the cube to tilt to an angle of 45 °. The kinetic energy T that the wheel can store is [5]

$$T = \frac{l}{2} I_{ring}\omega_{max}^2 \tag{2.5}$$

where ω_{max} is the maximum speed and I_{ring} is the moment of inertia that is described in equation 2.2. The energy V it needs to store is the difference in potential energy V between the states when it is lying horizontal on the surface and balancing at a 45 ° angle. This is mathematically described as [5]

$$V = \sqrt{2} m_{cube} g \frac{l}{2} \tag{2.6}$$

where m_{cube} is the mass of the cube, g is gravitational acceleration and l is the length of the cube. By using equation 2.2 and setting V equal to T the wheels minimum mass can be calculated to

$$m_{min,ring} = \frac{\sqrt{2} m_{cube} g}{\omega_{max}^2 r_{ring}^2}. \tag{2.7}$$

2.1.2 PID Controller

A PID controller is the most commonly used controller in the industry. The controller uses a reference point that is the desired value of the system and compares it with its current value and the difference is the error. It then uses the error to stabilize the system with three components. PID means that the controller consists of three components: a proportional, an integrative and a derivative component. The P-parameter K_P is used to adjust the gain, the I-parameter K_I is used to reduce the static error, and the D-parameter K_D compensates for changes in the system that increases the stability. [16]

Since Arduino includes a built in PID function within its libraries it will be used in this project. The PID function has three parameters K_P, K_I, and K_D that will be chosen experimentally. [3] The ideal PID regulator can be written as:

$$u(t) = K_P e(t) + K_I \int_{t_0}^{t} e(\tau)d\tau + K_D \frac{de(t)}{dt}. \tag{2.8}$$

There is different ways a PID controller can be tuned to stabilize a system. One method is the Ziegler-Nichols method where analysis of the system could give a good approximation of the three components creating the PID controller. Given a system the simplest method is to increase the gain for the system until a steady state oscillation occurs, giving the K_{cr}. This with the given oscillation period P_{cr} can with this method give a good starting point for tuning the PID. The method suggests the following values in relation with the analysed valued above: [20]

$$K_P = 0.6K_{cr}, \tag{2.9}$$

$$T_I = 0.5P_{cr}, \tag{2.10}$$

$$T_D = 0.125P_{cr}. \tag{2.11}$$

According to the method with the new parameters above, the equation 2.5 of the PID-controller can now be written as

$$u(t) = K_P(e(t) + \frac{1}{T_I} \int_{t_0}^{t} e(\tau)d\tau + T_D \frac{de(t)}{dt}). \tag{2.12}$$

2.2 Components

To be able to construct a control system the change of the position of the cube was needed to be detected and then to be compensated for. Components made it possible for the cube to physically accomplish this and was an important part in the construction of the cube. The electric schematic for the components can be seen in the Appendix A.

2.2.1 Motors

In this project a brushed DC motor was used. A brushed motor means that it is internally commutated. Because direct current is used ohm's law and Kirchhoff's voltage law can be used in calculations. The relationship between voltage U_A, current I_A and velocity ω can therefore be described by [21]

$$U_A = R_A I_A + K_2 \Phi \omega. \tag{2.13}$$

where R_A is the terminal resistance received from the datasheet that can be found in Appendix B. $K_2\Phi$ is a constant that can be calculated by measuring the speed of the rotor shaft when there is no load on the motor. This gives that the voltage is direct proportional to the rotating speed

$$U_A = K_2 \Phi \omega \tag{2.14}$$

where ω is the angular velocity.[21] $K_2\Phi$ can then be measured, but was in this project also found in Appendix B. The torque M that the motor generates is proportional to the current I_A and is mathematically described as

$$M = K_2 \Phi I_A.[21] \tag{2.15}$$

Combining equation 2.13 and 2.15 gives that the total motor torque M is

$$M = \frac{K_2 \Phi}{R_A} U - \frac{K_2 \Phi^2 \omega}{R_A}. \tag{2.16}$$

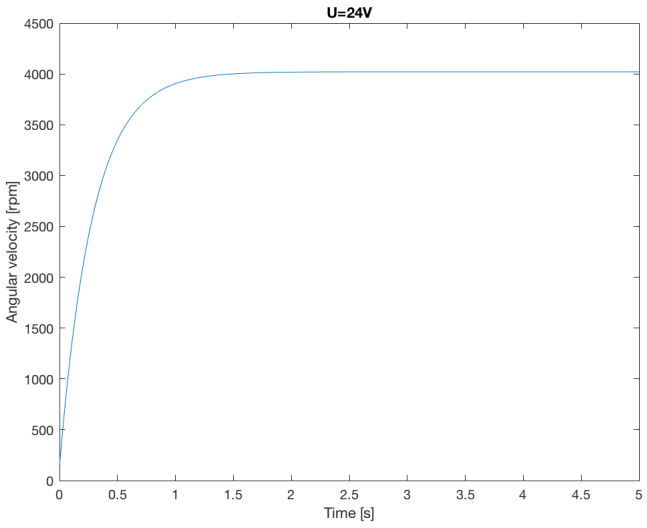


Figure 2.2. Angular velocity over time when a voltage of 24V is applied on the motor. Made in Matlab.

The equation 2.3 and 2.16 was used to plot how the angular velocity of the reaction wheel changes over time when the voltage 24V is applied on the motor. This is shown in Figure 2.2. The angular velocity was calculated with the Euler forward method and depends on the torque that was received from equation 2.16. In the datasheet the no load speed for the motor is 3900 rpm. The reason why the motor passes this speed despite there is load on the motor might be because the used model does not consider internal friction of the motor. A theoretical value of the load was calculated by approximating the reaction wheel as thin ring. The moment of inertia could then be calculated with equation 2.2.

2.2.2 Hall Effect Sensor

To be able to control the angular velocity of the reaction wheels hall effect sensors were used. The sensors can measure the strength of a magnetic field due to the disturbance of electrons and sends a signal to the micro controller when it detects a magnet. If a magnet is placed on the reaction wheel the hall effect sensors will sense a disturbance for every spin of the wheel. This information combined with the time difference can be used to calculate the angular velocity.

2.2.3 IMU

To determine how the cube was positioned relative to the room an IMU was used. The IMU's origo was set to the position the cube is supposed to balance on. When the cube is falling in any direction the IMU will transmit a signal to the micro controller of this change in angle. The micro controller was programmed to stabilize the cube and bring its position back to its setpoint.

2.2.4 Arduino

For this project an Arduino UNO was used as a MCU. Arduino is an open source development platform that makes it easy to control motors and sensors. This was done using the built in GPIO pins. Arduino UNO uses an ATmega328P processor which is programmable in the language C++. It has many of built in libraries that makes it easy to control sensors and motors. Arduino UNO is an 8 bit MCU and uses a reference voltage of 5V. Therefore when it sends analog signals it converts an int between 0-255 to a PWM signal between 0-5V. [1],[4]

2.2.5 H-bridge

To drive a DC-motor with higher voltage it was necessary to use an H-bridge. This was due to the Arduino's instability to output a voltage over 5 V. An H-bridge is used to control an external power source with small PWM-signals described in section 2.2.3. Conventionally an H-bridge is made of four transistors as shown see in Figure 2.3, that not only makes it possible to change the output voltage, it can also change the direction of the current. Another reason why it was important to

use an H-bridge was because DC motors also can work as generators. Therefore if some external force would make the motor start spinning it would induce a current which could destroy the Arduino. [21]

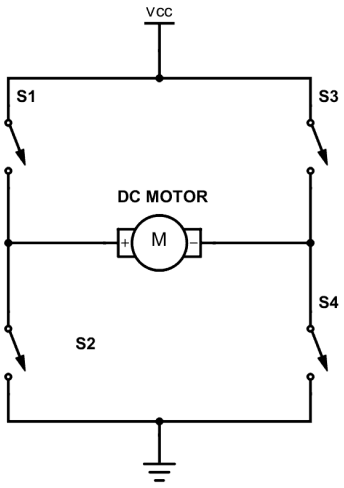


Figure 2.3. A schematic image of a H-bridge.[9]

Chapter 3

Design

In this chapter the visual design of the cube as well as the design for the code and the PID controller was established. The design was created according to the requirements and limitations of the project.

3.1 Shell

The cube was designed to have six sides acting as a protective shell with all of its necessary components within and its edges to become good balancing points. The sides were laser cut from acrylic glass with an thickness of three millimeters. Acrylic glass was chosen because of its low density and stiffness.

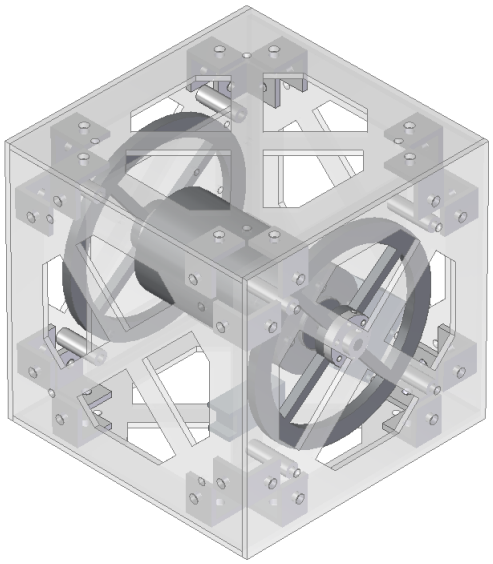


Figure 3.1. A CAD model that shows the cube shell with reaction wheels and motor installed. Made in Solid Edge.

In every corner three corner brackets were used to connect all plates to each other using M5 screws and nuts, that is visible in Figure 3.1. Then to connect the motor to the cube, two other plates were connected to each side of the motor with M4 screws. These plates were then connected to the cube with distance screws in each corner, see Figure 3.2. The two reaction wheels were placed on the motor axle on opposite sides on the motor, between the side of the cube and the motor holders, see Figure 3.2.

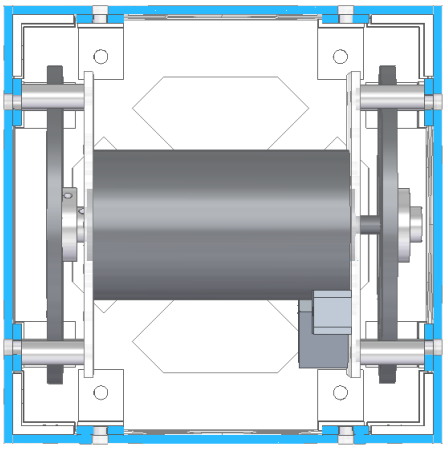


Figure 3.2. A section view of a CAD model of the cube seen from the side, showing the motor holders. Made in Solid Edge.

3.2 IMU

The MPU-6050 was used as an IMU in this project and can be seen in Figure 3.3. It gives values for the change of angle on the z-axis that was used as input for the PID controller to control the system.

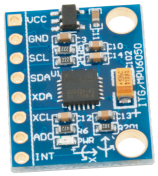


Figure 3.3. MPU-6050 - Gyroscope/Accelerometer. [13]

3.3 DC Motor and H-bridge

The motor chosen for this project was a Buhler 24 V DC motor because of its torque output. [8] To control the voltage over the motor an H-bridge was used and the choice of the specific H-bridge depended on the chosen motor. It had to be able to output a current as high as 12 A and control a voltage up to 24 V. The VNH3SP30 motor driver, see Figure 3.4, did meet the requirements and was chosen for this project. It can output a current up to 30 A and voltages up to 40 V. [24], [8]

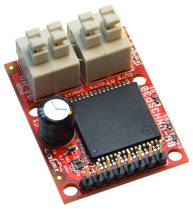


Figure 3.4. Motor driver VNH3SP30. [12]

3.4 Reaction Wheel

The wheel's purpose was to generate as much moment of inertia per mass, therefore the shape of a ring. Two wheels with a radius of 5,5 cm and the weight of 150 g were created to generate it. A model of the reaction wheel can be seen in Figure 3.5.

The reaction wheel is made of steel and was water cut because of its simplicity with the wheel only needed to be cut in one dimension. The wheel is then mounted on the motor axle with a hub that is connected directly on the reaction wheel with six M3 screws and then locked on the axle with two set screws from the hub.

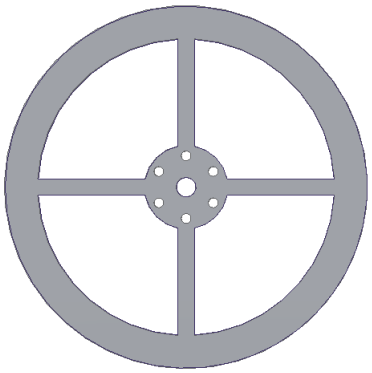


Figure 3.5. A CAD model of the reaction wheel seen from the side. Made in Solid Edge.

The 24V motor has a rated speed at 3000 rpm and with that as ω_{max} equation 2.7 gives that the wheel must weight at least 93 g. Since the wheel is not an ideal ring as assumed in equation 2.2 a mass of 150 g gives a big margin.

3.5 Measure of Motor Speed and Direction

To be able to control the DC motor it was necessary to measure the speed and direction of the motor. Since there was no rotary encoder available a method described in the thesis *Reaction Wheel Stabilized Stick* [18] was used instead. This method uses two hall effect sensors and two magnets, but in this project the number of magnets was increased to four to reduce vibrations that occur if the reaction wheel is unbalanced. The magnets was mounted on one of the reaction wheels and the sensors on the motor holder giving a good distance between them. The mounting of the magnets is shown in Figure 3.6. The sensors sends a signal every time a magnet is close to it and with a function in the Arduino library that measure past time, it gives us the angular velocity. Due to having two hall effect sensors it is possible to determine the direction of the reaction wheel, this is because one of the sensors will send a signal right before the other.

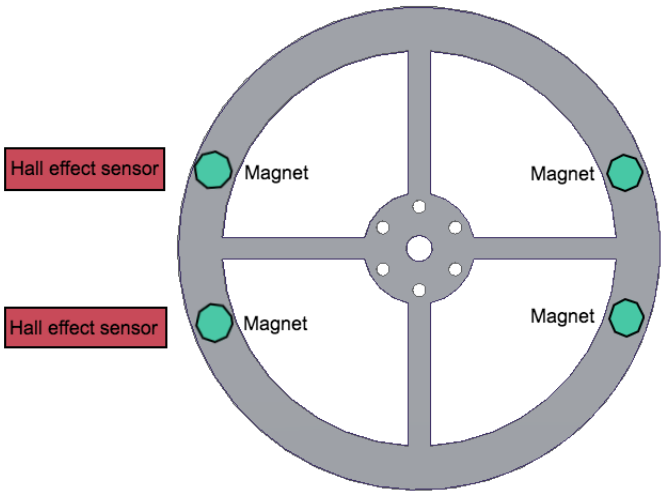


Figure 3.6. Sensors for measuring wheel speed. Made in Pixlr Editor. [17]

To measure the wheel speed a built in function called *micros()* was used that returns the time since the Arduino was powered on. [2] If both hall effect sensors detects a

magnet the wheel speed ω can be calculated by

$$\omega = \frac{\pi}{\Delta t} \tag{3.1}$$

where Δt is the time difference between the current and the previous measure. To determine which direction the wheel is rotating a variable called *state* was defined. Before both sensors detects a magnet at the same time, one of the magnets will have been detected by one of the hall effect sensor. This information is stored in the state variable and is used to determine the wheel rotating direction.

3.6 PID Controller

Arduino have a PID library that was used to control the system. [7] The PID function needed the three parameters K_P, K_I and K_D, as well as a setpoint value and an input value to create a stable system. The setpoint value is set to the 45° that is the wanted angle for the cube to balance at and the input it the current angle of the cube. Three different methods were used to determine the three PID parameters. The first one is the Ziegler-Nichols method which make it possible to determine the parameters with the analysis of the system when the gain is increased, see section 2.1.2. Method two is to alter each parameter to affect the system in a certain way. With the K_P increased to make the system respond faster, the K_I increased to make the system converge closer to the setpoint and the K_D to decrease the oscillation of the system. The third method is similar to the second, but take the angular velocity into account when calculating the output of the PID controller, given its effect on the torque in equation 2.16.

3.7 Code

The code was written in the Arduino IDE and can be read in Appendix C. At the beginning of the document all variables was defined. In the setup function the GPIO pins were initialize as inputs or outputs. To make it easy to use the IMU library [22] was downloaded and included in the file. With this library it comes an example code that was rewritten to suit the rest of the program. Also a PID library[7] was downloaded and included.

The PID function uses an input signal to calculate an output signal depending on the parameter K_p, K_i and K_d. Two methods were used to determine the input to the PID controller. One was to use the angle that is received from the IMU as an input signal. The other one was to include the value of the angular velocity due to its effect on the torque according to equation 2.16 and use that as input. The output signal is used as a PWM signal and regulates the speed of the motor. The PID values was calculated in the loop function and is therefore updated repetitively when the program is running.

To calculate the angular velocity the analog output was used on the hall effect sensors to detect the magnets if the magnet field was greater than a set value. This method worked better than using the digital output. Since the void function is running repetitively a Boolean variable was used to prevent that the code does not measure the velocity many times when a magnet is detected. If the Boolean is 1 when the sensor detects a magnet it measures the velocity, after it is set to 0. When the sensor doesn't detect a magnet anymore the Boolean is set to 1 again.

In Figure 3.7 a flowchart of the balancing cube describes the structure of the written code for the Arduino.

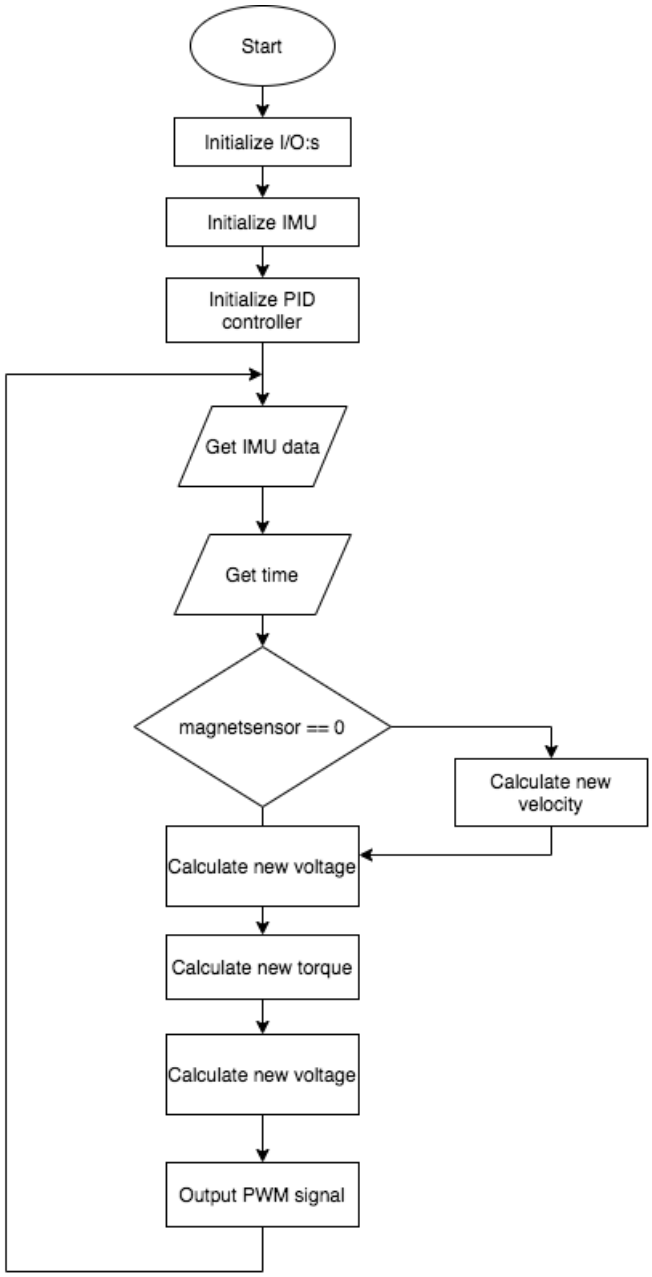


Figure 3.7. Flowchart for the balancing cube. Made in draw.io. [10]

Chapter 4

Results

The final product can be seen in Figure 4.1. It successfully contained all of its components except for a power supply which could be placed at the cube's side giving power to the cube. The design became very stable with its mass centre positioned very close to the centre of the cube which created a good fundamental mechanical stability. With this design the cube became very space efficient with almost no room for any other component within which led to a lighter shell.

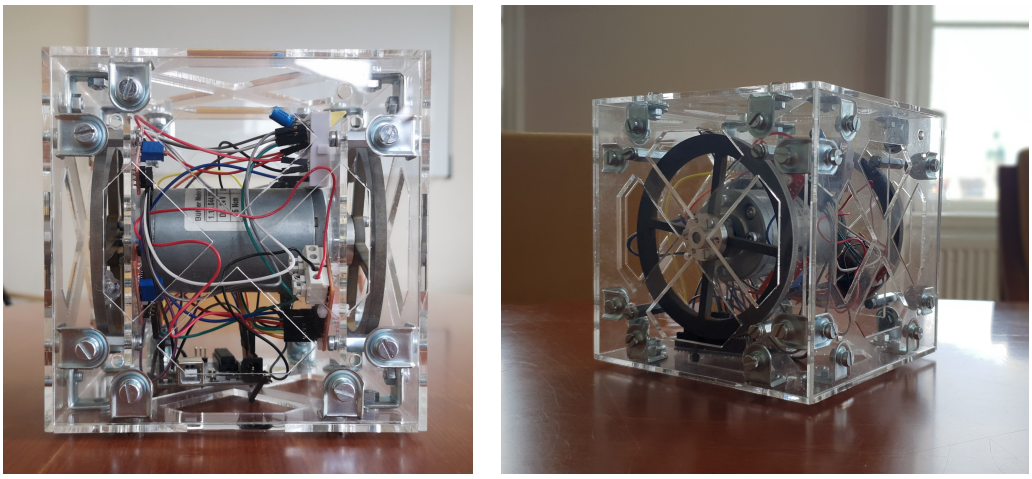


Figure 4.1. The final product seen from two different views.

4.1 IMU

At the beginning of the development process the MPU-6050 that was used as an IMU did not seem to work correctly. It took many seconds for the IMU to stabilize and it did not always stabilize at the same values. This problem was solved with a capacitor. The MPU-6050 is sensitive to voltage variations and the voltage supply from the Arduino is not always reliable. The capacitor compensated for this and the result is shown in Figure 4.2. A similar problem also occurred when the gain was increased in the PID controller. By adding a capacitor over the power supply it helped sustain a steady voltage even when the gain was increased. The Matlab code that was used for the plot can be found in Appendix C.1.

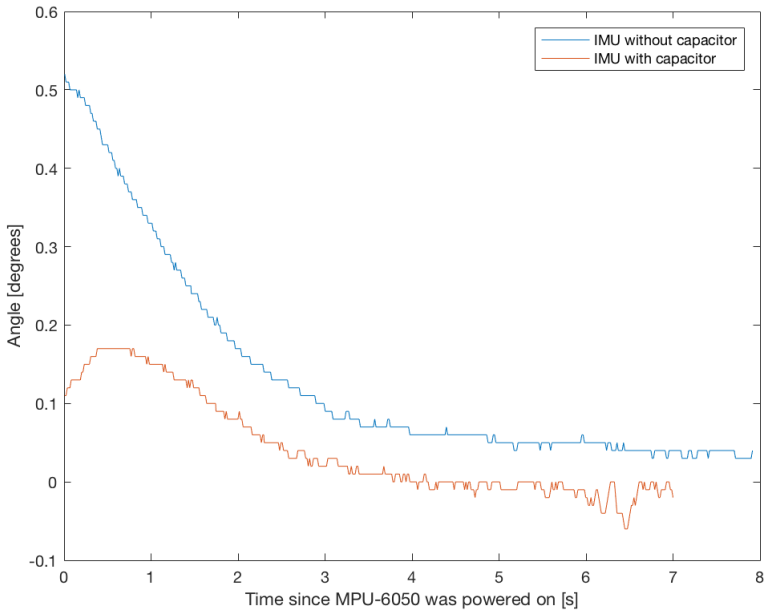


Figure 4.2. Measured values from the IMU over time with and without a capacitor. Made in Matlab.

4.2 Motor

An experiment was done to determine the deviation between the mathematical model used in Figure 2.2 and the real dynamics of the motor. All constant values for the mathematical model was received from the datasheet for the motor that can be found in Appendix B. The used motor was a Buhler 24V with terminal resistance $R_a = 2\Omega$ and the torque constant $k_2\phi = 5.7Ncm$. [8] In Figure 4.3 it is shown that there is a big difference between the theoretical values and the measured values. The Matlab code that was used can be found in Appendix C.2

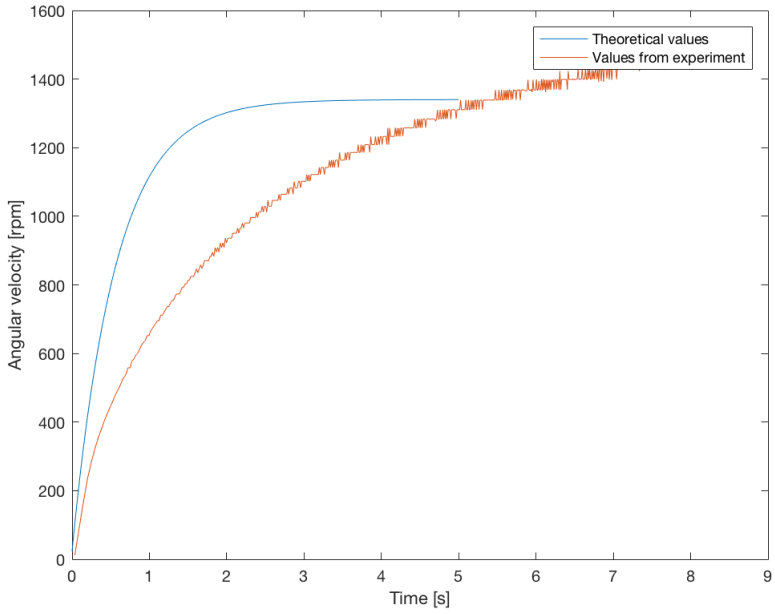


Figure 4.3. Comparison of theoretical rotating speed from equation 2.12 and measured values. Made in Matlab.

4.3 PID controller

The PID components were determined by various experiments made on the cube. The most successful method where to adjust one of the parameters at a time. The K_P component was increased until the value made the cube counteract its own weight when falling to one side. Though the controller was able to counteract the cube falling it couldn't counteract the response from the controller and the cube fell to the other side, the system was unstable. To compensate for this response a K_D component was added that will counteract the response based on the speed of the error change. When the cube oscillates with only the K_P component the K_D component will decrease the oscillation, making the system more stable. Due to the controller having the error as input it will only respond when an error occur making the system never settling to the set point. Adding a K_I component calculating the integral of the error over time will make the system even more stable. Though, using this method to find accurate parameters for the PID controller did not make the system stabilize completely. The system became more stable for a short amount of time but was over a period of time not stable. The values that gave this system were:

$K_P = 50$, $K_I = 3$ and $K_D = 0,3$.

Chapter 5

Discussion

During this project a research has been done in how a balancing cube should be constructed. The chosen design with the sides of the cube being put together with three corner brackets in each of its corners resulted in a very robust design. Using a laser printer to cut out all the necessary parts from acrylic glass also benefited the construction since this method gave high precision measurements. It worked very well to use two plates as motor attachments. This design was very stable and also allowed all the sensor to be mounted on the plates. A disadvantage with this design is that it only works for two dimensional balancing. For multidimensional balancing another motor must be added, that would not fit in this construction.

No PID parameters where found that could balance the cube. There are many possible reasons to why it didn't work. Despite that a lot of focus have been on developing a mechanically balanced cube there is a risk that weight is not equally distributed on both sides. Also, the angles from the gyro does not always settle at the same point and that causes problem in the controlling of the cube. Another problem is that the weight of the motor makes it harder to control the cube because a low power to weight ratio reduces the maximum deviation angle from the equilibrium point that is when the angle is zero. There are also methods to iterate the PID parameters, this is something that could be investigated.

The Ziegler-Nichols method was used to determine the parameters for the PID controller. But when using the method the system never seemed to get into a steady state oscillation. Given that the method did not work for our system it could seem that either the method was not applicable to the given system or that the gain from the PID controller needed to be even higher. This was not possible to acquire for the cube due to the limitation of 24 V the cube was able to output.

Another method used was to use the angular velocity to get a more accurate input to the PID controller, due to the relationship between the angular velocity and the torque, see equation 2.16. The adding of this term did not seem to change the

output to the better, but gave similar or worse values than with the first mentioned method in this section. Regarding the values of the angular velocity in Figure 4.3 they did not seem to be that accurate with the reality. It could therefore be one explanation to why this method did not work and it could not be concluded that this method is bad because of this.

For the cube to be able to jump up from horizontal position to balancing on an edge it would require a voltage of approximately 36 V. An alternative to this would be to design a brake system with servomotors that can perform the *"jump"* action and then a smaller motor can balance the cube for smaller angles. This would not only reduce the required voltage and current supply, but also free space and that is very important if the project would expand to multi dimensional balancing. It would also be interesting to use brushless motors since they generally have a higher torque per weight ratio.

At first the dynamic model of the DC motor described in section 2.2.1 was intended to be used to regulate the motor torque so it corresponded with the computed output from the PID controller. Since it was difficult to know how reliable the measurement of the calculated velocity from the hall effect sensors were, the output was instead directly used to control the PWM signal to the motor.

Another thing that can be analysed is why the result from the experiment and the theoretical values deviated so much in Figure 4.3. When the angular velocity is small it looks like the two lines are parallel and that is good because the torque is proportional to the slope. When the velocity gets higher it differs a lot. This might be because the internal friction is not considered in mathematical model and it can be assumed to be proportional to the rotating velocity. Also the moment of inertia might have been underestimated. [5]

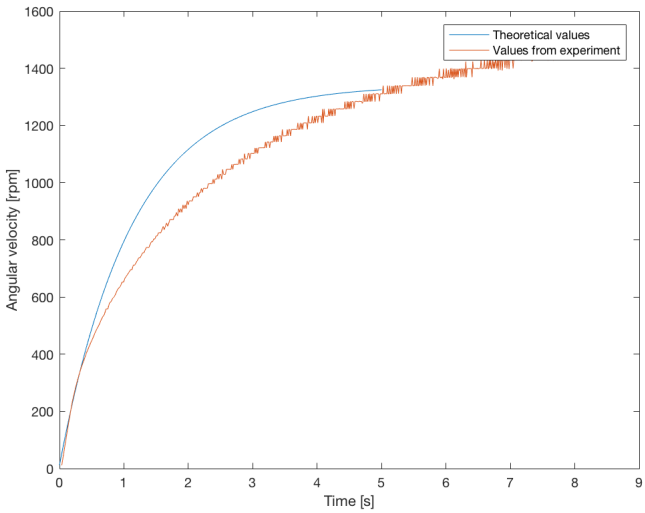


Figure 5.1. Comparison of theoretical rotating speed from *equation 2.12* and measured values, adjusted moment of inertia. Made in Matlab

In Figure 5.1 the reaction wheels moment of inertia has been increased with a factor of two. This was done because it makes the dynamical model looks more similar to the experimental values, but the curves still have different characteristics and therefore no conclusion can be taken.

Chapter 6

Conclusion

- What is a suitable way to construct a cube with the necessary hardware to make it balance on an edge using a reaction wheel?

A cube in acrylic glass makes a light, strong and easily made construction and with corner brackets in each corner it creates a robust shell that wont bend. Inside the cube, two plates separate the reaction wheels from the components while holding both them and the motor. All the components is carefully placed to maintain the center o mass in the center of the cube giving it a great mechanical stability. The sensors used gathered information and uses this as input in a PID controller that with the help of a motor driver and a DC motor controls the acceleration of the reaction wheels, balancing the cube.

- Is it possible with the resources available to balance the cube with a PID controller and what tuning method should be used?

Due to the inability to balance the cube with a PID controller for a longer period of time, it can not be concluded that with the resources given whether it is possible to balance the cube or not. Since the data from the IMU and the hall effect sensors was not fully reliable it can neither be concluded that the tuning method is the reason why the cube did not balance over time.

Chapter 7

Improvements

The balancing cube can be improved by using smaller corner brackets. The ones used in this project were very big and took up a lot of space in the cube. Smaller corner brackets would make it possible to have a bigger reaction wheel instead of two small, and that is more weight efficient. Also it would be recommended to use a brushless motor that generally can generate a lot of torque, but weights less. In this project hall effect sensors were used to measure speed and direction of the reaction wheels. Calculating the direction of the reaction wheels was difficult and therefore it is recommended to buy a rotary encoder instead that can be mounted on the reaction wheels. This was done when construction the Cubli that has be an inspiring source for this project. [15] In the project *Reaction Wheel Stabilized Stick* [18] a Kalman filter was used to reduce noise from the IMU. This was not used used to balance the cube because the noice appeared to be very small, but still this could be researched. To finish with, it might be possible with more knowledge in nonlinear control theory to be able to calculate the PID parameters that optimises the performance of the PID controller.

Bibliography

[1] Arduino.cc. *Ardunio uno rev3*.
https://store.arduino.cc/arduino-uno-rev3. Retrieved 2020-05-12.

[2] Arduino.cc. *micros()*.
https://www.arduino.cc/reference/en/language/functions/time/micros/. Retrieved 2020-05-11.

[3] Arduino.cc. *PID*.
https://playground.arduino.cc/Code/PIDLibraryConstructor/.
Retrieved 2020-03-29.

[4] Arduino.cc. *What is arduino?*.
https://www.arduino.cc/en/Guide/Introduction.
Retrieved 2020-05-12.

[5] Nicholas Apazidis. *Mekanik 1 Statik och partikeldynamik*.
Studentlitteratur, 2013.

[6] Nicholas Apazidis. *Mekanik 2 Partikelsystem, stel kropp och analytisk mekanik*.
Studentlitteratur, 2012.

[7] Brett Beauregard. *Arduino-PID-Library*.
https://github.com/br3ttb/Arduino-PID-Library/.
Retrieved 2020-05-28.

[8] Buehlermotor. *DC Motor 51x88*.
https://www.buehlermotor.com/fileadmin/user_upload/stock_service/datasheets/DC-Motor_51x88__1.13.044.2XX.pdf?fbclid=IwAR1ru7WgIEmUk8lEha7CSzmgLtvLIcKwwQ4AvKP0RA68nEo4glcVtn7Yeh0.
Retrieved 2020-05-28.

[9] Build Electronic Circuit. *H-bridge switches*.
https://www.build-electronic-circuits.com/wp-content/uploads/2018/11/H-bridge-switches.png. Retrieved 2020-05-28.

[10] Draw.io *Flowchart maker and online diagram software*,
https://app.diagrams.net/.
Retrieved 2020-05-28.

[11] Electrokit. *Pulsgivare halleffekt digital*. https://www.electrokit.com/uploads/productimage/41015/41015710.jpg. Retrieved 2020-05-28.

[12] Electrokit. *Motordrivare VNH3SP30 1 kanal 5.5-36V 30A*. Retrieved 2020-05-28. https://www.electrokit.com/uploads/productimage/41014/41014059.jpg. Retrieved 2020-05-28.

[13] Electrokit. *MPU-6050 accelerometer 3-axel gyro monterad på kort*. https://www.electrokit.com/uploads/productimage/41016/41016233.jpg. Retrieved 2020-05-28.

[14] Simon Frölander, André Säll. *Implementing Gyroscopic Stabilization*. 2017, KTH. oai:DiVA.org:kth-226681 http://www.diva-portal.org/smash/get/diva2:1201171/FULLTEXT01.pdf.

[15] Mohanarajah Gajamohan, Michael Merz, Igor Thommen, Raffaello D'Andrea. 2012. *The Cubli: A Cube that can Jump Up and Balance*. https://ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Research_DAndrea/Cubli/Cubli_IROS2012.pdf.

[16] Torkel Glad, Lennart Ljung. *Reglerteknik: grundläggande teori*. Studentlitteratur, 2006.

[17] Pixlr. *Pixlr Editor*. Online software 2020. https://pixlr.com/.

[18] Pontus Gräsberg, Bill Lavebratt. *Reaction Wheel Stabilized Stick*. KTH. 2019. oai:DiVA.org:kth-264481. http://www.diva-portal.org/smash/get/diva2:1373776/FULLTEXT01.pdf

[19] Arthur Grönlund, Christos Tolis. *Riderless self-balancing bicycle*. 2018. KTH. oai:DiVA.org:kth-230169 http://www.diva-portal.org/smash/get/diva2:1237256/FULLTEXT01.pdf.

[20] Dr. Hodge Jenkins. *Tuning for PID Controllers*, http://faculty.mercer.edu/jenkins_he/documents/TuningforPIDControllers.pdf.

[21] Hans Johansson. *Elektroteknik*. Instutitionen för maskinkonstruktion och mekatronik, 2013.

[22] Jeff Rowberg. *MPU6050*. https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050. Retrieved 2020-05-28.

[23] Fredrik Sandahl, William Miles. *Ball balancing robot*. 2017, KTH. oai:DiVA.org:kth-226683 http://www.diva-portal.org/smash/get/diva2:1201175/FULLTEXT01.pdf.

[24] ST. *Fully integrated H-bridge motor driver*. `https://www.electrokit.com/uploads/productfile/41014/vnh3sp30.pdf`. Retrieved 2020-05-12.

Appendix A

Electric Schematic

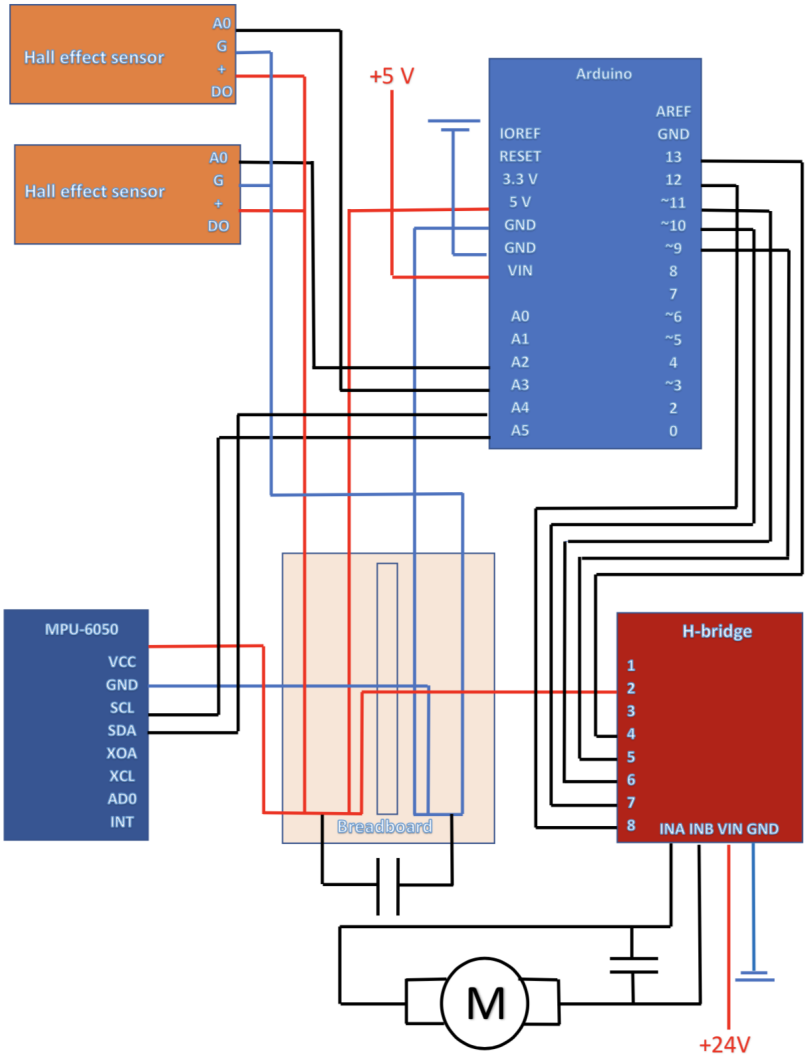


Figure A.1. Electric schematics for the balancing cube. Made in Microsoft Word.

Appendix B

Datasheet for the motor

Type / Baureihe 1.13.044.XXX				235	236
Characteristics*	**Nenndaten***				
Rated voltage	Nennspannung	U/V	V	12	24
Rated power	Nennleistung	P_N	W	50	50
Rated torque	Nenndrehmoment	T_N/M_N	Ncm	15	15
Rated speed	Nenndrehzahl	n_N	rpm/min^{-1}	3000	3000
Rated current	Nennstrom	I_N	A	6.2	3.1
No load characteristics*	**Leerlaufdaten***				
No load speed	Leerlaufdrehzahl	n_0	rpm/min^{-1}	3980	3900
No load current	Leerlaufstrom	I_0	A	0.4	0.2
Stall characteristics*	**Anlaufdaten***				
Stall torque	Anlaufmoment	T_S/M_H	Ncm	64	64
Stall current	Anlaufstrom	I_S/I_H	A	24	12
Performance characteristics*	**Leistungsdaten***				
max. Output power	max. Abgabeleistung	P_{max}	W	70	70
max. Constant torque	max. Dauerdrehmoment	T_{max}/M_{max}	Ncm	10	10
Motor parameters*	**Motorparameter***				
Weight	Gewicht	G	g	765	765
Rotor inertia	Läuferträgheitsmoment	J	gcm^2	180	180
Terminal resistance	Anschlusswiderstand	R	Ohm	0.5	2.0
Inductance	Induktivität	L	mH	0.5	1.0
Mech. time constant	Mech. Zeitkonstante	τ_m	ms	13	13
Electr. time constant	Elektr. Zeitkonstante	τ_e	ms	1.0	0.5
Speed regulation constant	Drehzahregelkonstante	R_n	rpm/Ncm	60	60
Torque constant	Drehmomentkonstante	k_I/k_M	Ncm/A	2.8	5.7
Thermal resistance	Thermischer Widerstand	R_{th}	K/W	9.5	9.5
Thermal time constant	Thermische Zeitkonstante	τ_{th}	min	9.5	9.5
Axial play	Axialspiel		mm	< 0.01	< 0.01
Direction of rotation	Drehrichtung			bidirectional / bidirektional	

Figure B.1. Datasheet for the electric motor. [8]

Appendix C

Source code

C.1 IMU experiment in Matlab

```
%  KEX13
%  PID Regulated Balancing Cube
%  PID Reglerad Balnserande Kub
%  Date 2020-05-29
%  Written by: Sebastian Brandmaier och Denis Ramsden
%  Examiner: Nihad Subasic
%  TRITA-ITM-Ex: 2020:33
%  Course code: MF133
%
%  Bachelor thesis at KTH in mechatronics
%
%  This program plots values from an IMU and compares
%  how it stabilizes over time with or without a capacitor.

%% clean command window and workspace
clear all, close all, clc

%Read data when no capcitor is used from a text file.
data = load("gyro.txt")
Time1 = data(:,2)/(10^6)
angle1 = data(:,1) %Angle without capacitor


%Read data when no capcitor is used from a text file.
data = load("gyro_with_capacitator.txt") %with capacitor on IMU
Time2 = data(:,2)/(10^6)
angle2 = data(:,1) %Angle without capacitor


figure % Plot the result in the same figure
plot(Time1, angle1)
hold on
plot(Time2, angle2)
xlabel('Time since MPU-6050 was powered on [s]')
```

```
ylabel('Angle [degrees]')
legend('IMU without capacitor', 'IMU with capacitor')
```

C.2 Motor experiment in Matlab

```
%  KEX13
%  PID Regulated Balancing Cube
%  PID Reglerad Balnserande Kub
%  Date 2020-05-29
%  Written by: Sebastian Brandmaier och Denis Ramsden
%  Examiner: Nihad Subasic
%  TRITA-ITM-Ex: 2020:33
%  Course code: MF133X
%
%  Bachelor thesis at KTH in mechatronics
%
%  This program plots values from an experiment and
%  compares it with a theoretical model.
%

%% clean command window and workspace
clear all, close all, clc

% Calculations with more decimals.
format long

%% Mathematical motor model:

% Mehanical constants
m_cube = 2 % Total mass of th cube [kg]
l_cube = 15*10^(-2) % length of cube [m]
m_wheel = 0.300 % Reaction wheel mass[kg]
r_wheel = (11/2)*10^(-2) % radius of the flywheel [m]
I_wheel = m_wheel*r_wheel^2 % Moment of inertia of the flywheel [ kgm
    ^2]
I_motor = 180*10^(-7) % momen of inertia of the rotor shaft [ kgm^2]

J = I_wheel+I_motor; %Total moment of inertia on the motor.

%motor varibles:
omega = 0; % Angular velocity [rad/s]
u = 8 % Voltage [V]


%Motor specifications:
Ra = 2 % Terminal resistance [ohm]
Kt = 5.7*10^(-2) %Torque constant Nm/A
U_max = 8 % max voltage [V]

% Other constants
N = 500 % Number of iterations
```

```
%%Motor dynamics:
t = 5 % tid [s]

dU = U_max/(N-1) % Voltage differens
dt = t/(N-1) % Time difference

T = [0:dt:t]; % Time vector



M = [] % Stores generated torque values
OMEGA = []; %Stores velocity att different time

for x = T
  torque = (Kt/Ra)*U_max-(((Kt^2)/Ra))*omega;  % Calculate torque

  acceleration = torque/J; % Calculate acceleration
  omega = omega + acceleration * (t/(N-1)); %Calculate angular velocity
      , euler forward.

  OMEGA = [OMEGA omega]; % Store in values in a vector.
   M = [M torque];
end

OMEGA = OMEGA*60/(2*pi) % Convert from rad/s to rpm

figure
plot(T,OMEGA) % Plots theoretical values.
xlabel('Time [s]')
ylabel('Angular velocity [rpm]')
set(gcf,'color','w');
hold on
%title('U=24V')

%% Values from experiment

data = load("kex/8v_motor.txt") %Read data from a text file.
Time = (data(1:666,1)/(10^6))-1.3;
rpm = data(1:666,2)*0.5;
plot(Time, rpm)
legend('Theoretical values', 'Values from experiment')
```

C.3 Arduino code in C++

```
/* KEX13
 *  PID Regulated Balancing Cube
 *  PID Reglerad Balanserande Kub
 *  Date 2020-05-29
 *  Written by: Sebastian Brandmaier och Denis Ramsden
 *  Examiner: Nihad Subasic
 *  TRITA-ITM-Ex: 2020:33
 *  Course code: MF133X
 *
 *  Bachelor thesis at KTH in mechatronics
 *
 *  This code tries to balance a cube with help of reaction wheels.
 *  The components for this used is:
 *    - 1 pcs Arduino UNO
 *    - 1 pcs 24V burshed DC motor.
 *    - 1 pcs motor driver VNH3SP30,
 *    - 2 pcs hall effect sensor,
 *    - 1 pcs gyroscope MPU6050,
*/

#include <PID_v1.h> // Include PID library

// Varibles for motor controll
const int inaPin = 13;
const int inbPin = 9;
const int pwmPin = 11;
const int diagaPin = 10;
const int diagbPin = 12;
const int buttonPin = 2;
const int trimPin = A0;
int on = 0;
int i = 0;
double pwmSignal;

// Motor constants
double k2 = 0.057;
double R = 2;

// Varible us for the gyro to be able to stabilze
double Time;

// PID-varibles
double input;
double output;
double setpoint;

//Different PID setups
double Kp = 50 , Ki = 3, Kd= 0.3;  //PID-parmeters
//double Kp = 35 , Ki = 0, Kd= 0.03 ; // kp= 37 , Ki=0 Kd=0.05      //
    speciql 2400 //35 nuuu
```

```
//double Kp = 42  , Ki = 1, Kd= 0.3;

float roll_deg; // stores the angle the cube is tilted
float real_output;  //Adjusting outputsignal

// Defines a PID controller
PID PID_controller(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);


// Pin for reading hall effect sensor values
int magnetSensor1 = A3;
int magnetSensor2 = A2;

// Store values from hall effect sensors
int Sensorval1;
int Sensorval2;

// Help variables to calculate speed direction
int state1;
int state2;
int state3;


double oldTime; // Define a varible that current stores time.
double currentTime; // Define a varible that old stores time.
double dx; // Stores a time difference
double radVelocity; // Stores velocity in radians
double velocity; // stores velocity in rpm
double U; // Voltage, number 0-255, 255 results in a 24V of output
    voltage.


// Code below is written by Jeff Rowberg and can be found at https://
    github.com/jrowberg/i2cdevlib

// I2Cdev and MPU6050 must b installed as libraries, or else the .cpp/.
    h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include
    file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
    implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense
```

```
    evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high


// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions
    coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0
    = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42
    bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor
    measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel
    sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel
    sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll
    container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00,
    '\r', '\n' };


// ================================================================
// ===                      INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU
    interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}
```

```
// ===============================================================
// ===                      INITIAL SETUP                      ===
// ===============================================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output,
        but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue
        immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or
        Ardunio
    // Pro Mini running at 3.3v, cannot handle this baud rate reliably
        due to
    // the baud timing being too misaligned with processor ticks. You
        must use
    // 38400 or slower in these cases, or use some kind of external
        separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection
        successful") : F("MPU6050 connection failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and
        demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
```

```
  mpu.setXGyroOffset(112);
  mpu.setYGyroOffset(55);
  mpu.setZGyroOffset(14);
  mpu.setZAccelOffset(9694); // 1688 factory default for my test chip

  // make sure it worked (returns 0 if so)
  if (devStatus == 0) {
      // turn on the DMP, now that it's ready
      Serial.println(F("Enabling DMP..."));
      mpu.setDMPEnabled(true);

      // enable Arduino interrupt detection
      Serial.println(F("Enabling interrupt detection (Arduino
          external interrupt 0)..."));
      attachInterrupt(0, dmpDataReady, RISING);
      mpuIntStatus = mpu.getIntStatus();

      // set our DMP Ready flag so the main loop() function knows it'
          s okay to use it
      Serial.println(F("DMP ready! Waiting for first interrupt..."));
      dmpReady = true;

      // get expected DMP packet size for later comparison
      packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
      // ERROR!
      // 1 = initial memory load failed
      // 2 = DMP configuration updates failed
      // (if it's going to break, usually the code will be 1)
      Serial.print(F("DMP Initialization failed (code "));
      Serial.print(devStatus);
      Serial.println(F(")"));
  }

  // Code written by Jeff Rowberg ends here

Time = micros(); // Return time since the arduino was powered on.


// Setup pins for motor control as specified in datasheet fpr the
    VNH3SP30.
pinMode(buttonPin, INPUT);
pinMode(inaPin, OUTPUT);
pinMode(inbPin, OUTPUT);
pinMode(pwmPin, OUTPUT);
pinMode(diagaPin, INPUT);
pinMode(diagbPin, INPUT);
pinMode(trimPin, INPUT);



// Setup for PID controller
PID_controller.SetMode(AUTOMATIC);
PID_controller.SetTunings(Kp,Ki,Kd);
```

```
  PID_controller.SetOutputLimits(-255, 255);


  // Vinkelhastighet
  oldTime = micros(); // Stores the time when the program is start
      running.
  state3 = 0; // Sets initial state to 0.

}


// Code below is written by Jeff Rowberg and can be found at https://
    github.com/jrowberg/i2cdevlib

// ================================================================
// ===                      MAIN PROGRAM LOOP                     ===
// ================================================================

void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        // other program behavior stuff here
        // .
        // .
        // .
        // if you are really paranoid you can frequently test in
            between other
        // stuff to see if mpuInterrupt is true, and if so, "break;"
            from the
        // while() loop to immediately process the MPU data
        // .
        // .
        // .
    }

    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is
        too inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        Serial.println(F("FIFO overflow!"));

    // otherwise, check for DMP data ready interrupt (this should
        happen frequently)
```

```
    } else if (mpuIntStatus & 0x02) {
        // wait for correct available data length, should be a VERY
            short wait
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an
            interrupt)
        fifoCount -= packetSize;


        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
         //    Serial.print("ypr\t");
          //   Serial.print(ypr[0] * 180/M_PI);
          //   Serial.print("\t");
           //  Serial.print(ypr[1] * 180/M_PI);
         //    Serial.print("\t");
          //   Serial.println(ypr[2] * 180/M_PI+97);
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);


 // Code written by Jeff Rowberg ends here



roll_deg = ypr[2] * 180/M_PI+86.5; // Stores data from the gyra in a
    varible, was adjuste so the angle is 0 when the cube is in
    horisontal position.

input = roll_deg;  // Uses angle as input signal for the PID controller
    .


//Calculation of the angular speed of the reaction wheels.

//Read value from hall effect sensor 1
Sensorval1 = analogRead(magnetSensor1);  // 560 55

//Read value from hall effect sensor 2
Sensorval2 = analogRead(magnetSensor2);  // 650 630


  if(state3 == 1){
```

```
    if(state1==1 && Sensorval1 > 500){ //If sensor 1 is sensing a
        magnet, used to calculate direction
      state2 = 1;
      state3 = 0;
      //Serial.println(state2);
    }

    if(state1==1 && Sensorval2 > 600){ //If sensor 2 is sensing a
        magnet, used to calculate direction
      state2 = 2;
      state3 = 0;
      //Serial.println(state2);
    }
  }


  if(Sensorval1 > 500 && Sensorval2 > 600){ // If both sensor i sensing
      a magnet
    if(state1 == 1){ //If both sensors has been low, to avoid measure
        many times at the same magnet.
      currentTime = micros(); // get time
      dx = currentTime - oldTime; // calculate time difference
      radVelocity = 1000000*3.1415/dx; // calculate angular speed
      oldTime = currentTime; // set current time to new time
   //    if(state2 == 2){
    //     radVelocity = -1*radVelocity;
    //   }

      state1 = 0;
      state3 = 1;
    }
  }

  if(Sensorval1 < 500 && Sensorval2 <  600){ // If both sensors are low
      , change state to 1.
    state1 = 1;
  }



if(state2 == 1){ // State2 stores the direction that the wheel is
    spinning.
  velocity = -1*radVelocity; // Neagtiv direction
}
if(state2 == 2){
  velocity = radVelocity; // Positive.
}


// Code for PID
setpoint = 45; //0 degrees offset from the equillibrium
```

```
  if(micros()-Time > 8000000){ // Needed a delay so that the IMU could
      stabilaze, using delay() caused overflow.
     PID_controller.Compute(); // Compute new output value

     U=k2*velocity+(R/k2)*output; // Compute voltage the gives a
         certain torque, dempending on velocity speed.


   // if((input>46) || (input<44)){
     if(output >= 0){ // Since arduino only sorks with positive current
        , the direction was swapped instead.
      pwmSignal = output;
      digitalWrite(inaPin, LOW); //CW direction of motor.
      digitalWrite(inbPin, HIGH);
    } else {
      pwmSignal = -1*output;
      digitalWrite(inaPin, HIGH); //CCW direction of motor.
      digitalWrite(inbPin, LOW);
    }
   }

Serial.print(roll_deg); // Print the angle in serial monitor.
Serial.print("␣");
Serial.print(input); // Print input
Serial.print("␣");
Serial.println(output);  // Print output

// Output pwm signal.
analogWrite(pwmPin, pwmSignal);

}
}
```

TRITA -ITM-EX 2020:33