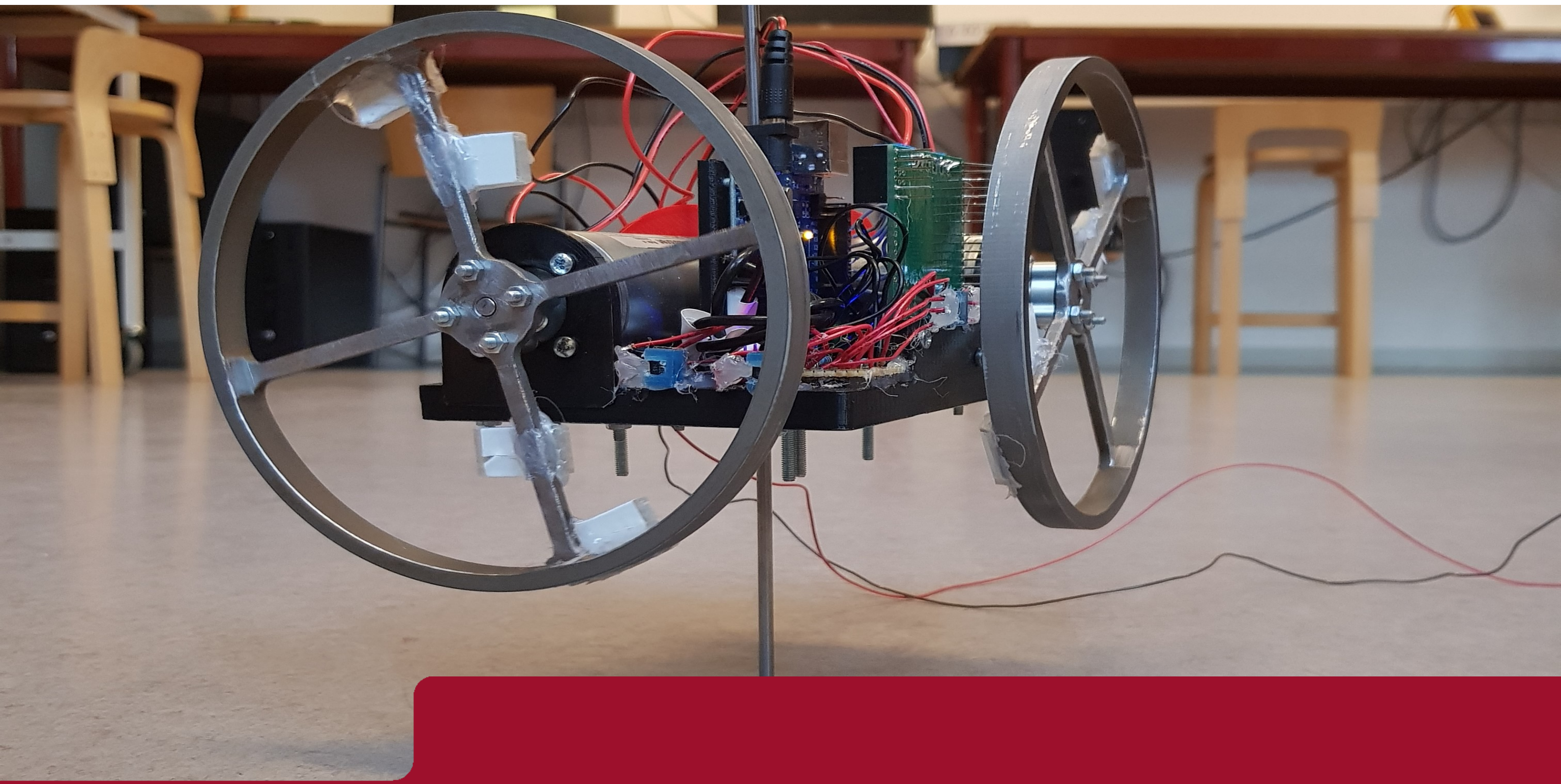
DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2019

Reaction Wheel Stabilized Stick

PONTUS GRÄSBERG

BILL LAVEBRATT



KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

Reaction Wheel Stabilized Stick

Bachelor's Thesis in Mechatronics

PONTUS GRÄSBERG
BILL LAVEBRATT

Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2019:38

Abstract

Control theory can be used to make an unstable system stable. This thesis seeks to do this, where the system is a two DOF inverted pendulum with reaction wheels for stabilisation. The thesis also seeks to answer what is most important for making it stabilize for a longer period of time. It was decided that a state space controller was to be used with various sensors measuring the states. To be able to design a functioning demonstrator, a mathematical model of the system dynamics was developed. In the end the demonstrator proved to function as desired, being able to balance indefinitely. It was found that it is absolutely necessary to either give the controller a perfect set point or to implement an automatic set point.

Keywords: Mechatronics, Reaction wheel, Inverted pendulum, State space, Control theory.

Referat

Reaktionshjuls stabiliserad pinne

Reglerteknik kan användas för att göra ostabila system stabila. Målet med detta projekt var att göra detta med ett system i form av en inverterad pendel med två frihetsgrader som balanseras med hjälp av två svänghjul. Projektet söker att besvara frågan om vad som är de viktigaste faktorerna för att få systemet att vara stabilt över en längre tid. En tillståndsåterkoppling användes som regulator vilket innebar att flera olika sensorer behövdes för att mäta de olika tillstånden. För att kunna konstruera en fungerande prototyp utvecklades en matematisk modell av systemet vilken användes för simulering av systemet. Till slut konstruerades en fungerade prototyp som till synes kunde balansera över oöverskådlig tid. En av de faktorer som visade sig påverka huruvida systemet uppnår stabilitet över längre tid var hur bra referenspunkt som gavs till regulatorn, det vill säga det tillstånd som regulatorn reglerar systemet mot. Det visade sig vara möjligt att implementera en självjusterande referenspunkt som gjorde systemet stabilt över tid.

Acknowledgements

First of all we would like to thank KTH prototype center and Tomas Östberg for helping us make the reaction wheels. Also a big thanks to Staffan Qvarnström for helping us order parts and finding motors for the project as well as to the lab assistants for helping us and answering various questions.

Contents

List of Figures

Nomenclature

Name	**Description**
m	The mass of the structure
I	The moment of inertia of the structure
θ	Angular diviation from equilibrium point
l	Height of the center of mass
M_{motor}	torque applied by motor
ω	Angular velocity of reaction wheel
$k_2\phi$	Torque constant of motor
R_A	Terminal resistance of motor
U	Voltage to the motors
θ_{IMU}	Angle of IMU
dx	Set point
T	Time period between the updates of PWM signal
L	Feedback gain vector
k	Gain for automatic set point

List of Abbreviations

Abbreviations	**Description**
DOF	Degrees of Freedom
IMU	Inertial measurement unit
MEMS	microelectromechanical systems
PWM	Pulse Width Modulation
ODE	Ordinary differential equation
RPM	Revolutions per minute

Chapter 1

Introduction

1.1 Background

The problem of balancing an inverted pendulum is a classical problem in the field of dynamics and control theory. The problem arises for example when dealing with rockets or self balancing robots. One method to balance the inverted pendulum is with reaction wheels.

A reaction wheel is a flywheel that is designed to give a reaction torque when torque is applied to it. Reaction wheels are primarily used in spacecrafts for attitude control, eliminating the need for thrusters. The advantage of reaction wheels is that they can apply a torque to a system in an environment where there is no external structure or medium to provide a torque.

1.2 Purpose

The Purpose of the project is to use reaction wheels to stabilize an inherently unstable system. In this case the system is a two DOF inverted pendulum, when left alone the pendulum will fall. The objective is to keep the pendulum standing. By using motors to apply torque on the reaction wheels reaction forces will arise which could keep the pendulum standing. This thesis seeks to answer:

- In what way is it possible to construct a functioning demonstrator with the resources available?
- What are the most important parameters in order to keep the system stable over time?

1.3 Scope

Even though the dynamics of the system is nonlinear this thesis only considers linear control methods. There are several types of controllers that could work, but this thesis investigates only how a state space controller performs. A micro controller will be used to control the system. The demonstrator shall not be of a height of more than 0,4 m and shall carry all components needed except the power supply for the motors.

1.4 Method

First the dynamics of the system was analysed and a mathematical model was developed. With the mathematical model of the system, simulations were made in MATLAB. The simulations were used to test various designs and controllers. With constrains given by the simulations, components were chosen and gathered. The demonstrator was built and tests were conducted to determine which parameters are of the most importance to balance the pendulum.

Chapter 2

Theory

2.1 System dynamics

In order to understand the characteristics of the system and to be able to design an effective control algorithm, a simplified mathematical model of the system was developed.

2.1.1 Dynamics of the system

The dynamics of the pendulum can be simplified by analysing its dynamics in only one plane. If the chosen plane is parallel with one of the reaction wheels the system simplifies to that of an inverted pendulum in one DOF with a single reaction wheel attached to it. This pendulum can only rotate around one axis. This simplification can be made since the action of the second reaction wheel, perpendicular to the plane, should not greatly affect the dynamics of the pendulum in the plane being analysed.

According to figure 2.1 and Newtons second law, the rotational dynamics of the pendulum around O can be described as

$$e_z : I\ddot{\theta} = mglsin(\theta) - M_{motor}, \tag{2.1}$$

where I is the moment of inertia in the z-direction of the pendulum at O, θ is the angular deviation from the equilibrium point when the pendulum is upright, m is the mass of the system, l is the distance to the center of gravity and M_{motor} is the torque applied by the motor on the reaction wheel. M_{motor} can be thought of as the reaction torque between the pendulum and the reaction wheel. Therefore the rotational dynamics of the reaction wheel can be described as

$$e_z : I_r\dot{\omega} = M_{motor}, \tag{2.2}$$

where I_r is the combined moment of inertia of the reaction wheel and the rotor of the motor and ω is the angular velocity of the reaction wheel and the motor.

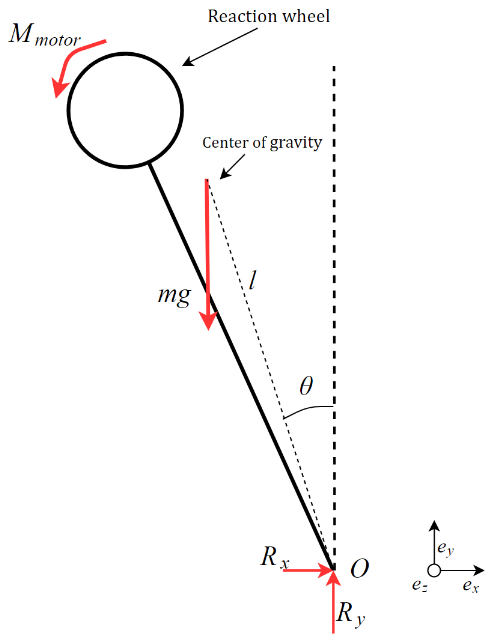


Figure 2.1. Force analysis of inverted pendulum. Drawn in draw.io.

Knowing that the same equations applies in the yz-plane the model should be sufficiently accurate for the purpose to stabilize the pendulum.

2.1.2 Motor dynamics

The motors used in this project were a pair of brushed DC motors. A relation between the angular velocity of a motor, the voltage applied on the motor and the motor torque can be stated as

$$M_{motor} = \frac{k_2\phi}{R_A}U - \frac{k_2\phi^2\omega}{R_A}, \tag{2.3}$$

where $k_2\phi$ is the torque constant, R_A is the terminal resistance and U is the voltage applied on the motor[1].

2.1.3 State space model

To simplify the analysis of the system and the design of the controller, the system dynamics can be written as a set of first order linear differential equations. Since θ can be assumed to be small and $sin(\theta) \approx \theta$ for small θ, equation 2.1 can be linearised as

$$I\ddot{\theta} = mgl\theta - M_{motor}. \tag{2.4}$$

The state variables x_1, x_2 and x_3 are introduced as $x_1 = \theta$, $x_2 = \dot{\theta}$ and $x_3 = \omega$. With equation 2.2, 2.3 and 2.4 the dynamics can now be written in the following form

$$\dot{x}(t) = Ax(t) + Bu(t)) \tag{2.5}$$

$$y(t) = Cx(t) + Du(t) \tag{2.6}$$

as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{mgR}{I} & 0 & \frac{k_2\phi^2}{R_A I} \\ 0 & 0 & -\frac{k_2\phi^2}{R_A I_r} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{k_2\phi}{R_A I} \\ \frac{k_2\phi}{R_A I_r} \end{bmatrix} u(t) \tag{2.7}$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{2.8}$$

which is the state space model of the system. Here, $u(t)$ is the input variable, or the voltage in this case, and $y(t)$ is the output variable, or the state that is to be controlled, which in this case is θ.

2.2 Control design

The system itself is not asymptotically stable, which can be verified by computing the eigenvalues of matrix A. If the real part of the eigenvalues are strictly negative the system is asymptotically stable [2]. To change the dynamics of the system and make it asymptotically stable, active feedback can be used by letting the voltage U depend on the state variables.

In this project a state space feedback controller with integral action was chosen as the controller. Integral action makes the system more robust to modelling errors and disturbances [2]. A new state variable x_4 is introduced to denote the integrator state which is the integral of the output error. Let r denote the reference or the wanted output, then the following holds:

$$\dot{x}_4 = r - y = r - Cx. \tag{2.9}$$

The system becomes a closed-loop system as the states are fed back to form the input as

$$u(t) = -Lx = -\begin{bmatrix} L_1 & L_2 & L_3 & L_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \tag{2.10}$$

where the vector L contains the feedback gain. All states can be measured. The resulting system now becomes

$$\begin{bmatrix} \dot{x} \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ x_4 \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r \tag{2.11}$$

and with the feedback

$$\begin{bmatrix} \dot{x} \\ \dot{x}_4 \end{bmatrix} = \left(\begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix} - \begin{bmatrix} B \\ 0 \end{bmatrix} L \right) \begin{bmatrix} x \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r. \tag{2.12}$$

By choosing L, the poles of the system can be placed as desired. Since the system is asymptotically stable, it will converge to a static state that satisfies $r - Cx = 0$. In other words $x_1 = r$. In this case $r = 0$ since the pendulum shall be balancing around the equilibrium point where $\theta = 0$.

2.2.1 Simulation and Pole Placement

Simulation and pole placement were done in MATLAB. The code can be seen in appendix C.1. The system dynamics is given by equation 2.12. The system of differential equations can for example be solved numerically with a Runge-Kutta method. A fourth ordered Runge-Kutta method was implemented in MATLAB. By building the ODE solver and not using one of MATLABs predifined ODE solvers, things like maximum voltage and anti windup could more easily be accounted for. The simulation gives the behaviour of all the states, the torque applied by the motors and the input voltage. Figure 2.2 and 2.3 show the angular deviation of the pendulum and the input voltage when some poles with corresponding L were chosen. Here the initial conditions of the states were chosen to be 0,1 rad for the angular deviation of the pendulum and zero for the rest.

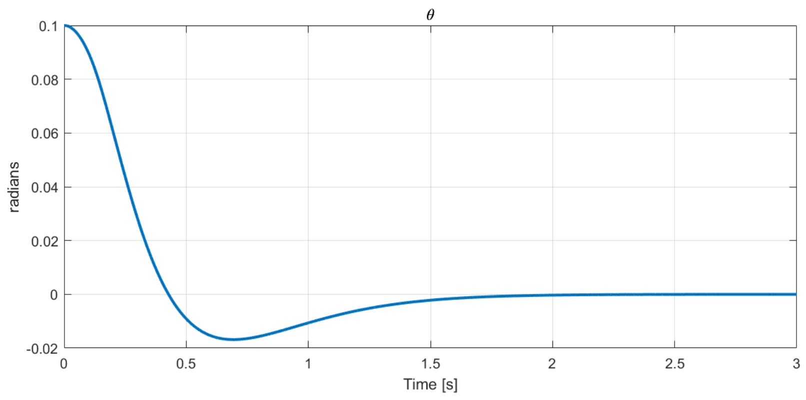


Figure 2.2. Simulation of angular deviation of the pendulum. Made in MATLAB.

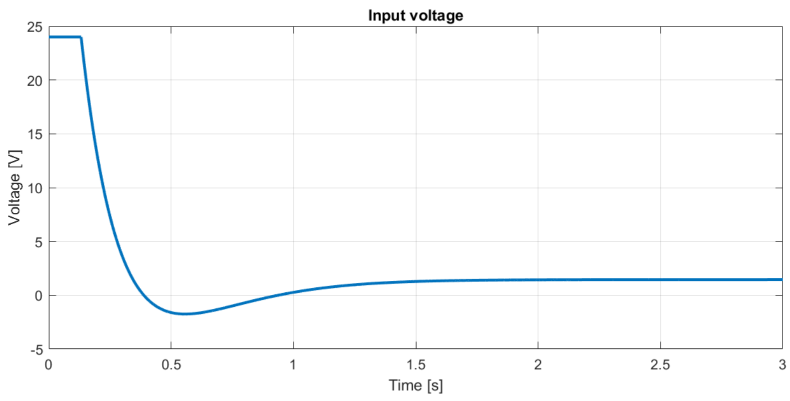


Figure 2.3. Simulation of corresponing input voltage. Made in MATLAB.

2.3 Sensors

In order to use a state space controller it is necessary to somehow measure or estimate the states. In this project the states were measured by using a few different sensors. The accelerometer and the gyroscope were used to measure the angle and the angular velocity of the system, and the the Hall effect sensors were used to measure the angular velocity of the reaction wheels.

2.3.1 Accelerometer

Accelerometers measure the acceleration. To measure the acceleration accelerometers typically use a mass spring system which measures the capacitance. There might be vibrations in the structure which for the accelerometer in this case are unwanted noise. The measurement also has a bias which will be constant and can therefore be mostly removed[3]. The principle of one type of MEMS accelerometer is shown in figure 2.4.

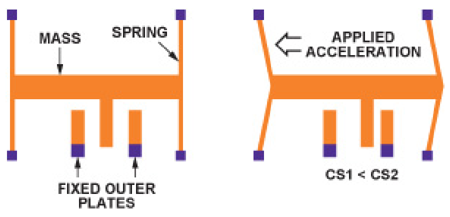


Figure 2.4. MEMS accelerometer structure [3].

2.3.2 Gyroscope

A gyroscope measures the angular velocity. It normally does this by having a spinning disc and using the Coriolis force. A cheaper and often a more practical alternative is to use a microelectromechanical system (MEMS) gyroscope. However because the calculation of the angle requires integration of the angular velocity measurement, the error of the gyroscope drifts with time [4]. The drifting of the error is caused by multiple factors and has complex non-linear characteristics [4].

2.3.3 Hall effect sensor

A Hall effect sensor is a device that responds to a magnetic field by generating an output voltage whose magnitude depends on that of the magnetic field. There are Hall effect sensors with a built in Schmitt trigger making the output digital. Such sensors were used to measure the angular velocity of the reaction wheels.

2.4 Kalman filter

Because of the noise that occurs in the MEMS sensors as described in section 2.3 it is of importance to filter the data gathered from the sensors. To achieve this a Kalman filter was used. The name implies that the Kalman filter works as a filter, however it works more like an observer. An observer is a system that estimates unknown states with measured states. The idea of a Kalman filter is to use current and past sensor readings and to use probability and statistics to minimise the mean squared errors to estimate a state [5]. It was of importance to not simply filter the data with for instance a butterworth low-pass filter, since such a filter would delay the data signal in time, which can cause the system to become unstable or overshoot when changing rapidly. If the measurements are very delayed the state space controller reacts too slow to the change in angle and thereby making the system more unstable. For a more exact mathematical description on Kalman filter see [5].

2.5 Controlling motors

To control the motors it was necessary to be able to control the voltage given to the motors. This can be achieved by sending a Pulse Width Modulation (PWM) signal to an H-bridge. A PWM signal is, as the name implies, a signal consisting of pulses of high and low. By changing the pulse width, or the portion of the time where the signal is high, the mean value of the signal changes. The PWM signal opens and closes switches in the H-bridge that control if current from the external power supply can flow through. Using this technique the output voltage to the motors can be controlled. Figure 2.5 shows 2 different PWM signals and the resulting average voltage.

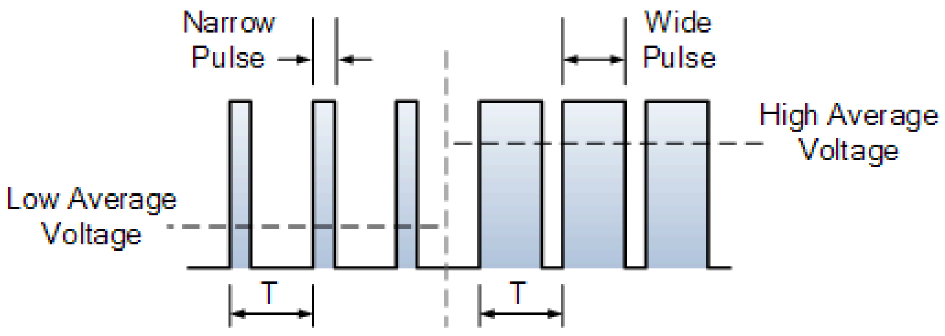


Figure 2.5. Two examples of PWM signals [6]

An H-bridge is a circuit with transistors making it possible to switch the polarity of the output voltage. This way the motors could be controlled in both directions. Figure 2.6 shows a circuit of an H-bridge.

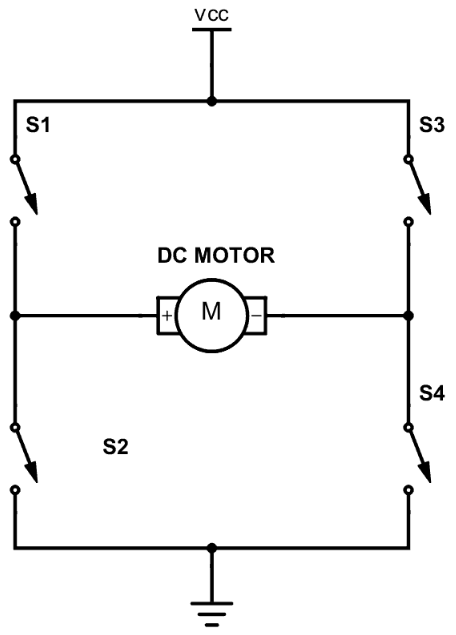


Figure 2.6. A simple H-bridge circuit [7]

Chapter 3

Design and implementation

This chapter describes the design of the demonstrator, the relevant hardware that was used and the reasoning behind them. It also considers the software implementation. The simulation model in MATLAB with data from the CAD model indicated how the demonstrator should be designed to function as desired. Due to cost limitations and components available our ideal design could not be achieved and some compromises had to be made.

3.1 Motors

Simulations were done in order to find approximate requirements for the motors. The system performance is largely dependent on its mass and the power that the motors can deliver, and since the power a motor can deliver is proportional to its mass, there was no easy way to calculate the exact requirements on the specifications of the motors. Here the mathematical model of the system dynamics that was built was crucial. It made it possible to simulate the system behaviour with various parameter values. A motor with high power to weight ratio could improve the system performance. It was estimated that a motor that can deliver around 0,2 Nm of torque and with the mass of about 0,4 kg could stabilize the system. One of the motors available with similar specifications was a motor manufactured by Dunkermotoren of model GR 42x40. The mass of this model is 0,49 kg and it can deliver 0,33 Nm during stall torque. It was found in the simulations that the pendulum should be able to balance using this motor. The data sheet for the GR 42x40 motor can be seen in appendix A, figure A.1.

3.1.1 Motor driver

Since the Arduino is not powerful enough to drive the motors, an external power supply had to be used. To control the polarity and the size of the voltage supplied to the motors, a dual H-bridge was used. The aim was to be able to supply the motors with 24 V which could draw as much as 4 A for short periods of time. From these conditions the H-bridge Pololu Dual VNH5019 Motor Driver was chosen. This H-bridge can operate at 24 V and deliver a continuous current of 12 A per channel, which is sufficient for the purpose. The VNH5019 also features PWM operation up to 20 kHz which enables quieter and smoother running of the motors[8]. General specifications for the H-bridge taken from the Pololu website can be seen in appendix A figure A.2.

3.2 Measuring motor speed

The angular velocity of the motors can be measured using Hall effect sensors that detect magnets which are attached on the reaction wheels. In order to tell which direction the motors are rotating, two Hall effect sensors per motor were used. The figure below illustrates how the direction is determined.

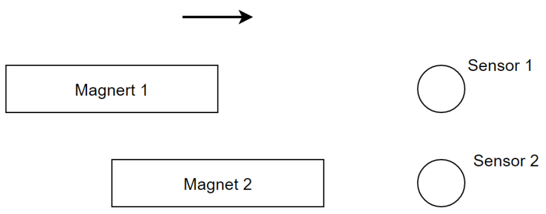


Figure 3.1. Illustration of how to measure direction. Drawn in draw.io.

When sensor 1 detects a magnet the system checks if sensor 2 is detecting a magnet. If so, the magnets are moving to the right, otherwise the magnets are moving to the left. By counting detections and measuring the time difference between the detections the angular velocity can be calculated.

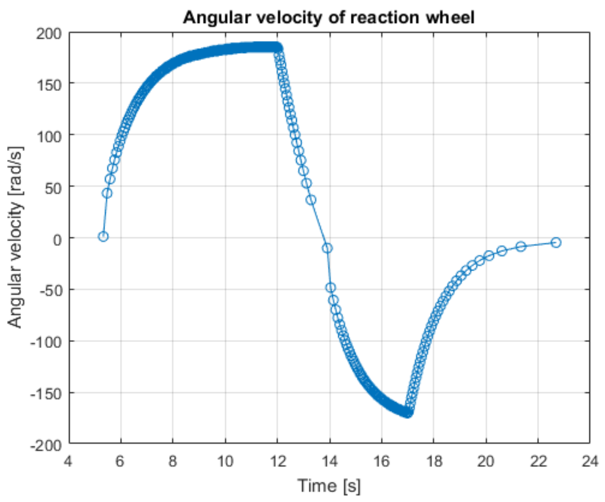


Figure 3.2. Plot of measured angular velocity of reaction wheel. Made in MATLAB.

Figure 3.2 shows the measured angular velocity of one of the reaction wheels where the motor was given 24 volts for seven seconds, -24 volts for seven seconds and then zero volts until the angular velocity became zero.

3.3 Inertial measurement unit

For measuring the angle a MEMS gyroscope and accelerometer was used. The chosen sensor was the MPU-6050. As described in chapter 2 these measurements have both noise and drift, therefore a Kalman filter was used. To implement the Kalman filter a library and example code made by Kristian Sloth Lauszus was used [9].

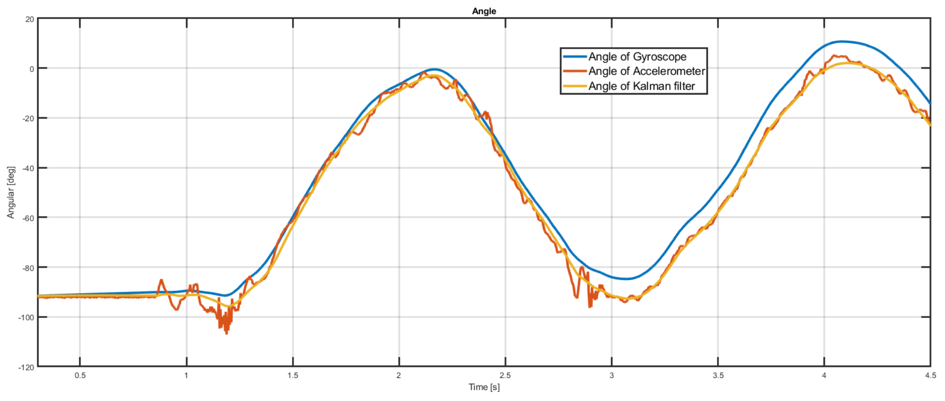


Figure 3.3. The angle as given by the accelerometer, gyroscope and Kalman filter. Made in MATLAB.

Figure 3.3 shows the angle as given by the accelerometer, gyroscope and Kalman filter. It can be seen that the accelerometer had a lot of noise and that the gyroscope drifted with time as stated in section 2.3. The Kalman filtered data has low noise and is not time delayed as some filters can cause, making it suitable for use in the controller. Note that figure 3.3 shows two 90° turns of the system. In reality the the demonstrator will only move a few degrees making the value given directly by the accelerometer useless because of the noise to signal ratio.

3.4 Construction

It was decided that a platform to hold the components and the stick together would be 3D-printed. The aim of the platform was to get the system as compact as possible and the weight close to the stick to get the center of mass approximately above it. It was also of importance to fasten the motors to reduce vibrations caused by them. The design also features an adjustable height of the platform since there was uncertainties about which height that would work. The stick chosen was a beam of steel with a radius of 2 mm.

The reaction wheels were designed to have low mass while simultaneously keeping a high moment of inertia. They were designed in Solid Edge and then manufactured by using a water cutter to cut out the silhouette out of a sheet of steel, and then milling material away from the inner parts to lower the mass while keeping a high moment of inertia. The whole structure was then put together with screws and set screw hubs as shown in figure 3.4 where the CAD model is seen. All of the electrical components were also put on the platform and connected as the schematics figure B.1 illustrates. Figure 3.5 shows the finished demonstrator while balancing. Here the lower side of the platform is placed on a height of 10 cm and the diameter of the reaction wheels is 15 cm.

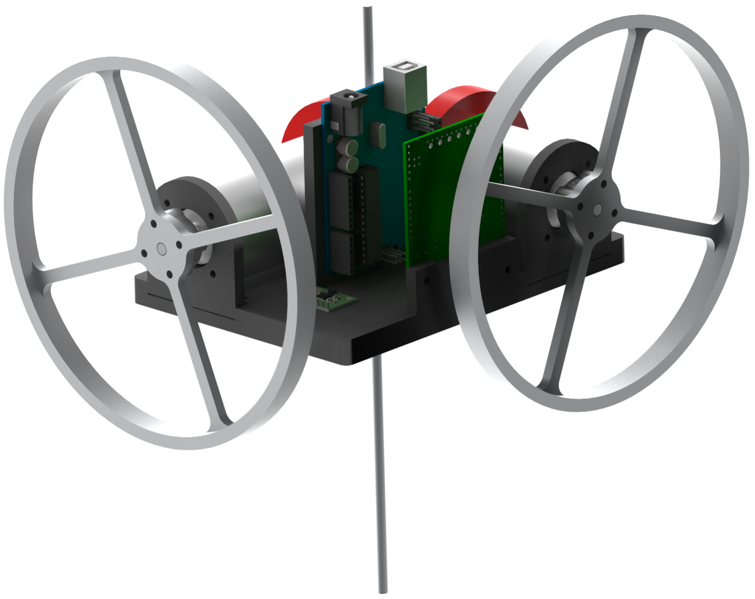


Figure 3.4. Cad model. Made in Solid Edge ST10.

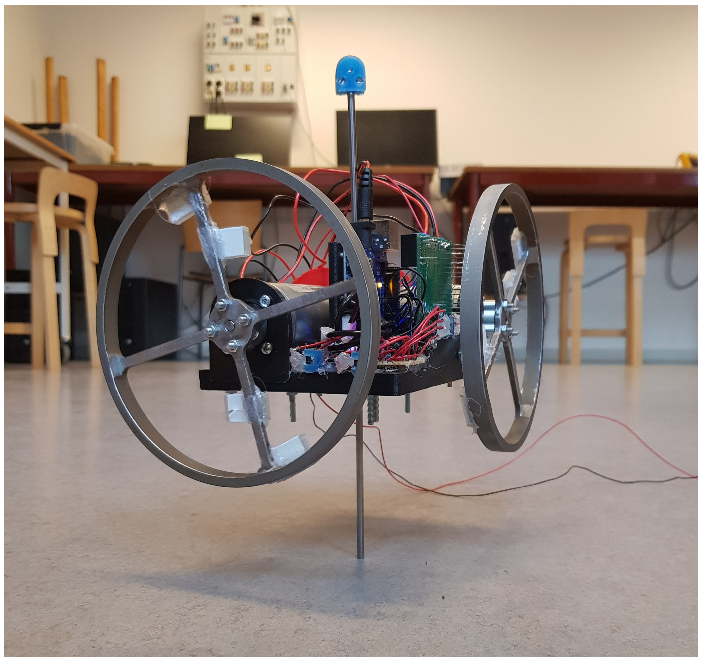


Figure 3.5. The finished demonstrator while balancing.

3.5 Code

An Arduino Uno was used as the controller unit due to its ease of use and inexpensiveness. The code was written in Arduinos IDE and then uploaded to the Arduino Uno. A simplification of the logical flow of the program is illustrated in the flowchart in figure 3.6. In the main loop the data from the IMU is updated every iteration, while the angular velocity of the reaction wheel is only recalculated and updated when the hall sensors have detected more than three magnets going in the same direction. This means the calculated angular velocity is the mean value of the angular velocity during the last three detections whenever the angular velocity is updated. Accordingly there are delays in this data and it could be more precise by adding more magnets.

The PWM signal is updated when the time T has elapsed since the last time the PWM signal was updated. When the PWM signal is updated a function is called that takes the states as input and returns the voltage that should be sent to the motor. The function basically uses equation 2.10 ($u = -Lx$) to calculate the voltage, but with the restriction of setting the voltage to 24 V if equation 2.10 gives a voltage that exceeds 24 V. It also integrates the error to form the integrator state as according to equation 2.9. Though if the voltage exceeds 24 V the input signal is saturated and no integration is done. This is refereed to as an anti windup method and it keeps the integrator from growing when the actuator is saturated and can not produce the wanted output. Anti windup prevents large overshoot and oscillations in the output signal of the system [10].

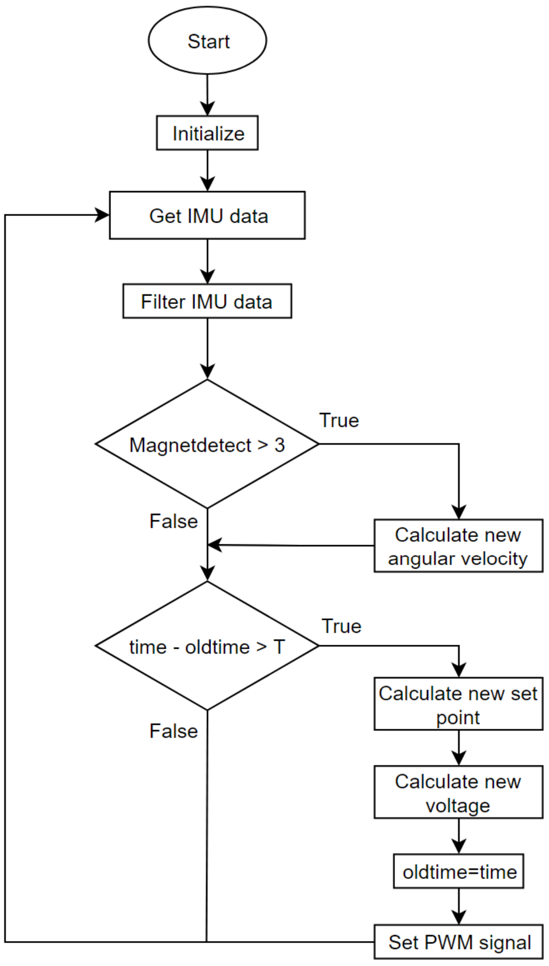


Figure 3.6. Flowchart. Drawn in draw.io.

3.6 Tuning of the controller

In the simulation several different poles where tried with corresponding feedback gain L. These L where implemented in the code. After several tests it became clear which L worked best.

3.6.1 Automatic set point

The state space controller as described in section 2.1.3 relies on that the center of mass is located at $\theta = 0$ and that the IMU is attached to the structure correctly. Since the center of mass is not exactly at $\theta = 0$ and the IMU is a not completely aligned to the line of center of mass, the data has to be compensated for this. This is done by subtracting a small angle dx to the angle read by the IMU θ_{IMU} before it goes into the controller. This means the controller will try to get $\theta_{IMU} = dx$, and therefore dx can be interpreted as the set point. dx compensates for the slanting IMU and the center of mass not being precisely above the stick. If this compensation is not exactly correct the stabilisation will fail after a short time. The system will try to regulate itself to a state where the center of mass is not above the interface between the ground and the stick. This will result in a small torque constantly pulling in one direction and the reaction wheels will constantly have to accelerate to keep up for this. The motors will build up speed until they can no longer apply any torque. This problem can be solved by changing dx over time as $dx_{n+1} = dx_n + k\frac{dx_n - \theta_{IMU}}{|dx_n - \theta_{IMU}|}$, where k is some positive gain. This means dx or the set point will increase if $dx > \theta_{IMU}$ and decrease if $dx < \theta_{IMU}$. If k is chosen so that the set point change a bit slower than θ_{IMU} it solves the problem of not knowing the correct set point. The set point will automatically go towards a good value. See figure 3.7 where the stabilisation starts at time=0.

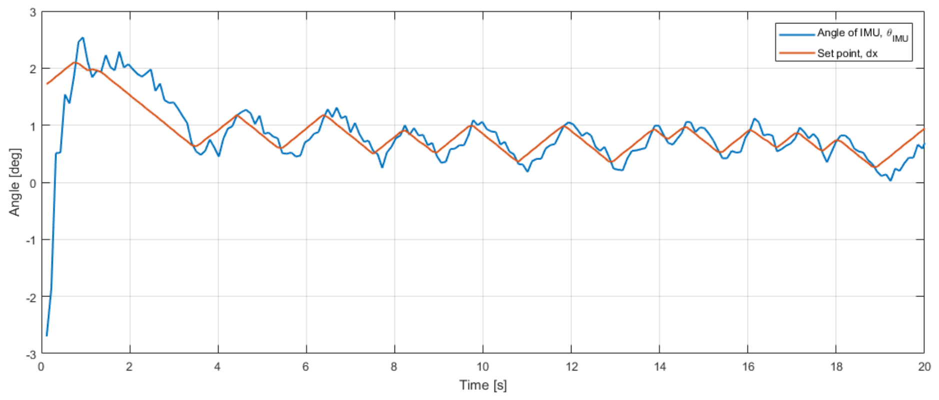


Figure 3.7. Shows how the set point moves compered to the angle while stabilising. Made in MATLAB.

A mathematical proof for that this should work could not be found. But there are some similar work where similar solutions were briefly discussed [11].

Chapter 4

Results

When implementing the state space controller on its own the system never seemed to be able to balance over time. The pendulum could stand up for a while but always fell after tens of seconds or at best after some minutes. We strongly suspect that the problem was because of the set point, which is discussed in chapter 3.6. After the implementation of the automatic set point, the system became seemingly stable. There was no sign that the system could not balance indefinitely when no disturbances were applied. Figure 4.1 shows the angle and angular velocity of the IMU while stabilising.

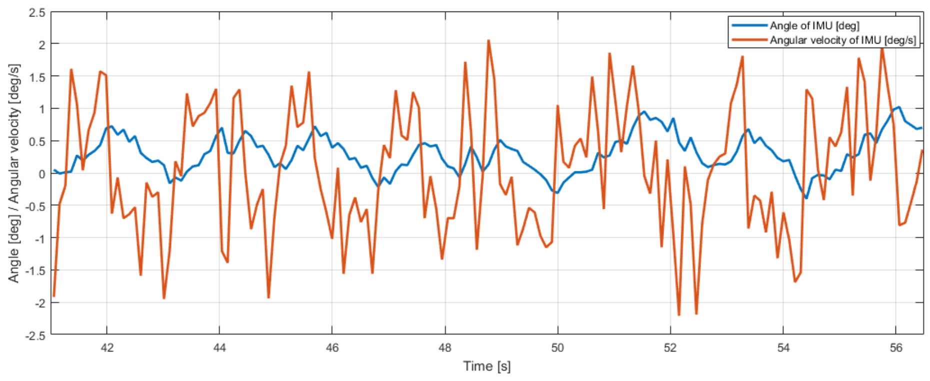


Figure 4.1. Shows the angle and angular velocity of the IMU while stabilising. Made in MATLAB.

The demonstrator could handle disturbances such as someone pushing slightly on its platform. Also the automatic set point made the system able to handle changes to the center of mass while balancing. Figure 4.2 shows the behaviour when a mass is put near one of the platforms edges. It illustrates how the system automatically finds the new angle it should stabilize around. In the end $L = [-550 \ -60 \ -0,0585 \ 1000]$, $k = 0,00003$ and $T = 0,001$ seconds were considered to give the best performance. This is while the platform was placed on a height of 10 cm. Large vibrations started to occur when the platform was placed a couple of centimetres higher.

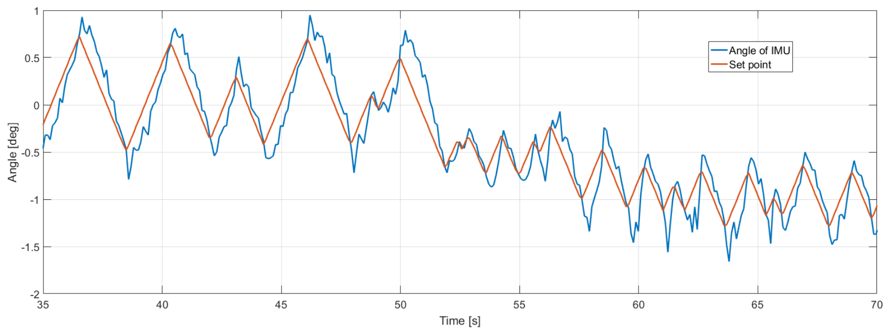


Figure 4.2. Shows how the set point and the angle moved when the center of mass was changed while stabilizing. Made in MATLAB.

Chapter 5

Discussion and conclusion

5.1 Discussion

In order to acquire data to make the graphs in this report the arduino had to send data to the computer. Since the arduino did not have wireless transmission built in nor was this added, this data transfer went through a cable. Because of the rather thick cable the system was not optimal when collecting data. The cable caused disturbances by pulling on the system. This was however not the greatest disturbance to the system while sending data. When the Arduino sent data there were delays causing the code to slow down. This made the system more unstable and caused an increased swaying movement that can be seen in figure 3.7, 4.1 and 4.2. To deal with this effect data was sent less often to the computer. It made the system more stable but reduced the resolution of the data shown in figure 3.7, 4.1 and 4.2.

The height of the platform proved to be an important factor to make the system stable. This was predicted with the mathematical model of the system. What was not as anticipated were the large vibrations that occurred when the the height of the platform was increased. The objective was to put the platform a bit higher, but it turned out that the vibrations made this very difficult. Apart from the stick being to weak, the reaction wheels were not perfectly symmetrical and this might have caused unnecessary excitement to the stick making it vibrate.

The first feedback gain L that was implemented in the micro controller was according to the simulations enough to make the system stable. It was then discovered that an L that was predicted to give a much faster system was superior. This was probably due to the delays that occurred in the real system that had not been accounted for in the simulations. Also the mathematical model was an approximation and was not expected to describe the system perfectly.

5.2 Conclusion

Several parameters such as the mass, the height of the center of mass and the moment of inertia of the reaction wheels are of importance to get the system stable. As expected it is important to implement a good controller and especially a correct set point so that the system does not try to stabilize towards a wrong angle. While methods for finding a good set point can be developed, the system is still very sensitive to changes in the center of mass and may still fail to balance after some time. In order to make the system stable over time an automatic set point was implemented as described in 3.6.1, which makes the system stable indefinitely.

Chapter 6

Recommendations and future work

6.1 Recommendations

Even if the demonstrator worked it had some flaws and there are room for improvements. There were resonance frequencies in the system when the height of the platform was increased. For future work it would be of importance to use a stiffer stick for the demonstrator. It would also be recommended to use reaction wheels that are symmetrical as to not cause any vibrations. Finally it would be recommended to use a rotary encoder to measure the angular velocity of the reaction wheels with higher precision.

6.2 Future work

Future work can for example be to investigate how high the system could be made and still be stable. One could also look at how noisy or biased state measurements effect the system. It would be interesting to investigate more profoundly how the system handles disturbances and see how robust the system is. It would also be of interest to attempt to integrate the power supply on the pendulum and make the system independent of outer facilities.

Bibliography

[1] H Johansson. *Elektroteknik*. KTH, Department of Machine Design, Mechatronic.

[2] L Glad T. Ljung. *Reglerteknik*. Studentlitteratur AB.

[3] O.J. Woodman. *An introduction to inertial navigation*. Tech. rep. University of Cambridge.

[4] C.-X. Shiau J.-K. Huang and M.-Y Chang. "Noise Characteristics of MEMS Gyro's Null Drift and Temperature Compensation." In: *Journal of applied Science and Engineering* 15.3 (2012), pp. 239–246.

[5] G Welch G. Bishop. *An Introduction to the Kalman Filter*. Tech. rep. University of North Carolina at Chapel Hill, Department of computer Science.

[6] *Pulse Width Modulation*. URL: `https://www.electronics-tutorials.ws/blog/pulse-width-modulation.html` (visited on 05/01/2019).

[7] *What is an H-Bridge?* URL: `https://www.build-electronic-circuits.com/h-bridge/` (visited on 05/01/2019).

[8] H Amlinger. *Reduction of Audible Noise of a Traction Motor at PWM Operation*. Tech. rep. KTH.

[9] K Lauszus. *Kalman Filter*. URL: `https://github.com/TKJElectronics/KalmanFilter` (visited on 05/01/2019).

[10] Erik Torstensson. *Comparison of Schemes for Windup Protection*. Tech. rep. Lund University Department of Automatic Control.

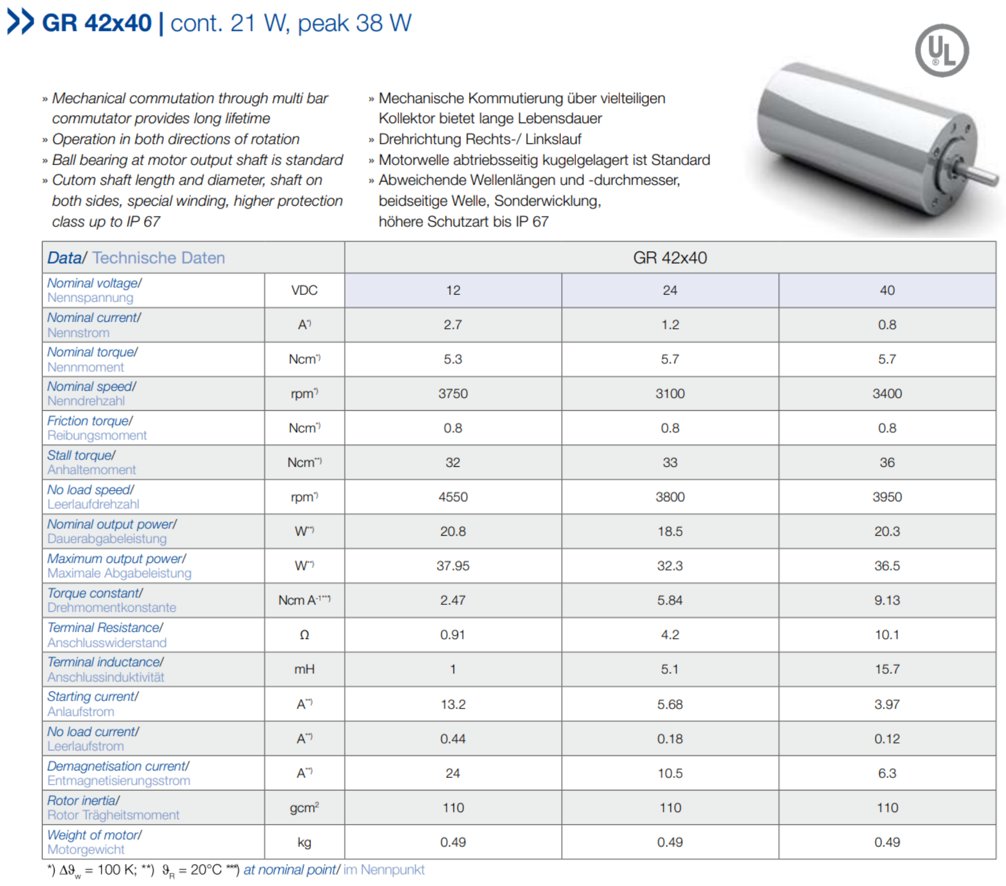
[11] P Brevik. *Two-Axis Reaction Wheel Inverted Pendulum*. Tech. rep. Norwegian University of Science and Technology.

[12] *GR 42x40*. URL: `https://www.dunkermotoren.com/uploads/tx_products/downloads/MKS/gr-42x40-8842702010.pdf` (visited on 05/07/2019).

[13] *Pololu Dual VNH5019 Motor Driver Shield for Arduino*. URL: `https://www.pololu.com/product/2507/specs` (visited on 05/07/2019).

Appendix A

Datasheets



GR 42x40 | cont. 21 W, peak 38 W

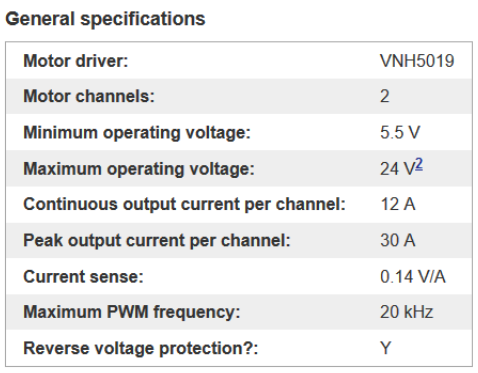
- » *Mechanical commutation through multi bar commutator provides long lifetime*
- » *Operation in both directions of rotation*
- » *Ball bearing at motor output shaft is standard*
- » *Cutom shaft length and diameter, shaft on both sides, special winding, higher protection class up to IP 67*

- » Mechanische Kommutierung über vielteiligen Kollektor bietet lange Lebensdauer
- » Drehrichtung Rechts-/ Linkslauf
- » Motorwelle abtriebsseitig kugelgelagert ist Standard
- » Abweichende Wellenlängen und -durchmesser, beidseitige Welle, Sonderwicklung, höhere Schutzart bis IP 67

Data/ Technische Daten		GR 42x40		
Nominal voltage/ Nennspannung	VDC	12	24	40
Nominal current/ Nennstrom	A*)	2.7	1.2	0.8
Nominal torque/ Nennmoment	Ncm*)	5.3	5.7	5.7
Nominal speed/ Nenndrehzahl	rpm*)	3750	3100	3400
Friction torque/ Reibungsmoment	Ncm*)	0.8	0.8	0.8
Stall torque/ Anhaltemoment	Ncm**)	32	33	36
No load speed/ Leerlaufdrehzahl	rpm*)	4550	3800	3950
Nominal output power/ Dauerabgabeleistung	W**)	20.8	18.5	20.3
Maximum output power/ Maximale Abgabeleistung	W**)	37.95	32.3	36.5
Torque constant/ Drehmomentkonstante	Ncm A^{-1}***)	2.47	5.84	9.13
Terminal Resistance/ Anschlusswiderstand	Ω	0.91	4.2	10.1
Terminal inductance/ Anschlussinduktivität	mH	1	5.1	15.7
Starting current/ Anlaufstrom	A**)	13.2	5.68	3.97
No load current/ Leerlaufstrom	A**)	0.44	0.18	0.12
Demagnetisation current/ Entmagnetisierungsstrom	A**)	24	10.5	6.3
Rotor inertia/ Rotor Trägheitsmoment	gcm^2	110	110	110
Weight of motor/ Motorgewicht	kg	0.49	0.49	0.49

*) $\Delta\vartheta_w = 100$ K; **) $\vartheta_R = 20°C$ ***) *at nominal point*/ im Nennpunkt

Figure A.1. Datasheet for the motor [12].



General specifications

Motor driver:	VNH5019
Motor channels:	2
Minimum operating voltage:	5.5 V
Maximum operating voltage:	24 V[2]
Continuous output current per channel:	12 A
Peak output current per channel:	30 A
Current sense:	0.14 V/A
Maximum PWM frequency:	20 kHz
Reverse voltage protection?:	Y

Figure A.2. General specifications for pololu H-bridge [13].

Appendix B

Schematics

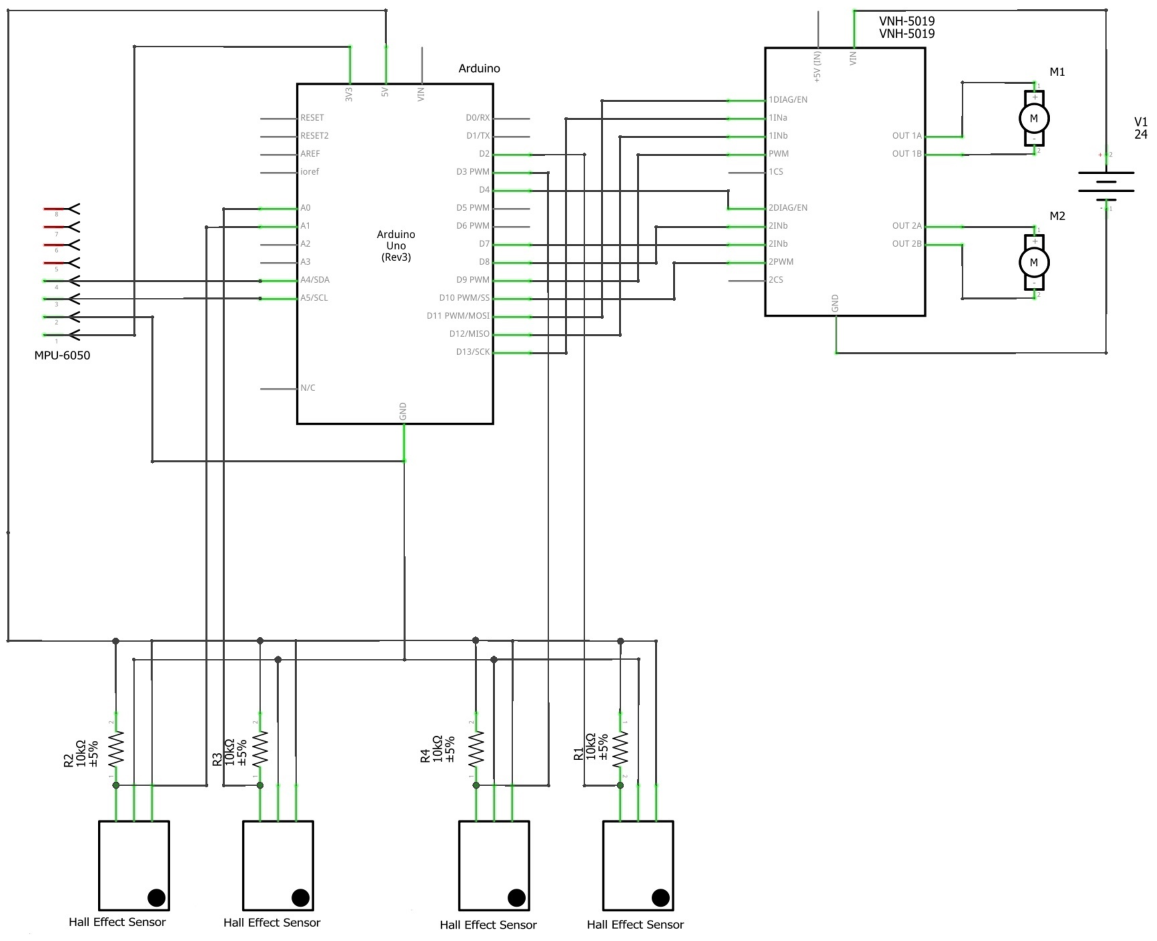


Figure B.1. Schematic of all the electrical components. Made using Fritzing.

Appendix C

Code

C.1 MATLAB code

statespace_motor_integrator.m

```
% Project name: Reaction Wheel Stabilized Stick
% Bachelor thesis in mechatronics at KTH, spring 2019
% Bill Lavebratt, Pontus Gräsberg
% Description of code:
% Matlab program for simulation of the system. Needs the function
 RK4systk.m
% It predicts how the system will behave when implementing a State
 space
% controller

clear all, clc, close all
m=1.75;                 % 1.734; % Mass of the system, from cad model
R=0.14;                 % Distance from interface between stick and
 ground to the center of mass, from cad model

I= 350/100^2;           % Moment of inertia of the system, at the
 interface between stick and ground, from Cad model
Ihh=10.196/100^2;       % Moment of inertia of the reaction wheel,
 from Cad model
Im=110/(1000*100^2);    % Moment of inertia of the rotor in motor,
 from datasheet of motor
Ih=Ihh+Im;              % Combined moment of inertia of the reaction
 wheel and motor
g=9.82;                 % Gravitational acceleration

k2fi=5.84/100;          % Torque constant of motor, from datasheet of
 motor
Ra=4.2;                 % Terminal resistance, from datasheet of motor

% State space model
A=[0         1   0;
   m*g*R/I   0   k2fi^2/(I*Ra);
   0         0  -k2fi^2/(Ih*Ra)];

B=[0; -k2fi/(I*Ra); k2fi/(Ih*Ra)];

C=[1 0 0];

% State space model when imlpementing the integrator
Any=[A zeros(size(A,1),1);-C 0];
Bny=[B;0];
Cny=[C 0];
ref=0; % Set point
D=[zeros(size(A,1),1);1];

Lny = place(Any, Bny, [-2,0,-1+0.5i -1-0.5i]); % Gives the feedback
 gain constants in L as a function of the poles of the system.

Amax=[A zeros(size(A,1),1); zeros(1,4)]; % Is used when desired
 voltage is larger than the maximum allowed.
```

```
Umax=24; % Maximum voltage

% System of differential equations of the system with controller and
 anti windup, accounts for Umax.
f=@(tvec,yvec) ((((Any-Bny*Lny)*yvec' +ref*D)*(abs(-
yvec(end,:)*Lny')<=Umax))+(Amax*yvec'+Bny*Umax*abs(-yvec(end,:)*Lny')/
(-yvec(end,:)*Lny'))*(abs(-yvec(end,:)*Lny')>Umax)')';

y0=[0.125 0 0 0]; % Initial conditions of the states [Agle, Angular
 velocity, Angular velocity of recation wheel, Integraion of the
 error]
tspan=[0 10];    % Simulation spans over this time.
h=0.001;         % Step length in ODE solver.
[tv,yv]=RK4systk(f,tspan,y0,h); % Solution of differential equation
 with implementation of RK4, See code "RK4systk.m "
figure
plot(tv,yv(:,1))
grid on, title('Angle of pendulum \theta'), xlabel('t [s]'),
 ylabel('rad');

figure
plot(tv,yv(:,2))
grid on, title('Angular velocity of pendulum'), xlabel('t [s]'),
 ylabel('rad/s');

figure
plot(tv,yv(:,4))
grid on, title('Integral(ref-angle)'), xlabel('t [s]');

figure
plot(tv,yv(:,3))
grid on, title('Angular velocity of reaction wheel'), xlabel('t [s]'),
 ylabel('rad/s');

% The voltage that was sent to the motor
U=-yv*Lny';
for ii=1:length(U)
    if abs(U(ii))>Umax;

        U(ii)=Umax*U(ii)/abs(U(ii));
    end
end

M=@(w,Ua) Ua*k2fi/Ra-(k2fi)^2/Ra*w; % Equation of the motor
Mmotor=M(yv(:,3),U); % The torque the motor gave

figure
plot(tv,U)
grid on, title('Voltage'), xlabel('t [s]'), ylabel('V');


figure
plot(tv,Mmotor)
grid on, title('Torque from motor'), xlabel('t [s]'), ylabel('Nm');
```

RK4systk.m

```
% Project name: Reaction Wheel Stabilized Stick
% Bachelor thesis in mechatronics at KTH, spring 2019
% Bill Lavebratt, Pontus Gräsberg
% Description of code:
% Matlab function for simulation of the system.
% Implementation of a Runge-Kutta 4th Order method for System of
 differential equations.
% The input is f: function of system of differential equations, tspan:
% start time and end time of simulation, y0: Initial conditions, h:
% steplength.
% The output is tv: the time vector, yv: matrix of the solution to the
% system of differential equations.

function [tv,yv]=RK4systk(f,tspan,y0,h)

    n=(tspan(2)-tspan(1))/h;
    tv=(tspan(1)+h*(0:n))';
    p=length(y0);
    yv=zeros(n+1,p);
    yv(1,:)=y0;

    for ii=1:n
        k1=feval(f,tv(ii),yv(ii,:));
        k2=feval(f,tv(ii)+h/2,yv(ii,:)+h/2*k1);
        k3=feval(f,tv(ii)+h/2,yv(ii,:)+h/2*k2);
        k4=feval(f,tv(ii)+h,yv(ii,:)+h*k3);
        yv(ii+1,:)=yv(ii,:)+h/6*(k1+2*k2+2*k3+k4);
    end
end
```

Published with MATLAB® R2017b

C.2 Arduino code

```
/*
 * University:          Royal Institute of Technology, KTH
 * TRITA number:        TRITA-ITM-EX 2019:38
 * Authors:             Bill Lavebratt and Pontus Gr sberg
 * Name of project:     Reaction wheel stabalized stick
 *
 * Description of code:
 * Main program used to balance the reaction wheel stabalized stick.
 * Filtered state measurements goes into a controler/regulator function
    that calculates the desired voltage to the motors.
 * Then the coresponding PWM signals is sent to an H-bridge.
 */

#include <PWM.h> // library for change in pwm freq
//Source: https://github.com/RCS101/PWM

#include <filters.h>    //Library for Butterworth filter
#include <filters_defs.h>
//Source: https://github.com/MartinBloedorn/libFilter

#include <Wire.h>
#include <Kalman.h> // Source: https://github.com/TKJElectronics/
    KalmanFilter

#define RESTRICT_PITCH // Comment out to restrict roll to  90deg
    instead

Kalman kalmanX; // Create the Kalman instances
Kalman kalmanY;

float BY2;  // Create varible for Butterworth filter of gyrorate
float BX2;

int i;
int ii;
float dx1;
float dy1;

const float cutoff_freq   = 5.0;  //Cutoff frequency in Hz for
    Butterworth filter
const float sampling_time = 0.0055; //Sampling time in seconds.
IIR::ORDER  order  = IIR::ORDER::OD3; // Order (OD1 to OD4)

// Low-pass filter
Filter f(cutoff_freq, sampling_time, order); //Function for Butterworth
     filter
Filter g(cutoff_freq, sampling_time, order);

float gyroXrate3; // Varible for gyrorate given by MPU
float gyroYrate3;
float dx=0.03;    //Starting setpoint
float dy=0.006;

/* IMU Data */
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int16_t tempRaw;

double gyroXangle, gyroYangle; // Angle calculate using the gyro only
double kalAngleX, kalAngleY; // Calculated angle using a Kalman filter

uint32_t timer;
uint8_t i2cData[14]; // Buffer for I2C data

// Varibles fo controller
float L1=-550; // State space feedback gain
float L2=-60;
float L3=-0.0585;
float L4=1000;
float v_ref=0;
float DT=0.001; //Time period between the updates of PWM signal (
    Helpful for integrator part)
unsigned long oldTime=0;

  //State variables for direction 1
  float v_vink1;
  float v_hast1;
  float radps1;
  float x41=0;

  //State variables for direction 2
  float v_vink2;
  float v_hast2;
  float radps2;
  float x42=0;

  // varible for controller
  float Umax=24;  // Maximum voltage
  float U1;
  float U2;


//Pins for H-bridge
#define M1PWM 9
#define M1INA 13
#define M1INB 12
#define M1CS A0
int M1EN = 11;


#define M2PWM 10
#define M2INA 8
#define M2INB 7
```

```
int M2EN = 4;


int32_t freq =20000; //PWM freq

float pwmOutput1=0;  // PWM signal varibles
float pwmOutput2=0;

float half_revolutions1;  // Magnet detecting counters
float half_revolutions2;

//Variables for Hall sensor
unsigned long DTH1;
unsigned long DTH2;
unsigned long timeold1;
unsigned long timeold2;
int interruptPin11=2;
int interruptPin21=3;
int drPin12=A0;
int drPin22=A1;


void setup() {
InitTimersSafe();
SetPinFrequencySafe(M1PWM, freq);

  // For motor
  pinMode(M1PWM, OUTPUT);
  pinMode(M1INA, OUTPUT);
  pinMode(M1INB, OUTPUT);
  pinMode(M1EN, OUTPUT);
  pinMode(M2PWM, OUTPUT);
  pinMode(M2INA, OUTPUT);
  pinMode(M2INB, OUTPUT);
  pinMode(M2EN, OUTPUT);


  // Set initial rotation direction
  digitalWrite(M1INA, HIGH);
  digitalWrite(M1INB, LOW);
  digitalWrite(M1EN, HIGH);
  digitalWrite(M2INA, HIGH);
  digitalWrite(M2INB, LOW);
  digitalWrite(M2EN, HIGH);

   //For Hall sensor
   //Initialize the intterrupt pin (Arduino digital pin 2)
   attachInterrupt(digitalPinToInterrupt(interruptPin11),
    magnet_detect1, FALLING);
   //Initialize the intterrupt pin (Arduino digital pin 3)
   attachInterrupt(digitalPinToInterrupt(interruptPin21),
    magnet_detect2, FALLING);
   pinMode(drPin12, INPUT);
   pinMode(drPin22, INPUT);
  Serial.begin(115200);
  Wire.begin();
  #if ARDUINO >= 157
    Wire.setClock(400000UL); // Set I2C frequency to 400kHz
  #else
    TWBR = ((F_CPU / 400000UL) - 16) / 2; // Set I2C frequency to 400
    kHz
  #endif

  i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000
    Hz
  i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256
     Hz Gyro filtering, 8 KHz sampling
  i2cData[2] = 0x00; // Set Gyro Full Scale Range to  250deg/s
  i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to  2g
  while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four
    registers at once
  while (i2cWrite(0x6B, 0x01, true)); // PLL with X axis gyroscope
    reference and disable sleep mode
  while (i2cRead(0x75, i2cData, 1));
  if (i2cData[0] != 114) { // Read "WHO_AM_I" register
    Serial.print(F("Error reading sensor"));
    while (1);
  }

  delay(100); // Wait for sensor to stabilize

  /* Set kalman and gyro starting angle */
  while (i2cRead(0x3B, i2cData, 6));
  accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
  accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
  accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);

  // Source: http://www.freescale.com/files/sensors/doc/app_note/AN3461
    .pdf eq. 25 and eq. 26
  // atan2 outputs the value of -  to  (radians) - see http://en.
    wikipedia.org/wiki/Atan2
  // It is then converted from radians to degrees
#ifdef RESTRICT_PITCH // Eq. 25 and 26
  double roll  = atan2(accY, accZ) * RAD_TO_DEG;
  double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) *
    RAD_TO_DEG;
#else // Eq. 28 and 29
  double roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) *
    RAD_TO_DEG;
  double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
#endif

  kalmanX.setAngle(roll); // Set starting angle
  kalmanY.setAngle(pitch);
  gyroXangle = roll;
  gyroYangle = pitch;
  timer = micros();

}


//MAIN LOOP
void loop() {

MPU_Kalman(); // Read data given by Kalman filter

  // Calculates angular velocity [rad/s] for Hall effect sensors
  if (abs(half_revolutions1)>3) {
     DTH1 = (micros() - timeold1);
     radps1 = -3.1416*1000000*(half_revolutions1/DTH1)/2.0;
     half_revolutions1 = 0;
     timeold1 = micros();

   }
  if (abs(half_revolutions2)>3) {
     DTH2 = (micros() - timeold2);
     radps2 = -3.1416*1000000*(half_revolutions2/DTH2)/2.0;
     half_revolutions2 = 0;
     timeold2 = micros();

   }

  //Calculates new voltage when the time T has elapsed since last time
    the voltage was calculated
  if ((micros()-oldTime)>(DT*1000000)){

    dx1=(-v_vink1)/abs(v_vink1);    // Change set point
    dx=dx+0.00003*dx1;

    dy1=(-v_vink2)/abs(v_vink2);
    dy=dy+0.00003*dy1;

    //Calculate new voltage with the controller/regulator function that
     takes the states and returns the desired voltage.
    U1=regulator(v_vink1,v_hast1,radps1,&x41);   // Voltage for motor1
    U2=regulator(v_vink2,v_hast2,radps2,&x42);   // Voltage for motor2
    oldTime=micros();


      // Set voltage to zero if some x or y angle is more than 20
    degrees,
      //since the system canot be stabilized in that case.
    if (abs(v_vink1*180.0/3.1415)>20 || abs(v_vink2*180.0/3.1415)>20){
        U1=0;
        U2=0;
    }

    // Sens desired voltage to functions that sets the PWM signal
    corresponding to the desired voltage
    motor1pwm(U1);
    motor2pwm(U2);

  }
}

//FUNKCTIONS BELOW

 // Regulator function that takes the states and returns the voltage
  float regulator(float v_vink,float v_hast,float s_hast,float *x4){
    float temp=*x4;
    (*x4)=(*x4)+(v_ref-v_vink)*DT; //integrator part of controller
    float UU;

    UU=-(L1*v_vink+L2*v_hast+L3*s_hast+L4*(*x4));
      // When the desired voltage is to large
      if(abs(UU)>Umax){
        UU=Umax*abs(UU)/UU;
        (*x4)=temp; // anti windup
        }

    return UU;
    }

// Function that sets the PWM signal corresponding to the desired
    voltage
void motor2pwm(float spanning1){

    if(spanning1>=0){                 //Decide the direction of the
    current
      digitalWrite(M1INA, HIGH);
      digitalWrite(M1INB, LOW);
      digitalWrite(M1EN, HIGH);
    }
    else{
      digitalWrite(M1INA, LOW);
      digitalWrite(M1INB, HIGH);
      digitalWrite(M1EN, HIGH);
      spanning1=abs(spanning1);
    }
 pwmOutput1 = map(spanning1, 0, Umax, 0 , 255); // Map the
    potentiometer value from 0 to 255
 pwmWrite(M1PWM,pwmOutput1);
}

// Function that sets the PWM signal corresponding to the desired
    voltage
void motor1pwm(float spanning2){

    if(spanning2>=0){               //Decide the direction of the
    current
      digitalWrite(M2INA, HIGH);
      digitalWrite(M2INB, LOW);
      digitalWrite(M2EN, HIGH);
    }
    else{
      digitalWrite(M2INA, LOW);
      digitalWrite(M2INB, HIGH);
      digitalWrite(M2EN, HIGH);
      spanning2=abs(spanning2);
    }
 pwmOutput2 = map(spanning2, 0, Umax, 0 , 255); // Map the
    potentiometer value from 0 to 255
 pwmWrite(M2PWM,pwmOutput2);
}


//This function is called whenever a magnet/interrupt is detected by
    Hall sensor1, counts the detections of magnets
 void magnet_detect1()
 {
  if(digitalRead(drPin12)==LOW){
    half_revolutions1++;
  }
  else {
    half_revolutions1--;
  }
 }

//This function is called whenever a magnet/interrupt is detected by
    Hall sensor2, counts the detections of magnets
  void magnet_detect2()
 {
  if(digitalRead(drPin22)==LOW){
    half_revolutions2++;
  }
  else {
    half_revolutions2--;
  }
 }

//Kalman fuction made by Lauszus
void MPU_Kalman() {
  /* Update all the values */
  while (i2cRead(0x3B, i2cData, 14));
  accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
  accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
  accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);
  tempRaw = (int16_t)((i2cData[6] << 8) | i2cData[7]);
  gyroX = (int16_t)((i2cData[8] << 8) | i2cData[9]);
  gyroY = (int16_t)((i2cData[10] << 8) | i2cData[11]);
  gyroZ = (int16_t)((i2cData[12] << 8) | i2cData[13]);;

  double dt = (double)(micros() - timer) / 1000000; // Calculate delta
    time
  timer = micros();

  // Source: http://www.freescale.com/files/sensors/doc/app_note/AN3461
    .pdf eq. 25 and eq. 26
  // atan2 outputs the value of −   to    (radians) − see http://en.
    wikipedia.org/wiki/Atan2
  // It is then converted from radians to degrees
  #ifdef RESTRICT_PITCH // Eq. 25 and 26
    double roll  = atan2(accY, accZ) * RAD_TO_DEG;
    double pitch = atan(−accX / sqrt(accY * accY + accZ * accZ)) *
   RAD_TO_DEG;
  #else // Eq. 28 and 29
    double roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) *
   RAD_TO_DEG;
    double pitch = atan2(−accX, accZ) * RAD_TO_DEG;
  #endif

  double gyroXrate = gyroX / 131.0; // Convert to deg/s
  double gyroYrate = gyroY / 131.0; // Convert to deg/s

#ifdef RESTRICT_PITCH
  // This fixes the transition problem when the accelerometer angle
    jumps between −180 and 180 degrees
  if ((roll < −90 && kalAngleX > 90) || (roll > 90 && kalAngleX < −90))
     {
    kalmanX.setAngle(roll);
    kalAngleX = roll;
    gyroXangle = roll;
  } else
    kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the
     angle using a Kalman filter

  if (abs(kalAngleX) > 90)
    gyroYrate = −gyroYrate; // Invert rate, so it fits the restriced
    accelerometer reading
  kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
#else
  // This fixes the transition problem when the accelerometer angle
    jumps between −180 and 180 degrees
  if ((pitch < −90 && kalAngleY > 90) || (pitch > 90 && kalAngleY <
    −90)) {
    kalmanY.setAngle(pitch);
    kalAngleY = pitch;
    gyroYangle = pitch;
  } else
    kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); // Calculate
    the angle using a Kalman filter

  if (abs(kalAngleY) > 90)
    gyroXrate = −gyroXrate; // Invert rate, so it fits the restriced
    accelerometer reading
  kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the
    angle using a Kalman filter
#endif

  gyroXangle += gyroXrate * dt; // Calculate gyro angle without any
    filter
  gyroYangle += gyroYrate * dt;
  gyroXrate3 = kalmanX.getRate(); // Calculate gyro angle using the
    unbiased rate
  gyroYrate3 = kalmanY.getRate();

  // Reset the gyro angle when it has drifted too much
  if (gyroXangle < -180 || gyroXangle > 180)
    gyroXangle = kalAngleX;
  if (gyroYangle < -180 || gyroYangle > 180)
    gyroYangle = kalAngleY;

float filterFrequency = 1000.0;
BX2 = f.filterIn(gyroXrate3);  // Butterworth filtering
BY2 = g.filterIn(gyroYrate3);

v_vink1=3.1415/180.0*kalAngleX-dx; //convert to Radians
v_vink2=3.1415/180.0*kalAngleY-dy;

v_hast1=3.1415/180.0*BX2;
v_hast2=3.1415/180.0*BY2;

}
```

```
/* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights
    reserved.

 This software may be distributed and modified under the terms of the
    GNU
 General Public License version 2 (GPL2) as published by the Free
    Software
 Foundation and appearing in the file GPL2.TXT included in the
    packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works
    based
 on this software must also be made publicly available under the terms
    of
 the GPL2 ("Copyleft").

 Contact information
 -------------------
 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
 */

const uint8_t IMUAddress = 0x68; // AD0 is logic low on the PCB
const uint16_t I2C_TIMEOUT = 1000; // Used to check for errors in I2C
    communication

uint8_t i2cWrite(uint8_t registerAddress, uint8_t data, bool sendStop)
    {
```

```
  return i2cWrite(registerAddress, &data, 1, sendStop); // Returns 0 on
     success
}

uint8_t i2cWrite(uint8_t registerAddress, uint8_t *data, uint8_t length
    , bool sendStop) {
  Wire.beginTransmission(IMUAddress);
  Wire.write(registerAddress);
  Wire.write(data, length);
  uint8_t rcode = Wire.endTransmission(sendStop); // Returns 0 on
    success
  if (rcode) {
    Serial.print(F("i2cWrite failed: "));
    Serial.println(rcode);
  }
  return rcode; // See: http://arduino.cc/en/Reference/
    WireEndTransmission
}
uint8_t i2cRead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes)
     {
  uint32_t timeOutTimer;
  Wire.beginTransmission(IMUAddress);
  Wire.write(registerAddress);
  uint8_t rcode = Wire.endTransmission(false); // Don't release the bus
  if (rcode) {
    Serial.print(F("i2cRead failed: "));
    Serial.println(rcode);
    return rcode; // See: http://arduino.cc/en/Reference/
    WireEndTransmission
  }
  Wire.requestFrom(IMUAddress, nbytes, (uint8_t)true); // Send a
    repeated start and then release the bus after reading
  for (uint8_t i = 0; i < nbytes; i++) {
    if (Wire.available())
      data[i] = Wire.read();
    else {
      timeOutTimer = micros();
      while (((micros() - timeOutTimer) < I2C_TIMEOUT) && !Wire.
    available());
      if (Wire.available())
        data[i] = Wire.read();
      else {
        Serial.println(F("i2cRead timeout"));
        return 5; // This error value is not already taken by
    endTransmission
      }
    }
  }
  return 0; // Success
}
```

TRITA TRITA-ITM-EX 2019:38