DEGREE PROJECT IN MECHANICAL ENGINEERING,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2021

Balancing a Monowheel with a PID controller
Balansering av ett Monowheel med hjälp av en PID regulator

FRITIOF ANDERSEN EKVALL

NILS WINNERHOLT

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ARCHITECTURE AND THE BUILT ENVIRONMENT

Balancing a monowheel with a PID controller

FRITIOF ANDERSEN EKVALL
NILS WINNERHOLT

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

TRITA-ITM-EX 2021:49

Abstract

This bachelor's thesis aimed to create a self balancing monowheel, a vehicle type consisting of one wheel, using a PID controller. The wheel is equipped with an accelerometer to gather data about the tilt of the construction, which is then filtered using a Kalman filter. A DC motor is propelling the monowheel forward whereas a stepper motor with a battery pack attached will actively balance the wheel with the help of a PID-controller. This method of balancing had limited success allowing the vehicle to travel for up to 7 seconds before falling over, compared with up to 4 seconds with no balancing implemented.

Keywords: Mechatronics, Monowheel, Balance, PID & Arduino

Referat

Balansera ett monowheel med en PID kontroller

Detta kandidatexamensarbete har målet att skapa ett självbalanserande hjul med hjälp av en PID kontroller tillsammans med en acceleromter vars utsignal filtreras med ett kalman filter. En DC motor driver hjulet frammåt medan en stepper motor med ett batteripack fastsatt är den rörliga vikten som balanserar konstruktionen. PID metoden lyckades balansera upp till 7 sekunder vilket är en marginell ökning jämfört med upp till 4 sekunder helt utan aktiv balansering.

Nyckelord: Mekatronik, Monowheel, Balans, PID & Arduino

Acknowledgements

We would like to thank all our classmates who have all been very helpful. We would also like to offer a special thanks to Rayan Alnakar for his help and encouraging words.

Abbreviations

CAD Computer Aided Design

DC Direct Current

IDE Integrated Development Environment

PID Proportional Integral Derivative

PWM Pulse Width Modulation

USB Universal Serial Bus

CoG Center of Gravity

KF Kalman filter

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

A monowheel is a vehicle with only a single wheel that has all of its driving action taking place inside of the wheel. Since the construction only has one point of contact to the ground it suffers from instability if it starts to tilt. Thus a form of stabilization is often implemented where the most common stabilization method is a shifting of the wheels' center of gravity[1]. A classical monowheel has a person sitting inside the machine balancing it. This version will use a method for stabilizing using a shiftable weight to alter the center of gravity. Another possible method of stabilization is control moment gyroscope which uses a gyroscope paired with motors and flywheel, as was done in a previous bachelor's project by Arthur Grönlund and Christos Tolis[2]. The challenge of creating a method of stabilization for such a vehicle has enticed us to start this project.

1.2 Purpose

The purpose of this project is to create a stable monowheel using a weight shifting stabilization method. The project aims to answer the research question:

"How well can a monowheel be stabilized using an Arduino microcontroller coupled with a shiftable weight using PID regulation?"

1.3 Scope

This project will focus on keeping the monowheel stabilized running in a straight line forwards and backwards using a PID controller. How to get the monowheel to turn is an interesting aspect of further investigation that is beyond the scope of this project.

1.4 Method

A prototype will be constructed using a 3D printer. One battery pack will be the shiftable weight and another will be placed as low as possible for increased stability and balancing the weight and the DC motor. Utilizing the battery packs like this saves weight as no additional material is needed to be added for the purpose as ballast. The shifting of the battery pack will be done by a stepper motor which the battery pack is mounted to and moves perpendicular to the forward motion of the monowheel. An accelerometer and gyroscope will sense the current angle of tilt and then send that information to an Arduino that will control the positioning of the shiftable weight.

To assess where and how to place the motors and electronic controllers as efficiently as possible a CAD drawing will be made of the chosen prototype and then different placing schemes for the component mounts will be tested inside of the CAD software. This will allow us to try out multiple geometries at no extra cost.

Chapter 2

Theory

2.1 Arduino

Arduino is a brand of open source microcontrollers, which are small computers made on a single board. Arduino comes in many different models with their own best areas of implementation. In this project an Playknowlogy Uno Rev.3 Arduino compatible developement board is used, which can bee seen in figure 2.1. The microcontroller is attached to a board that has a series of both digital and analog input/output pins. Electrical components such as sensors and motors can be connected to those pins and controlled using the programmable microcontroller. It is programmed by writing code in the Arduino IDE v1.8.13 and then transferring that code onto the microcontroller using a USB cable[3].

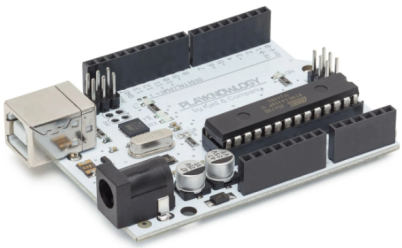

Figure 2.1. An Arduino Uno compatible developement board [14]

2.2 Gyroscope and accelerometer

An acceleromemeter is a sensor that detects its own acceleration in space. A gyroscope measures its own rotation and angular velocity. With these two types of sensors an accurate representation of an objects orientation in space can be con-

structed. The MPU6050 pictured in figure 2.2 is a sensor that combines a 3 axis accelerometer and a 3 axis gyroscope into one part. The gyroscope measure the angular rate by measuring the displacement of a mass inside of the sensor due to the Coriolis effect. The mass will move, causing the capacitance between the mass and fixed plates to change. The accelerometer also work by measuring the capacitance between a mass and a fixed plate, but does it by mounting the mass on springs that allow the mass to be offset from its steady state when it is subjected to an acceleration[4].

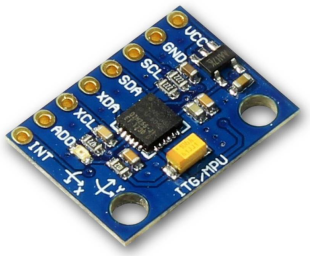

Figure 2.2. MPU6050: gyroscope and accelerometer chip [15]

2.3 DC and stepper motor

A DC motor has three main parts: rotor, stator and commutator. It uses direct current to induce an electromagnetic force through the commutator to the rotor located within the stator which translates to torque and mechanical energy. It can be adjusted through altering the voltage supplied to the motor. The relation follows the equation:

$$U_A = R_A I_A + k_2 \Phi \omega \tag{2.1}$$

with U_A as the voltage, R_A is the internal resistance of the motor, I_A is the current going over it, $k_2\Phi$ is a motor specific constant and ω is the angular velocity[5].

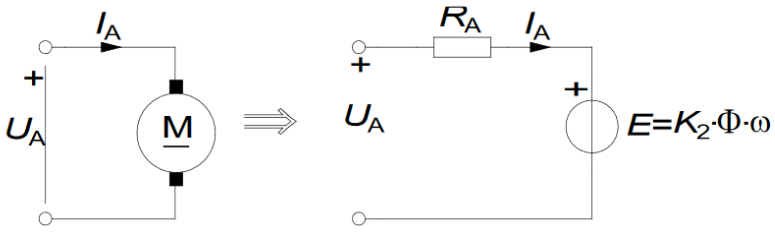

Figure 2.3. Circuit diagram of DC motor with a power source [5]

A stepper motor has a similar construction to that of a DC motor but it's method of control is different. The stepper motor rotates in steps with one or a pair

of electromagnets powering on for each step[6]. It is possible to change the amount of steps to make smaller individual movements with microstepping, up to 6400 from the regular 200[16].

2.4 H-Bridge

The H-Bridge is an electrical circuit switches polarity of the voltage for a DC motor, thus allowing for control of the direction it spins. It does this with the help of four transistors that function like power switches. When transistor S1 and S4 in figure 2.4 are activated while S2 and S3 are deactivated the current flows into the positive pole of the motor and out of the negative pole, thus making the motor start spinning. To make the motor spin the other way S2 and S3 are activated while S1 and S4 are deactivated[5].

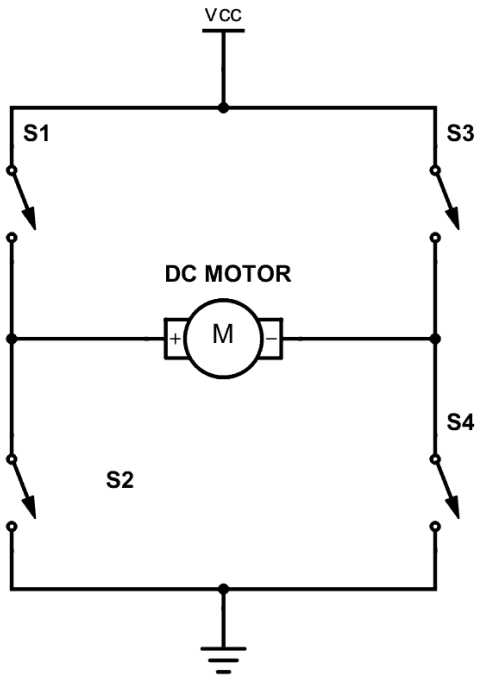

Figure 2.4. Circuit diagram of H-bridge [13]

2.5 PID

The PID controller is a mechanism utilized in closed feedback loops in order to get the desired output y from a system. This is done by comparing the current output of the system with a reference signal r and then taking the difference between them

to get the error e.

$$e(t) = r(t) - y(t) \tag{2.2}$$

Using the error, the input signal to the system in the following way:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{d}{dt}e(t) \tag{2.3}$$

The equation for the input signal has three different coefficients that correspond to different properties of the controller. K_P is the proportional gain and improves the rise time of the system and decreases the steady state error, but does not eliminate it, introduces overshoot to the system and causes instability if increased to much. K_I is the integrated gain and eliminates the steady state error and improves the rise time further, but adds additional overshoot and instability at high values. The overshoot is corrected for with the last parameter K_D is the derivative gain and also decreases instability. Fine tuning of the coefficients allows the system to get the desired properties[7] .

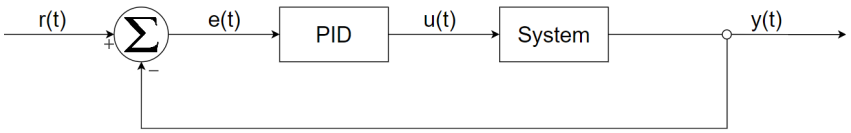

Figure 2.5. Block diagram of an implementation of a PID controller. Drawn in Paint

2.6 Kalman filter

Kalman filter, also called linear quadratic estimation is a recursive algorithm. Here it will be used for reducing the noise from the sensors[8]. To use a Kalman filter, first the Kalman gain will be calculated with E_{EST} as the error (uncertainty) of the estimate and E_{MEA} is the error in the measurment :

$$KG = \frac{E_{EST}}{E_{EST} + E_{MEA}} \tag{2.4}$$

The estimate error is calculated with:

$$E_{EST} = (1 - KG)E_{EST_{prev}} \tag{2.5}$$

The estimation EST updates regarding the previous estimate EST_{prev} and the measurment MEA through the Kalman gain KG:

$$EST = EST_{prev} + KG(MEA - EST_{prev}) \tag{2.6}$$

As the estimate error decreases the Kalman gain will decrease allowing the the estimate to depend more on the previous estimate and less on the measurement[9].

Chapter 3

Demonstrator

3.1 Problem formulation

The demonstrator needed to fulfill a number of requirements in order to function in the desired way. Those functions are:

1. Powertrain that transfers the rotation of a DC-motor to the outer wheel
2. Accelerometer that detects the deviation from upright
3. Balancing mechanism that adjusts the CoG of the monowheel

To test and improve the theory, a prototype will be constructed. The shell is modelled in CAD and 3D printed. Two motors, their drivers, sensors and the Arduino mounted inside shown below in figure 3.1 and figure 3.2.

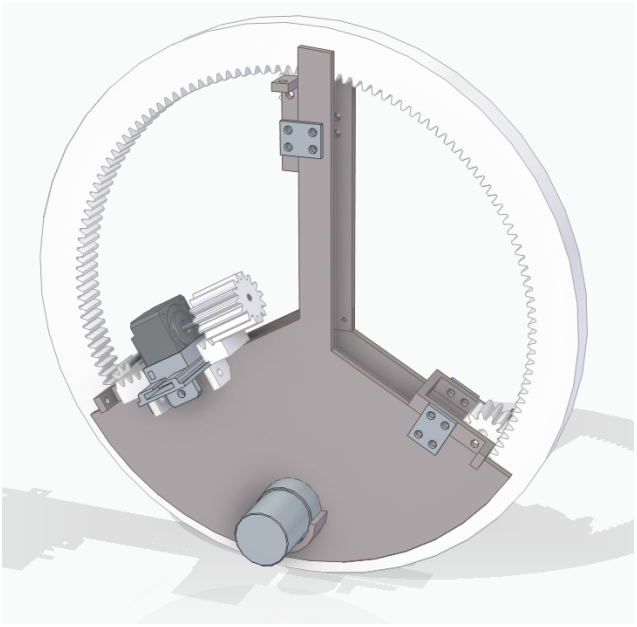

Figure 3.1. CAD model of the construction. Made in SolidEdge 2020 University Edition

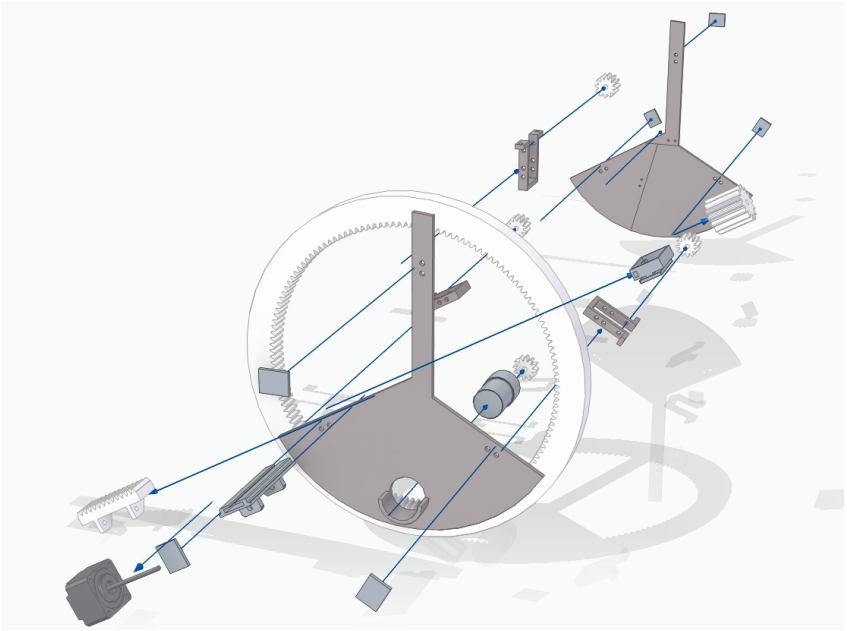

Figure 3.2. Exploded view of the CAD model. Made in SolidEdge 2020 University Edition

3.2 Powertrain

The powertrain consists of an outer gear with four inner gears. One of the inner gears is driven by the DC motor sitting on the side of figure 3.1.

3.3 Simulation

To verify the wiring it was first drawn up and tested in Tinkercad and fritzing as seen in figure 3.3 before being physically connected. First for the DC motor and afterwards for the stepper motor and last for the gyroscope and accelerometer sensor.

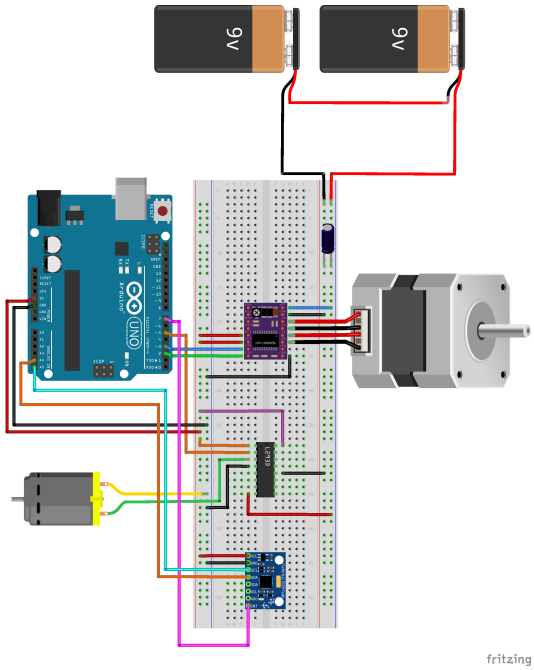

Figure 3.3. Wiring diagram of the electronics for the monowheel. Made in fritzing 0.9.3

3.4 Component selection

A 24V DC motor is coupled with a stepper motor with a driver accepting 8-45V making it possible to run the same voltage for them both. This will be achieved with 9V batteries connected both in parallel and in series for 18V and extra capacity.

3.5 Software

The tuning of the PID parameters has been done with the help of PIDtuner[17], a website that allows for the optimization of parameters when supplied with data from the use of a controller. The parameters were fine tuned by repeated use of the software.

Arduino IDE was used to program the Arduino Uno microcontroller. In the code a set of libraries were utilized to aid in the programs simplicity. The inbuilt Wire.h library allows for communication with I2C and TWI devices. The PIDv2.h library, which was created by Brett Beauregard, creates a PID-controller using parameters that the users chooses[18].

Chapter 4

Results

The prototype was constructed using 3D-printed parts held together with screws and reinforced with tape at connection points. Some parts were too large to print in one piece so were split into modules that could fit onto the printing surface. The final PID tuning parameters used are shown below in table 4.1.

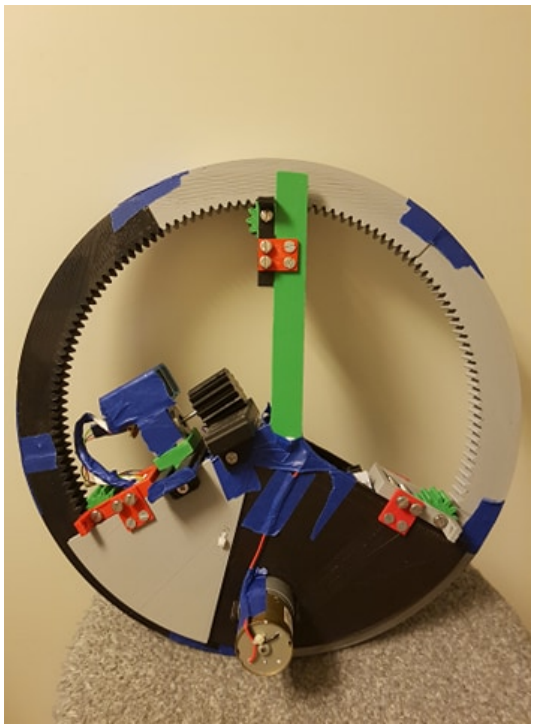

Figure 4.1. The constructed prototype. Photo taken by Nils Winnerholt

Table 4.1. Table of the PID paramters used

K_P	0.7764
K_I	-0.004131
K_D	-72.97

The balance of the monowheel was tested while rolling forward by timing how long it could roll forward on a slightly uneven surface. The test was performed using different balancing methods to evaluate the dependency of the balance method for the performance of the monowheel. The tested balancing methods were PID-controller+Kalman filter, PID-controller and P-controller. The test was also performed with no active balancing as a negative control.
As seen in figure 4.1, with no active balancing the monowheel where balanced for approximately four seconds. The result was similar for all the three tries. The longest balancing period was obtained when a PID-controller was used to balanced the monowheel and was approximately 7 seconds. The shortest balancing period was also obtained when using a PID-controller. When using a PID-controller together with a Kalman filter, the monowheel was balanced for approximately 4.5 seconds. As summarized in table 4.2 below.

Table 4.2. Table of the time balanced with different methods of balancing

	PID+KF	PID	P	No balancing
Average time (ms)	4485	4897	3568	4189
Best time (ms)	4724	6920	4439	4334

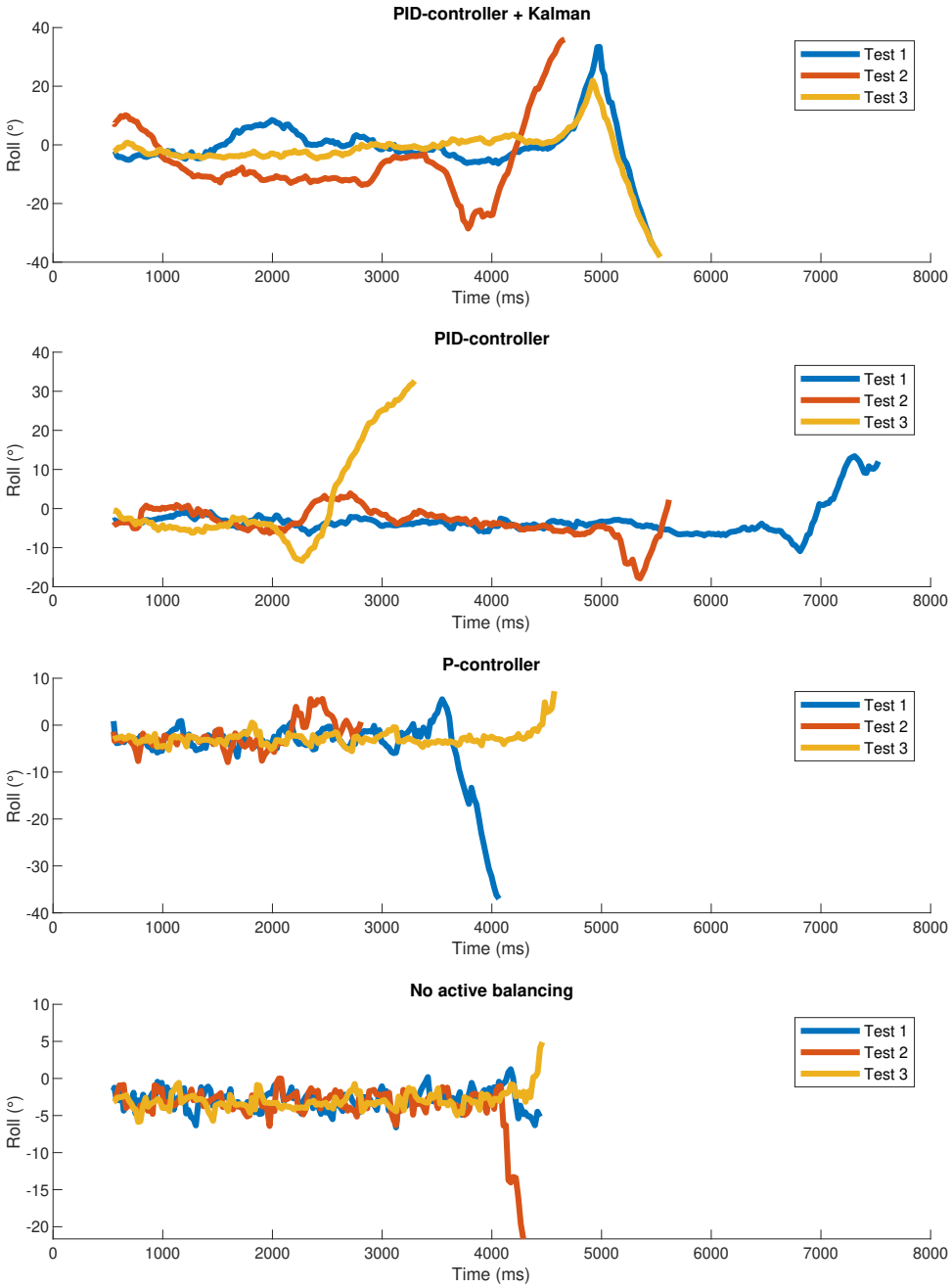

Figure 4.2. Test results for different types of balancing strategies

Chapter 5

Conclusions and Discussion

5.1 Discussion

This section will address and allow discussion regarding the results and the project as a whole.

5.1.1 Design

The play in the support cogs and the side covers allows a bit too much movement sometimes resulting in the wheel getting caught on a side cover, this makes balancing harder and also heavily increases the wear on the DC motor mounted cog (which has been replaced during testing). The cog for the stepper motors balancing motion also sometimes gets stuck which results loss of the active balance and a lot of vibrations causing the accelerometer to give out quite varied values.

5.1.2 Kalman filter

The kalman filter lowers the error from the accelerometer readings but it also incurs a small time penalty that does have some impact for the balancing of the monowheel as the immediate balance is quite time sensitive. Testing has not shown a big difference with the kalman filter included or not suggesting that the increased accuracy might not weigh up for the, albeit slight, time penalty.

5.1.3 PID

A PID controller is a great method for balance. It takes time, a lot of testing and adjusting of parameters for it to reach it's potential. The balance of the monowheel could probably be improved substantially with the right tuning parameters. Ziegler-Nichols method is one way of finding better values for the parameters. In it's current state it is sometimes slow, sometime over corrects and does only increase the balance moderately.

5.1.4 Stepper motor and the natural balance of the construction

There are more possibilities with the use of a stepper motor, most of the testing is done with the 200 base steps. For a smoother motion microstepping up to 6400 steps can be implemented. The attached battery-packs weight on the stepper motor aids the change in CoG it causes with it's movement but it is a quite light weight in comparison to the whole monowheel. Thus reducing it's ability to "save" the monowheel from exaggerated tilt angles. The DC motor has a large part of the total weight which also hangs out quite a bit from the center line of the wheel, to somewhat compensate for this one battery-pack and the powerbank for the arduino has been placed opposite the DC motor. This results in less movable weight and less impact from the stepper motors movement.

5.1.5 Proposal for future work

Other solutions might be better suited for this purpose, such as gyroscopic balance. Involving the DC motor in the algorithm for example: starting with a certain speed then when the monowheel starts tilting a small increase in the speed increases the gyroscopic effects and thus increasing the stability of the monowheel. Furthermore, a more powerful DC motor would provide extra speed for the monowheel, which would improve the stabilization even further. With more precise manufacturing of parts, less play and vibrations can be expected, probably improving on the design.

5.2 Conclusions

It is possible to balance the monowheel to some extent using an Arduino and a PID controller with the method used in this report. The balancing system increased the the average time per run by a second and at its best allowed for a doubling of time spent upright. The construction is however not optimal since the vibrations from both the powertrain and the balancing mechanisms introduced disturbances in the data that caused the balancing to fail.

Bibliography

[1] Patryk Cieslak, Tomasz Buratowsk, Tadeusz Uhl, Mariusz Giergiel, *The Mono-Wheel Robot With Dynamic Stabilization*, Robotics and Autonomous Systems, 59:611–619, 2011

[2] DEGREE PROJECT MECHANICAL ENGINEERING,FIRST CYCLE, 15 CREDITS, STOCKHOLM SWEDEN 2018 Arthur Grönlund, Christos Tolis, *Riderless self-balancing bicycle*, http://kth.diva-portal.org/smash/get/diva2:1237256/FULLTEXT01.pdf

[3] *Arduino. Date accessed: 2021-05-09* https://www.arduino.cc/

[4] *MEMS Accelerometer Gyroscope Magnetometer Arduino. Date accessed: 2021-05-09* https://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino/

[5] H. Johansson *Elektroteknik* Institutionen för Maskinkonstruktion, KTH, 2013

[6] *What is a Stepper Motor : Types Its Working* Date accessed: 2021-05-09 https://www.elprocus.com/stepper-motor-types-advantages-applications/

[7] Torker Glad, Lennart Ljung *Reglerteknik: Grundläggande teori* Studentlitteratur AB, 4(17), 2006

[8] Kalman, R. E. (1960). *A New Approach to Linear Filtering and Prediction Problems* Journal of Basic Engineering. 82: 35–45, 1960. Date accessed: 2021-05-09 https://pdfs.semanticscholar.org/bb55/c1c619c30f939fc792b049172926a4a0c0f7.pdf

[9] *Lectures in The Kalman Filter.* Date accessed: 2021-05-09 http://www.ilectureonline.com/lectures/subject/SPECIALTOPICS/26/190

[10] Gheorghe Deliu, Mariana Deliu, *Monowheel Dynamics.*Date accessed: 2021-05-09
http://www.recentonline.ro/027/DELIU_Gheorghe_02.pdf

[11] Dan Botezatu, *The Plane Motion of the Monowheel Vehicle.* Date accessed: 2021-05-09
http://www.recentonline.ro/042/Botezatu-R42.pdf

[12] Sreevaram Rufus Nireekshan Kumar, Bangaru Akash, T. Thaj Mary Delsy, *Designing the Mono Wheel by Using Self Balancing Techinque*, International Journal of Circuit Theory and Applications, 9(4):29–34, 2016

[13] *Build Electronic Circuits. H-bridge.* Date accessed: 2021-05-09
https://www.build-electronic-circuits.com/wp-content/uploads/2018/11/H-bridge-switches.png

[14] *Kjell & Company. Playknowlogy Uno Rev. 3* , Date accessed: 2021-05-09
https://www.kjell.com/globalassets/productimages/559575_88860.tif?ref=61DF06C805&format=jpg&w=960&h=960&mode=pad

[15] *F1 Depo. MPU6050*, Date accessed: 2021-05-09
https://st2.myideasoft.com/shop/dt/63/myassets/products/293/mpu6050-sensor.jpg?revision=1540787072

[16] *Makerguides. How to control a stepper motor with DRV8825 driver and Arduino.* Date accessed: 2021-05-09
https://www.makerguides.com/drv8825-stepper-motor-driver-arduino-tutorial/

[17] PID Tuner Controller, Date accessed: 2021-05-09
https://pidtuner.com/#/

[18] PIDv2.h library, Date accessed: 2021-05-22
https://playground.arduino.cc/Code/PIDLibrary/

Appendix A

Arduino

```
    /*
 *  Authors:  Nils Winnerholt & Fritiof Ekvall
 *  Date:     2021 - 05 - 08
 *  ----------------------------------------
 *  Made for Bachelors Project in Mechatronics
 *  MF133X
 *  ----------------------------------------
 *  Thank you to: Dejan from http://howtomechatronics.com
 *                Benne de Bakker from http://makerguides.com
 *  For providing example codes that will be used in this project
 *                Brett Beauregard from http://brettbeauregard.com/projects/
 *  For providing the PID_v2 library
 */

#include <Wire.h>
#include <PID_v2.h>

#define dirPin 2
#define stepPin 3
#define stepsPerRevolution 200

#define en  12
#define in1 8
#define in2 7

// Accelerometer and Gyroscope

const int MPU = 0x68;       // MPU6050 I2C address
float AccX, AccY, AccZ;     // Acceleration values
float GyroX, GyroY, GyroZ;  // Gyroscope values
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ; // Angles for the different outputs from the accelerometer
float roll, pitch, yaw;     // Calculated roll, pitch and yaw from the accelerometer readings
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ; // Errors in accelerometer readings
```

```
34 float elapsedTime, currentTime, previousTime;   // Keeps track of
       time
35 int c = 0;                   // Temporary variable
36
37 // Kalman
38
39 float Mea,KG;
40 float Est_t0,Est_t;
41 float E_mea, E_est, E_est0;
42
43
44 // PID
45 char receivedChar;
46 boolean newData = false;
47
48 unsigned long changeTime = 0; // Last time the sensor was triggered
49 double difference = 5;        // Temporary value for the difference
       between the desired angle and true angle
50 double stepperOut = 250;      // Temporary value for the strength of
       the stepper motor
51 double setAngle = -4;         // Desired angle that will result in
       balance
52 String inString;
53
54 double Kp = 0.7764;           // Proprtional constant
55 double Ki = -0.004131;        // Integrating constant
56 double Kd = -72.97;           // Derivating constant
57
58 PID myPID(&difference, &stepperOut, &setAngle,Kp,Ki,Kd, DIRECT);  //
       Creates a PID from the provided information
59
60
61 void setup() {
62
63   // PID
64   myPID.SetMode(AUTOMATIC);
65
66   // Assigns pins for the DC motor
67   pinMode(en,   OUTPUT);
68   pinMode(in1,  OUTPUT);
69   pinMode(in2,  OUTPUT);
70
71   digitalWrite(en,  HIGH);    // Enables the H-bridge
72
73   // From Benne
74   pinMode(stepPin, OUTPUT);
75   pinMode(dirPin, OUTPUT);
76
77   // From Dejan
78   Serial.begin(19200);
79   Wire.begin();                      // Initialize comunication
80   Wire.beginTransmission(MPU);       // Start communication with
       MPU6050 // MPU=0x68
81   Wire.write(0x6B);                  // Talk to the register 6B
```

```
  Wire.write(0x00);                  // Make reset - place a 0 into the 6B register
  Wire.endTransmission(true);        // End the transmission

  calculate_IMU_error();
  delay(20);

}


void loop() {

  // Runs the DC motor
  digitalWrite(in1, HIGH);    // Enables the first transistor pair
  digitalWrite(in2, LOW);     // Disables the second transistor pair

  MPU6050();                  // Calls for the accelerometer readings

  // Kalman filter to improve the quality of data from the accelerometer
  Mea = roll;                         //Inital measurment
  Est_t0 = 1;                         //Inital estimate
  E_mea=1;                            //Error in measurement
  E_est0=0.5;                         //Inital error in estimate
  KG = E_est0/(E_est0+E_mea);         //Kalman gain
  Est_t = Est_t0+KG*(Mea-Est_t0);     //Current estimate
  E_est = (1-KG)*E_est0;              //New error in estimate

  KG = E_est/(E_est+E_mea);
  Est_t = Est_t+KG*(Mea-Est_t);
  E_est = (1-KG)*E_est;

  KG = E_est/(E_est+E_mea);
  Est_t = Est_t+KG*(Mea-Est_t);
  roll=Est_t;

  difference = abs(setAngle-roll);  // Finds the difference between the desired- and acual angle

  myPID.Compute();                  // Compute PID for this itteration

  // Determines the direction for the stepper motor (balancing)
  if( roll > setAngle +1){

    // Set the spinning direction clockwise:
    digitalWrite(dirPin, LOW);

    digitalWrite(stepPin, HIGH);
    delayMicroseconds(stepperOut);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(stepperOut);
  }
  else if ( roll < setAngle-1){
```

```
132
133     // Set the spinning direction counterclockwise:
134     digitalWrite(dirPin, HIGH);
135
136     digitalWrite(stepPin, HIGH);
137     delayMicroseconds(stepperOut);
138     digitalWrite(stepPin, LOW);
139     delayMicroseconds(stepperOut);
140   }
141   else {
142
143     // Turns off the spinning
144     digitalWrite(stepPin, LOW);
145     delayMicroseconds(4000);
146     digitalWrite(stepPin, LOW);
147     delayMicroseconds(4000);
148
149   }
150
151
152 }
153
154
155
156
157 // Code that reads data from the accelerometer
158 // Written by Dejan
159 void MPU6050() {
160
161   // === Read acceleromter data === //
162   Wire.beginTransmission(MPU);
163   Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
164   Wire.endTransmission(false);
165   Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each
        axis value is stored in 2 registers
166   //For a range of +-2g, we need to divide the raw values by 16384,
        according to the datasheet
167   AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value
168   AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value
169   AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value
170   // Calculating Roll and Pitch from the accelerometer data
171   accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 /
        PI) - AccErrorX; // AccErrorX ~(0.58) See the calculate_IMU_error
        ()custom function for more details
172   accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) *
        180 / PI) + AccErrorY; // AccErrorY ~(-1.58)
173
174   // === Read gyroscope data === //
175   previousTime = currentTime;        // Previous time is stored
        before the actual time read
176   currentTime = millis();            // Current time actual time read
177   elapsedTime = (currentTime - previousTime) / 1000; // Divide by
        1000 to get seconds
178   Wire.beginTransmission(MPU);
```

```
  Wire.write(0x43); // Gyro data first register address 0x43
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 4 registers total, each axis value is stored in 2 registers
  GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // For a 250deg/s range we have to divide first the raw value by 131.0, according to the datasheet
  GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
  GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;
  // Correct the outputs with the calculated error values
  GyroX = GyroX + GyroErrorX; // GyroErrorX ~(-0.56)
  GyroY = GyroY - GyroErrorY; // GyroErrorY ~(2)
  GyroZ = GyroZ + GyroErrorZ; // GyroErrorZ ~ (-0.8)

  // Currently the raw values are in degrees per seconds, deg/s, so we need to multiply by sendonds (s) to get the angle in degrees
  gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg
  gyroAngleY = gyroAngleY + GyroY * elapsedTime;
  yaw =  yaw + GyroZ * elapsedTime;

  // Complementary filter - combine acceleromter and gyro angle values
  /*roll = 0.96 * gyroAngleX + 0.04 * accAngleX;
  pitch = 0.96 * gyroAngleY + 0.04 * accAngleY;
  */
  // Improved filter by Chris to stop drift
  gyroAngleX = 0.96 * gyroAngleX + 0.04 * accAngleX;
  gyroAngleY = 0.96 * gyroAngleY + 0.04 * accAngleY;

  roll = gyroAngleX;
  pitch = gyroAngleY-80;

  // Print the values on the serial monitor
  Serial.print("Roll:");
  Serial.print(roll);
  Serial.print(", ");
/*
  Serial.print("/");
  Serial.print(pitch);
  Serial.print("/");
  Serial.println(yaw);
*/
}


// Code that calculates the error in the data from the accelerometer
// Written by Dejan
void calculate_IMU_error() {

  // We can call this funtion in the setup section to calculate the accelerometer and gyro data error. From here we will get the error values used in the above equations printed on the Serial Monitor.
  // Note that we should place the IMU flat in order to get the proper values, so that we then can the correct values
```

```
225   // Read accelerometer values 200 times
226   while (c < 200) {
227     Wire.beginTransmission(MPU);
228     Wire.write(0x3B);
229     Wire.endTransmission(false);
230     Wire.requestFrom(MPU, 6, true);
231     AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
232     AccY = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
233     AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
234     // Sum all readings
235     AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow
      ((AccZ), 2))) * 180 / PI));
236     AccErrorY = AccErrorY + ((atan(-1 * (AccX) / sqrt(pow((AccY), 2)
      + pow((AccZ), 2))) * 180 / PI));
237     c++;
238   }
239   //Divide the sum by 200 to get the error value
240   AccErrorX = AccErrorX / 200;
241   AccErrorY = AccErrorY / 200;
242   c = 0;
243   // Read gyro values 200 times
244   while (c < 200) {
245     Wire.beginTransmission(MPU);
246     Wire.write(0x43);
247     Wire.endTransmission(false);
248     Wire.requestFrom(MPU, 6, true);
249     GyroX = Wire.read() << 8 | Wire.read();
250     GyroY = Wire.read() << 8 | Wire.read();
251     GyroZ = Wire.read() << 8 | Wire.read();
252     // Sum all readings
253     GyroErrorX = GyroErrorX + (GyroX / 131.0);
254     GyroErrorY = GyroErrorY + (GyroY / 131.0);
255     GyroErrorZ = GyroErrorZ + (GyroZ / 131.0);
256     c++;
257   }
258   //Divide the sum by 200 to get the error value
259   GyroErrorX = GyroErrorX / 200;
260   GyroErrorY = GyroErrorY / 200;
261   GyroErrorZ = GyroErrorZ / 200;
262   // Print the error values on the Serial Monitor
263   Serial.print("AccErrorX: ");
264   Serial.println(AccErrorX);
265   Serial.print("AccErrorY: ");
266   Serial.println(AccErrorY);
267   Serial.print("GyroErrorX: ");
268   Serial.println(GyroErrorX);
269   Serial.print("GyroErrorY: ");
270   Serial.println(GyroErrorY);
271   Serial.print("GyroErrorZ: ");
272   Serial.println(GyroErrorZ);
273 }
```

Appendix B

Acumen

```
//Bachelor's Thesis at ITM, KTH
//Balancing a monowheel with a PID controller
//Date:            2021-05-09
//Written by:      Fritiof Andersen Ekvall and Nils Winnerholt
//Examiner:        Nihad Subasic
//TRITA number:    TRITA-ITM-EX 2021:49
//Course code:     MF133X
//Description of the program:
//This is a simulation of the movement of the monowheel as it rolls forward.



model Main(simulator) =
    initially
        m1 = create Monowheel((0,0,0),green,(0,0,0)), //Here is the model created
        x1=0, x1'=0, x1''=0,  //Inital values for the x position and it's derivatives
        x2=0, x2'=0, x2''=0,
        z=0

    always
        if x1<20 //If the x position is small enough it will start moving
            then x1'' = -(x1'-5)
        else if x1'>0
                then x1'' = -7
                else x1'' = 0,
            m1.pos = (x1,0,0),// Change in x-axis position
            m1.rot = (0,0.2*x1,0),// Change in y-axis rotation
            x2'' = 0,
            z = x1-x2
```

```
model Monowheel(pos,col,rot) = //Here is the formula for the creation of the model.
    initially
        _3D =(), _Plot=()//View of what has been created

    always
        _3D =(Cylinder center = pos  //Choice of shape and position for the object
            size = (1,2)     //Size
            color = col    //Colour
            rotation = rot //How it rotates
            transparency = 0.5) //It's transparency
```

TRITA ITM-EX 2021:49