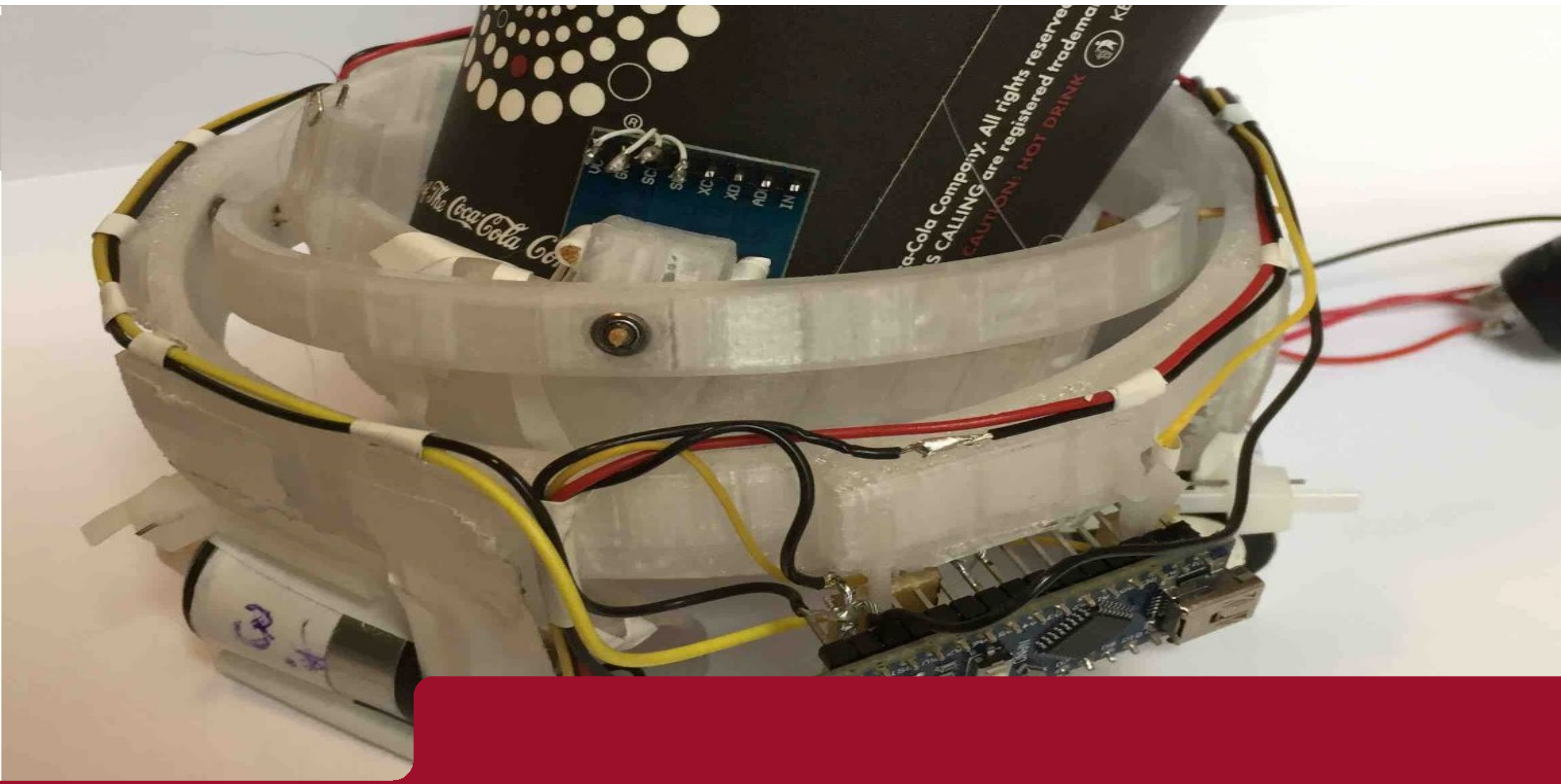
DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2019

Handheld container stabilizer

ALEXANDER MURTAZA

OSCAR STENSTRÖM



KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

Handheld container stabilizer

ALEXANDER MURTAZA
OSCAR STENSTRÖM

Bachelor Thesis at ITM
Examiner: Nihad Subasic

TRITA ITM-EX 2019:36

Abstract

Self-stabilizing systems can be found in many contexts. They are used in aircraft and camera gimbals to name a few. In this project, a self-stabilizing container was constructed. The construction consists of three parts. An inner ring which rotates around the Z-axis, an outer ring which rotates around the Y-axis and a handle with space for three DC motors and a microcontroller. In this project an Arduino Nano was used. To detect inclination an IMU (Inertial Measurement Unit) was deployed. An IMU is a sensor consisting of three gyroscopes and three accelerometers, one for each coordinate axis. The software for the construction consists of four parts; angle reading, a Kalman filter, two PID-controllers and a motor controller. When a container is inserted into the construction the four-part system keeps the container horizontal and stable. Experimental data shows that in 84% of the tests the construction could stabilize the container.

Keyword: Mechatronics, Stable, Self-stabilizing Container, Arduino, MPU6050, PID-controller

Referat

Självstabiliserande behållare

Självstabiliserande system kan man finna i många olika sammanhang. Några exempel på självstabiliserande system är flygplan och kamerastabilisatorer. I detta projekt konstruerades en självstabiliserande behållare. Konstruktionen består av tre delar. En ring som kan rotera runt Z-axeln, en ring som kan rotera runt Y-axlen och ett handtag med plats för likströmsmotorer och mikrokontroller. I detta projekt användes Arduino Nano. För att avläsa vinklarna användes en tröghetsmätare. En tröghetsmätare är en sensor som består av tre gyroskop och tre accelerometrar, en för varje axel. Mjukvaran i konstruktionen består av fyra delar; vinkelavläsning, ett Kalmanfilter, två PID-regulatorer och motorkontroller. Beroende på vilken vinkel konstruktionen har kommer någon av motorerna att korrigera vinkeln på behållaren. Testerna visade att konstruktionen kunde stabilisera behålaren i 84% av alla tester.

Nyckelord: Mekatronik, Stabil, Självstabiliserande behållare, Arduino, MPU6050, PID-regulator

Acknowledgements

We would like to thank the course examiner Nihad Subasic for his lectures and assistance during this project. We would also like to express gratitude to lab assistants Sresht Iyer and Seshagopalan ”Sesh” Thorapalli Muralidharan for all the support they have provided.

Contents

1 Introduction **3**

1.1 Background . . . 3

1.2 Purpose . . . 3

1.3 Scope . . . 4

1.4 Method . . . 4

2 Theory **5**

2.1 Microcontroller . . . 5

2.1.1 PWM-signals . . . 6

2.2 DC-motors . . . 6

2.3 Inertial Measurement Unit . . . 6

2.3.1 Collection of input data . . . 7

2.4 Kalman filter . . . 8

2.5 PID-controller . . . 8

3 Demonstrator **11**

3.1 Hardware . . . 11

3.1.1 The construction . . . 11

3.1.2 The electronic part . . . 12

3.2 Software . . . 13

3.2.1 The PID implementation . . . 13

3.2.2 Determining the duty cycle . . . 14

4 Results **17**

4.1 Test description . . . 17

4.2 Stabilizing in Z-axis results . . . 17

4.3 Stabilizing in Y-axis . . . 17

4.4 Test success rate . . . 18

5 Conclusions and discussion **19**

5.1 Discussion . . . 19

5.2 Conclusions . . . 20

6 Recommendations and future work **21**

Bibliography **23**

Appendices **24**

A Program code **25**

List of Figures

1.1 The final prototype 4

2.1 Arduino Nano 5
2.2 PWM signals 6
2.3 MPU6050 7
2.4 MPU6050, the orientation of the axes 7
2.5 Kalmanfilter 8
2.6 PID-controller 9

3.1 Rendered model of the prototype 11
3.2 Wiring diagram 12
3.3 Flowchart 13
3.4 IMU mounted on inner ring. Wire connection points 1,2 and 3 are highlighted 14

List of Tables

3.1 Table of the PID-constants 14

4.1 Stabilizing in Z-axis results 17
4.2 Stabilizing in Y-axis 17

List of Abbreviations

abs Absolute value

DC-motor Direct Current motor

IDE Integrated Developed Environment

IMU Inertial Measurement Unit

ms milliseconds

PID Proportional–Integral–Derivative controller

PWM Pulse Width Modulation

RPM Revolutions Per Minute

Chapter 1

Introduction

1.1 Background

Self-stabilizing systems can be found in many contexts. They are used in aircraft and camera gimbals applications to name a few. People with some form of motor disability may require aid to perform tasks such as eating or drinking. Different types of self-stabilizing systems could facilitate these people. In a previous project a self-stabilizing spoon was designed[1].

In this project, the aim was to construct a self-stabilizing container. The main task of the construction is to keep a container horizontal at all times and adapt to whatever change in inclination the container is experiencing. The only exception to this is when the user is drinking; here tilt should be allowed. In order to accomplish this, a switch will be added to the construction.

1.2 Purpose

The purpose of this project was to investigate how a microcontroller from Arduino can be implemented to stabilize a container. Other questions that were investigated in this project were:

- With regards to the container, what level of stability can be achieved?
- How quickly can the system respond if the container is being tilted?
- How should the respective duty cycles of the motors used be determined?

1.3 Scope

A series of possible designs were investigated aiming to allow the user to rotate a handle in different directions while the container remains horizontal. The connection between the handle and container was initially not defined. Close to friction free rotation around two axes was desired as well as the ability to apply force to the area where the container is mounted. The construction should also remain lightweight and compact, since its applications require it to be handheld.

The final prototype resembled a spherical gimbal, see Figure 1.1. Material was added to allow components to be mounted on the outer ring (the handle). Although supposedly different containers could be stabilized by the prototype only a plastic coffee mug was used during the project.

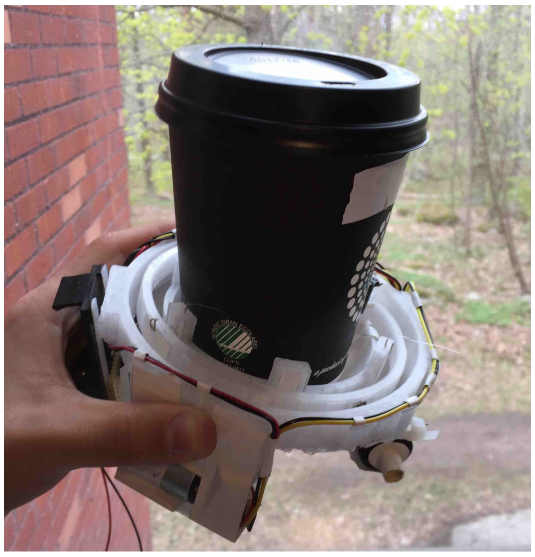


Figure 1.1: The final prototype

1.4 Method

The construction and other components that could not be easily found in stores were designed in Solid Edge ST 10[2] and then 3D-printed with a printer from Ultimaker. The programming of the construction was written in Arduino's IDE (Integrated Developed Environment).

To answer the purpose, some experiments were conducted. The experiments were designed to test the stability of the construction for each axis.

Chapter 2

Theory

2.1 Microcontroller

To control the electronics a programmable microcontroller was used. A microcontroller is a small computer. Just like a regular computer a microcontroller has a CPU (Central Processing Unit) and RAM (Random Access Memory). On the microcontroller, pins are mounted where you can connect electronic components. Normally when the pin is set to ”one” the pin has an output voltage of 5 V[3].

Arduino Nano was the chosen microcontroller for this project, see Figure 2.1. It has digital and analog input switches which allow input signals from sensors. Its PWM (Pulse Width Modulation) outputs can be used to vary the RPM (Revolutions Per Minute) of a DC-motor (Direct Current-motor) to balance the container.

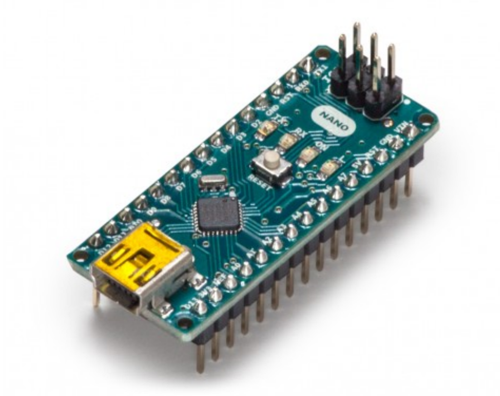


Figure 2.1: Arduino Nano[4]

2.1.1 PWM-signals

PWM is a method to control electronics through a microcontroller. The PWM output signals are shaped like square waves as they are periodically switched on and off at a certain peak value voltage, see Figure 2.2. The percentage of time when the voltage is "on" is a duty cycle. Consequently, the duty cycle will determinate the RPM of a motor. If the duty cycle is 100 percent the current will be supplied continuously, and the motor will have maximum RPM[5].

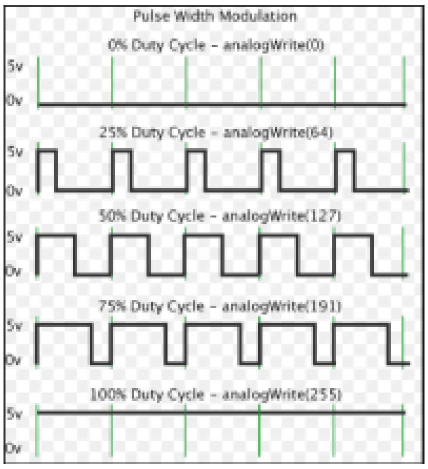

Pulse Width Modulation
0% Duty Cycle - analogWrite(0)
25% Duty Cycle - analogWrite(64)
50% Duty Cycle - analogWrite(127)
75% Duty Cycle - analogWrite(191)
100% Duty Cycle - analogWrite(255)
5v
0v

Figure 2.2: Square wave PWM signals[5]

2.2 DC-motors

In this project, three DC-motors were used. A DC-motor may be supplied with direct current from batteries. The electric effect that is supplied from batteries is converted to mechanical effect. The difference between the electric effect and the mechanical effect is considered losses[3].

2.3 Inertial Measurement Unit

An Inertial Measurement Unit(IMU) is a sensor consisting of three gyroscopes and three accelerometers, one for each coordinate axis. The IMU can detect angle changes and may be used to measure how much inclination a system is experiencing. The accelerometers detect how much a system accelerates, in a similar sense. This technique is used e.g in aircraft[6].

The accelerometer can only estimate the roll and pitch angle (horizontal plane) because it bases its measurements on gravitational (vertical) forces. This causes the gyroscope to set the yaw angle to 0° [7]. The chosen IMU model for this project was the MPU6050 from InvenSense Inc, see Figure 2.3[8]. Both its accelerometers and its gyroscopes have flaws which must be accounted for.

The accelerometer is sensitive to vibrations. This causes measurement errors to increase over time. The gyroscope has a drift error. Gyro values will drive far from real values even when no movement is occuring[9].

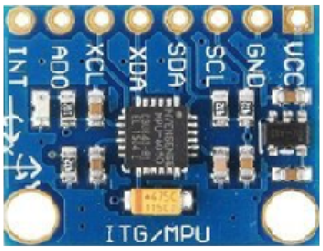


Figure 2.3: The chosen IMU for the project, model MPU6050[12]

2.3.1 Collection of input data

The microcontroller program to be used requires input data from the IMU. This model MPU6050 acts as a "slave" device when used in conjunction with the Arduino Nano microcontroller[10].

The measured data is stored on the different registers of the IMU. In code the data is collected by dereferencing the addresses of the correct registers. The registers are specified in the register map of the MPU6050[11]. By reading and writing to its registers the IMU can also be set up as needed. Its coordinate axes are shown in Figure 2.4 below.

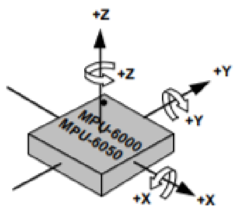


Figure 2.4: The orientation of the measurement axes of the IMU[8]

2.4 Kalman filter

A Kalman filter is an optional recursive algorithm. The filter is useful when the information received from a dynamic system is unpredictable (for this application the angle of the container to be stabilized). With a variety of incomplete, noisy measurements or other random perturbations, the algorithm can estimate the real state of the dynamic system. The algorithm uses statistical methods and acquired measurements to optimize final values[13]. See Figure 2.5 for an illustrated implementation.

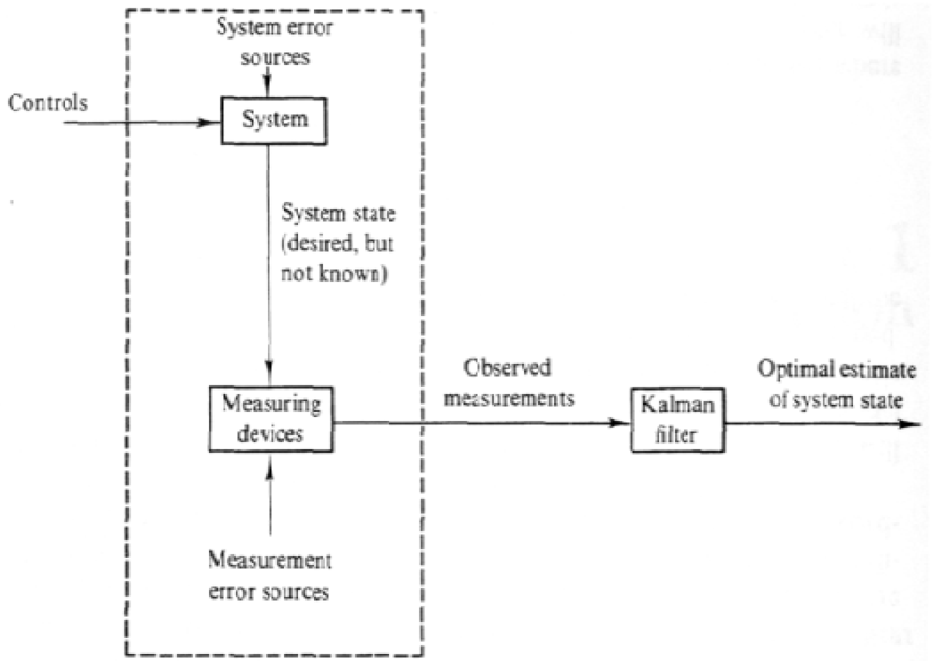

System error sources
Controls
System
System state (desired, but not known)
Measuring devices
Measurement error sources
Observed measurements
Kalman filter
Optimal estimate of system state

Figure 2.5: A Kalman filter implementation[14]

2.5 PID-controller

The proportional–integral–derivative controller (PID-controller) is one of the most common feedback mechanism deployed to control a system. With the steady-state error

$$e(t) = r(t) - y(t) \tag{2.1}$$

the input of the system $u(t)$ can be calculated. $r(t)$ is the desired value and $y(t)$ is the actual value. Figure 2.6 shows a typical feedback mechanism. $u(t)$ is calculated by the formula

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_D \frac{d}{dt} e(t) \tag{2.2}$$

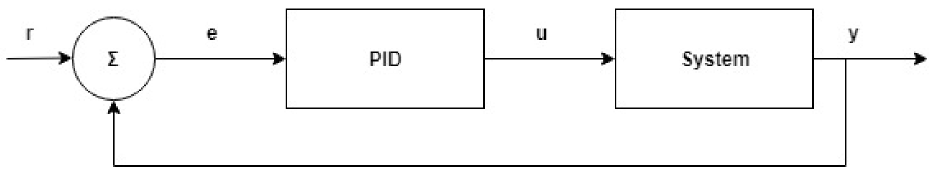


Figure 2.6: A PID-controller implementation[15]

Equation (2.2) has three parts: a proportional, an integral and a derivative term. The proportional part can compensate for disturbances but cannot eliminate their effect. The proportional part cannot eliminate the steady-state error. The integral part can eliminate the steady-state error but the system can be unstable for larger values of K_p and K_i. The derivative part grants improved stability overall[16].

Chapter 3

Demonstrator

3.1 Hardware

3.1.1 The construction

Similarly to other self-stabilizing systems, a two-axis gimbal design was utilized in this project. A two-axis gimbal can stabilize something in two axis[17]. In this project the construction should stabilize the container around both the Y- and Z-axis.

The construction consists of three parts. An inner ring rotating around the Z-axis, on which the container and gyroscope is mounted, an outer ring that can rotate freely around the Y-axis and an outer handle with space for the DC motors and Arduino Nano.

The construction has three motors which are utilized to correct the container when it tilts. To enable this, wires are connected between the revolving axes of the motors and the inner ring of the construction. Figure 3.1 displays a rendered model of the prototype base onto which additional hardware could be mounted.

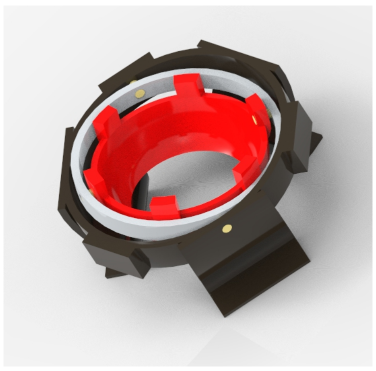


Figure 3.1: Rendered model of the prototype frame[18]

3.1.2 The electronic part

The electronic parts of the construction consist of a microcontroller, Arduino Nano, three DC-motors from Maxon, the IMU MPU6050 and three transistors. The transistors were connected to the microcontroller, the power supply and the motors. The input to the system was the angles received from the MPU6050 and the output was the duty cycle of the DC-motors. For the complete wiring diagram, see Figure 3.2.

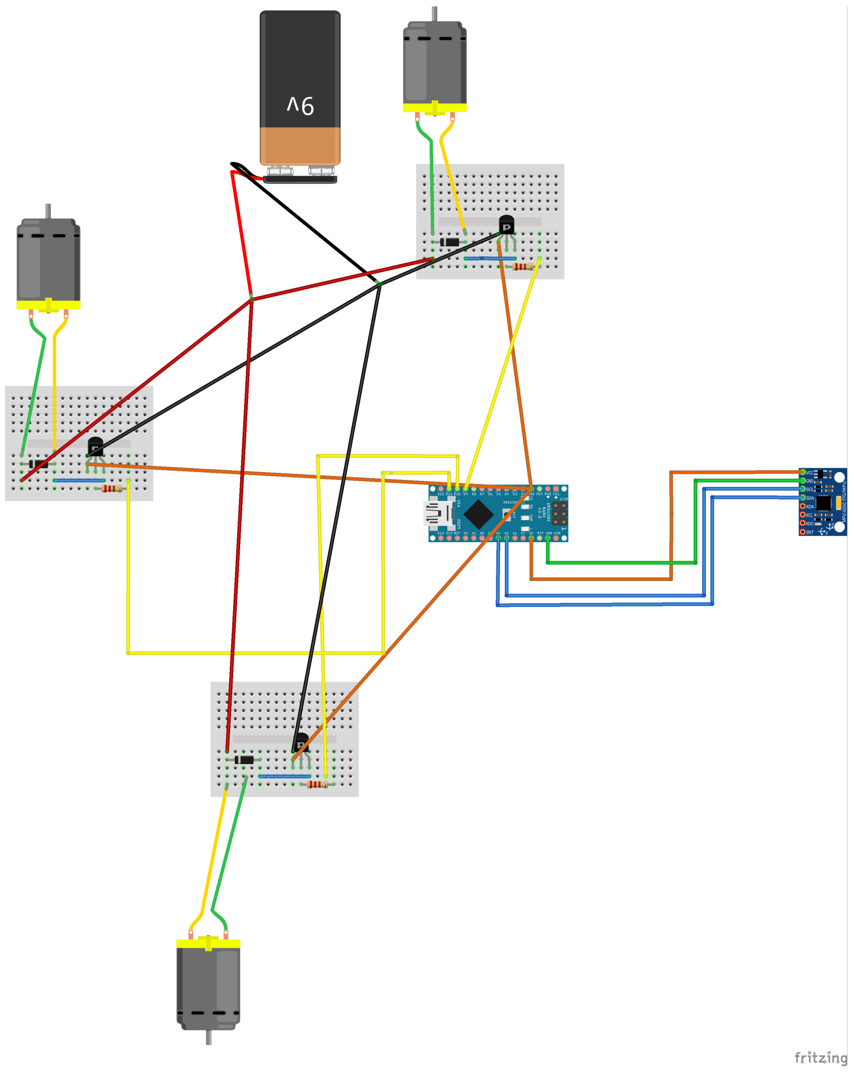


Figure 3.2: Wiring diagram[19]

3.2 Software

The software for the construction was programmed in Arduino IDE. The program is divided into four parts; the angle reading and data collection, the Kalman filter, the PID-controller and the motor controller. Depending on the received change of inclination different motors correct the angle of the container. The full Arduino code can be found in Appendix A and an overview is shown in Figure 3.4.

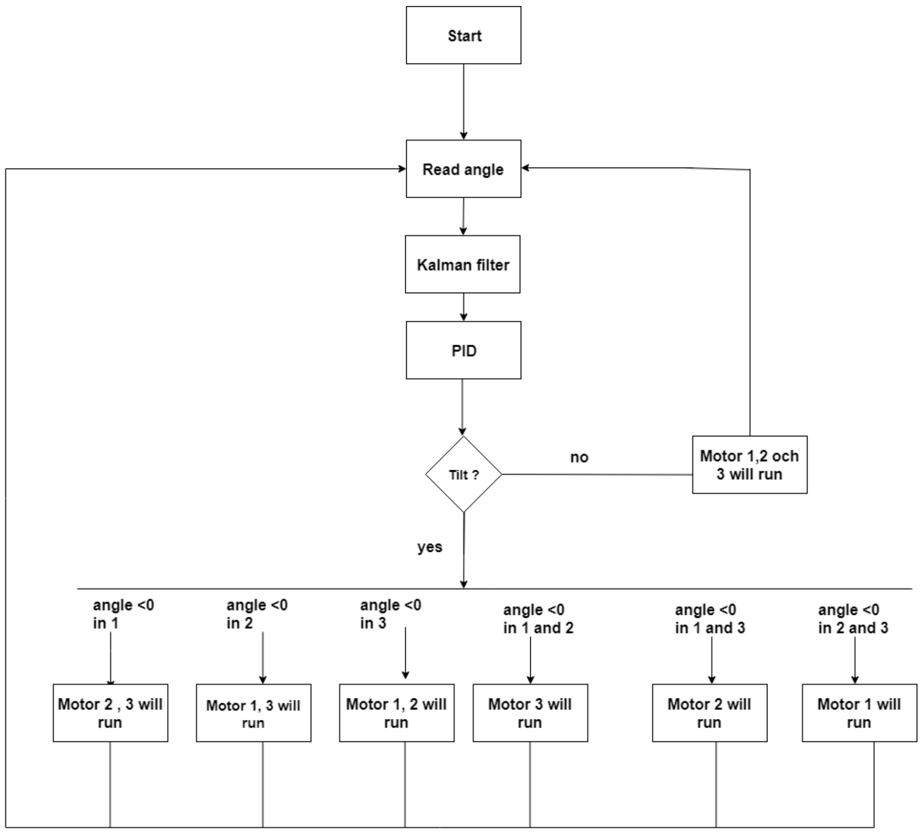


Figure 3.3: Flowchart of programmable software [15]

3.2.1 The PID implementation

In this project two PID-controllers were necessary, one for the Z-direction and one for the Y-direction. The constants of the PID-controllers K_p, K_i and K_d were chosen to allow the system to correct the tilting container as quickly as possible without considerable overshoots and/or steady-state errors. In this project the input is the angle and the output is the duty cycle. Table 3:1 show the chosen values.

Table 3.1: Table of the PID-constants

	Z	Y
K_p	0,047	0,05
K_i	0,0035	0,002
K_d	0,00000019	0,00000019

3.2.2 Determining the duty cycle

How the duty cycle of each motor should be determined was dependent on the conditions of this particular project. In this thesis, a method was suggested which accounts for both the total inclination of the container to be stabilized and for the dominant direction of this inclination.

The container is mounted on the innermost ring. This ring is being pulled by wires connected as shown in Figure 3.3.

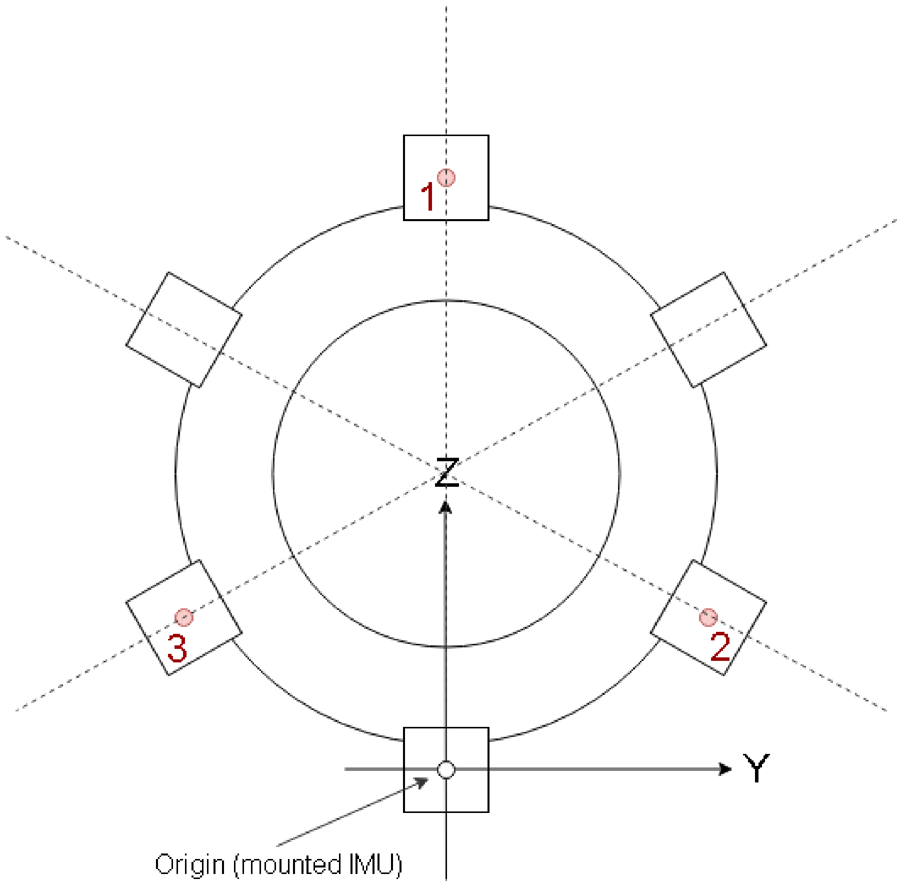


Figure 3.4: IMU mounted on inner ring.[15]

Inclination around the Y- and Z-direction should be translated into motor outputs, allowing the wires to apply force on the connected points. These connected points are fixed in the coordinate system of the IMU. Consequently a unit vector corresponding to each connection point of each motor can be found (these are not direction vectors, although similar):

$$\vec{e_1} = (1, 0) \tag{3.1}$$

$$\vec{e_2} = (-1/2, -\sqrt{3}/2) \tag{3.2}$$

$$\vec{e_3} = (-1/2, \sqrt{3}/2) \tag{3.3}$$

In the IDE program an output vector is produced by the PID-controller, containing required outputs in Y- and Z-directions. This output vector is projected onto the unit direction vectors of the motors $i = 1, 2, 3$.

$$(ProjectedOutputofmotor_i) = (InclinationY, InclinationZ) \cdot \vec{e_i} \tag{3.4}$$

The value of this projection is used directly to scale the RPM of the motors. The projection is multiplied by an appropriate constant causing the duty cycle to range from 0 to 255.

Chapter 4

Results

4.1 Test description

The experiments were designed to test the stability of the construction for each axis. An additional MPU6050 was utilized and placed on the handle. Using the two MPU6050 the respective angles of the handle and the container could be noted.

How rapid the construction stabilized was investigated during the experiments. The present angles after 250, 500 and 1000 ms were noted. During the experiments, the construction was tilted from 0° to an arbitrary random angle. The absolute value (abs) of the received angle average was calculated. The results can be seen in Table 4.1 and 4.2.

4.2 Stabilizing in Z-axis results

Table 4.1: Stabilizing in Z-axis results

The abs of the angle after 250 ms	9,58°
The abs of the angle after 500 ms	9,42°
The abs of the angle after 1000 ms	7,80°

4.3 Stabilizing in Y-axis

Table 4.2: Stabilizing in Y-axis

The abs of the angle after 250 ms	13,46°
The abs of the angle after 500 ms	11,04°
The abs of the angle after 1000 ms	7,10°

4.4 Test success rate

Each experiment also received a grade between 1-3, where 1 was considered unsatisfying and 3 was considered excellent. A steady-state error smaller than the absolute value of 15° was deemed successful. Experiments with a steady-state error greater than 15° received grade 1 and experiments with errors less than 15° received the grade 2 or 3, depending on possible overshoot behaviour. 84 % of the tests received grades 2 or 3 as a result of this.

Chapter 5

Conclusions and discussion

5.1 Discussion

As displayed in table 4.1 and 4.2 the construction stabilizes faster in the Z-direction than in the Y-direction. This may be due to the positioning of the wire connection points displayed in figure 3.3, where the Z-axis of rotation aligns with connection point no. 1. This means that only motors 2 and 3 cause rotation around the Z-axis. This eliminates possible errors occurring when three motors are deployed, which otherwise is the case when rotation around the Y-axis is required.

Conducted tests indicates that the construction is sensitive to disturbances. Vibrations from the hand, the motors or the surroundings can affect the system causing oscillation. A reason could be the MPU6050:s sensitivity to vibrations and possible manufacturing errors. To combat this, during testing both Arduino units had to be repeatedly restarted to reset the IMU:s.

Duty cycles and experiment results were notably dependent on precise mounting of the IMU:s. The axes of both units had to align as displayed in figure 3.3, which was difficult in practice. This may result in inaccurate motor unit vectors. Also this could have caused errors in measured starting angles of the handle during test experiments.

In general the construction works as intended but some hardware problems have been identified. Improvements to software is also possible. For example, appropriate PID-controller constants K_p, K_i and K_d are dependent on the current inclination of the system. They also depend on the current inclination errors present. Having these constants vary for different inclinations and inclination errors of the container could improve overall performance of the system.

5.2 Conclusions

Some conclusions can be made with respect to the research questions of the project. Increased level of stability is achieved, where in 84 % of cases forced inclination was reduced to below 15°. The system responds quickly as a result of its design, however, the resulting tilt is not always satisfying. A method is suggested by which to determine the duty cycles of each of the three motors. This is done by projecting inclination (around the Y-axis and Z-axis, respectively) onto the unit vectors of each motor. The test results are an indication of this suggested methods applicability and success, although refinement is possible.

Chapter 6

Recommendations and future work

Some possible changes have been identified which if performed may improve the stabilizer. A more dynamic PID-controller where the constants can be changed would be useful. These would be depending on the mass and inertia of the container that is being stabilized by the construction. A machine learning algorithm could possibly be implemented to determine these dynamic PID-constants. They would be depending on the different properties of the container.

Hardware changes could also be performed. Different material for the wire holders is desired since wire tangling was causing difficulties. In this project polylactic acid is used. A different placement of the wire connection points may also improve performance since motor connection point 1 currently aligns with the Z-axis of the IMU. The IMU itself is sensitive to vibrations but this issue is hard to address.

Bibliography

[1] J. Abrahamsson and J. Dano, *The Stabilizing Spoon.* Bachelor Thesis, KTH Royal Institute of Technology, School of Industrial Engineering and Management, 2016

[2] *Solid Edge ST 10.* [Software] Available: `https://solidedge.siemens.com/en/`

[3] H. B Johansson, *Elektroteknik* KTH Royal Institute of Technology, School of Industrial Engineering and Management, 2006

[4] Arduino *ARDUINO NANO* [Online] 2019-04-21 Available: `https://store.arduino.cc/arduino-nano`

[5] Arduino.cc, T. Hirzel *PWM*[Online] 2019-03-24 Available: `https://www.arduino.cc/en/Tutorial/PWM`

[6] G.Hasan, K.Hasan, R.Ahsan, T.Sultana, and R.C.Bhowmik, *Evaluation of a Low-Cost MEMS IMU for Indoor Positioning System* IJESE, vol 1, pp 70-77, 2013, Available: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.683.7012&rep=rep1&type=pdf`

[7] Y. C Han, K. I Wong, I. Murray, *2-Point Error Estimation Algorithm for 3-D Thigh and Shank Angles Estimation Using IMU* IEEE, vol 18, pp 8525-8531, 2018, Available: `https://ieeexplore-ieee-org.focus.lib.kth.se/stamp/stamp.jsp?tp=&arnumber=8438479`

[8] InvenSense Inc. *MPU-6000 and MPU-6050 Product Specification Revision 3.4*[Online] 2019-03-24 Available: `https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf`

[9] A. Cismas, I. Matei, V. Ciobanu, G. Casu, *Crash detection using IMU sensor* CSCS pp 672-676, 2017, Available: `https://ieeexplore-ieee-org.focus.lib.kth.se/stamp/stamp.jsp?tp=&arnumber=7968631`

[10] Arduino *MMPU-6050 Accelerometer + Gyro* [Online] 2019-03-24 Available: `https://playground.arduino.cc/Main/MPU-6050/`

[11] InvenSense Inc. *MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2* [Online] 2019-03-24 Available: `https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf`

[12] Indiamart *MPU-6050 MPU6050 Module IMU 6 DOF 3 Axis Gyro 3 Axis Accelerometer Sensor*[Online] 2019-03-24 Available: `https://5.imimg.com/data5/IU/JN/MY-44787231/mpu6050-02-500x500.jpg`

[13] Peter S. Maybeck, *Stochatic models, estimation and control volume 1*, 1979, Available: `https://www.cs.unc.edu/~welch/kalman/media/pdf/maybeck_ch1.pdf`

[14] Y. Chinniah, R. Burton, S. Habibi, *Failure monitoring in a high performance hydrostatic actuation system using the extended Kalman filter* Mechatronics, vol 16, pp 643-653, 2016 Available: `https://www-sciencedirect-com.focus.lib.kth.se/science/article/pii/S0957415806000663`

[15] *Draw.io* Available: `https://www.draw.io/`

[16] T. Glad och L. Ljung, *Reglerteknik Grundläggande teori* Studentliteratur, Lund, 2006

[17] B. Ahi and A. Nobakhti, *Hardware Implementation of an ADRC Controller on a Gimbal Mechanism* IEEE, vol 6 pp 2268-2275, 2018, Available: `https://ieeexplore-ieee-org.focus.lib.kth.se/stamp/stamp.jsp?tp=&arnumber=8059856`

[18] *Keyshot 6.4* [Software] Available: `https://www.keyshot.com/`

[19] *Fritizing* [Software] Available: `http://fritzing.org`

Appendix A

Program code

```
/* Author: Alexander Murtaza and Oscar Stenstr m
Name of the project: Dynamic displacement cancellation
   container
Name of the program: CDEPR
TRITA NUMBER: ITM-EX 2019:36
Date: 2019-05-05


// MPU Connection to Arduino
VCC->5V
GMD->GMD
SCL->A5
SDA->A4
motor pin D9, D10, D11




*/

/* 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
 */


#include <Wire.h>
#include <Kalman.h> // Source: https://github.com/
   TKJElectronics/KalmanFilter
#include <PID_v1.h>
#define RESTRICT_PITCH // Comment out to restrict roll to
      90deg   instead - please read: http://www.freescale.com
   /files/sensors/doc/app_note/AN3461.pdf

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

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

float gyroXangle, gyroYangle, gyroZangle; // Angle calculate
     using the gyro only
float compAngleX, compAngleY; // Calculated angle using a
   complementary filter
float kalAngleX, kalAngleY, kalAngleZ; // Calculated angle
   using a Kalman filter
float gyroZangleFilter=0;
uint32_t timer;
uint8_t i2cData[14]; // Buffer for I2C data

//PID data Z
double SetpointZ=0, InputZ, OutputZ;
double KpZ=0.047  , KiZ=0.0035, KdZ=0.00000019;
PID PIDZ(&InputZ , &OutputZ , &SetpointZ , KpZ, KiZ , KdZ,
   DIRECT) ;

// PID data Y
double SetpointY=0, InputY , OutputY;
double KpY=0.05   , KiY=0.002, KdY=0.00000019;
PID PIDY(&InputY , &OutputY , &SetpointY , KpY, KiY , KdY,
   DIRECT) ;


// TODO: Make calibration routine
void setup() {
  pinMode(11, OUTPUT);           //pin 11 to output
  pinMode(9, OUTPUT);             //pin 9 to output
  pinMode(10, OUTPUT);                //pin 10 to output
  Serial.begin(115200); //Sets the data rate in bits per
     second (baud)
  Wire.begin(); //Initiate the Wire library and join the I2C
      bus as a master or slave
#if ARDUINO >= 157
  Wire.setClock(400000UL); // Set I2C frequency to 400kHz
#else
  TWBR = ((F_CPU / 400000UL) - 16) / 2; // Set I2C frequency
      to 400kHz
#endif

  i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz
     /(7+1) = 1000Hz
  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] != 0x68) { // 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;
  compAngleX = roll;
  compAngleY = pitch;

  timer = micros();

//Convert the angles to PWM-signals big angle-->big PWM
   signal

if (kalAngleX<0){
InputZ=map(kalAngleX,0,30,0,255);}

else{
InputZ=-map(kalAngleX,0,30,0,255);}

if (kalAngleY<0){
```

```
InputY=map(kalAngleY,0,30,0,255);}

else{
InputY=-map(kalAngleY,0,30,0,255);}


//The desired value
SetpointY=0;
SetpointZ=0;

 //Turn the PID
PIDZ.SetMode(AUTOMATIC);
PIDY.SetMode(AUTOMATIC);
PIDZ.SetSampleTime(10);
PIDY.SetSampleTime(10);

} }

void loop() {
  /* 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(accX, accY) * RAD_TO_DEG;
  double pitch = atan(accZ / sqrt(accY * accY + accX * accX)
     ) * 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
  double gyroZrate = gyroZ / 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);
    compAngleX = 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);
    compAngleY = 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;
  gyroZangle += gyroZrate * dt;
  //gyroXangle += kalmanX.getRate() * dt; // Calculate gyro
     angle using the unbiased rate
  //gyroYangle += kalmanY.getRate() * dt;

  compAngleX = 0.93 * (compAngleX + gyroXrate * dt) + 0.07 *
      roll; // Calculate the angle using a Complimentary
     filter
  compAngleY = 0.93 * (compAngleY + gyroYrate * dt) + 0.07 *
      pitch;
  // Reset the gyro angle when it has drifted too much
  if (gyroXangle < -180 || gyroXangle > 180)
    gyroXangle = kalAngleX;
  if (gyroYangle < -180 || gyroYangle > 180)
    gyroYangle = kalAngleY;

  /* Print Data */
#if 0 // Set to 1 to activate
 // Serial.print(accX); Serial.print("\t");
  //Serial.print(accY); Serial.print("\t");
  //Serial.print(accZ); Serial.print("\t");

  // Serial.print(gyroX); Serial.print("\t");
  // Serial.print(gyroY); Serial.print("\t");

  Serial.print("\t");
#endif

  //Set the orgin
    kalAngleX=kalAngleX+90;
    kalAngleY=kalAngleY-8;




//Convert the angles to PWM-signals big angle-->big PWM
   signal
if (kalAngleX<0){
InputZ=map(kalAngleX,0,30,0,255);}
```

```
else{
InputZ=-map(kalAngleX,0,30,0,255);}

if (kalAngleY<0){
InputY=map(kalAngleY,0,30,0,255);}

else{
InputY=-map(kalAngleY,0,30,0,255);}

//compute the output
PIDZ.Compute();
PIDY.Compute();


 // Serial.print(roll); Serial.print("\t");
  //Serial.print(gyroXangle); Serial.print("\t");
 // Serial.print(compAngleX); Serial.print("\t");
  Serial.print(" | gX = ");
  Serial.print(kalAngleX); Serial.print("\t");

  Serial.print("\t");

  //Serial.print(pitch); Serial.print("\t");
  //Serial.print(gyroYangle); Serial.print("\t");
  // Serial.print(compAngleY); Serial.print("\t");
  Serial.print(" | gY = ");
  Serial.print(kalAngleY); Serial.print("\t");

// #if 0 // Set to 1 to print the temperature
 //Serial.print("\t");

  //double temperature = (double)tempRaw / 340.0 + 36.53;
  //Serial.print(temperature); Serial.print("\t");
// #endif


  Serial.print("\r\n");
  delay(50);

 //Determining the duty cycle

double e1x= 1;
double e1y= 0;
double e2x= -0.5;
double e2y= -sqrt(3)/2;

double e3x= -0.5;
double e3y= sqrt(3)/2;

if(kalAngleX<0){
  OutputZ=-OutputZ;
}

if(kalAngleY<0){
  OutputY=-OutputY;
  }



double theta1= OutputY*e1x+OutputZ*e1y;
double theta2= OutputY*e2x+OutputZ*e2y;
double theta3= OutputY*e3x+OutputZ*e3y;


Serial.print(" | theta1= ");
Serial.print(theta1); Serial.print("\t");
Serial.print(" | theta2= ");
Serial.print(theta2); Serial.print("\t");
Serial.print(" | theta3= ");
Serial.print(theta3); Serial.print("\t");





float PWM1, PWM2, PWM3;
if (theta1<0){   //The PWMM of motor 1
PWM1=80+map(theta1, -35, 0, -80, 0);}

else if (theta1>0){
PWM1=80+map(theta1, 0, 35, 0, 180);}



if (theta2<0){   //The PWMM of motor 2
PWM2=80+map(theta2, -35, 0, -80, 0);}
```

```
else if (theta2 >0){
PWM2=80+map(theta2 , 0, 35, 0, 180);}


if (theta3 <0){   //The PWMM of motor 3
PWM3=80+map(theta3 , −35, 0, −80, 0);}

else if (theta3 >0){
PWM3=80+map(theta3 , 0, 35, 0, 180);}


Serial.print(" | PWM1 = ");
Serial.print(PWM1); Serial.print("\t");
Serial.print(" | PWM2 = ");
Serial.print(PWM2); Serial.print("\t");
Serial.print(" | PWM3 = ");
Serial.print(PWM3); Serial.print("\t");


analogWrite(9, PWM1);     //Motor 1
analogWrite(10, PWM2);         //Motor 2
analogWrite(11, PWM3);         //Motor 3





}
```

ISSN TRITA ITM-EX 2019:36