Contents

INTRODUCTION

1.1 Project background

A two wheel self-balancing robot is an important kind of mobile robots. Balancing robots means the capability of the robot to balance on its two wheels without falling. The inverted pendulum system, unlike many other control systems is naturally unstable. Therefore, the system has to be controlled to reach stability in this unstable state. A two wheeled balancing robot is simply an inverted pendulum system which stands upright on two wheels.

The research of twowheeled balancing robots has increased in recent years due to the invention of human transporter application, Segway. This particular project consists of the modeling of the robot, design a Proportional-Integral-Derivative controller and implement the controller on the two-wheeled robot.

1.2 Problem Statement

The inverted pendulum system is naturally unstable. Therefore, a suitable control system technique and method needs to be investigated to control the system. The two wheel balancing robot is an application of the inverted pendulum that requires a controller to maintain its upright position. To achieve this,controller needs to be designed and implement on the robot to balance the inverted pendulum.

1.3 Aim and Objectives

The development of the two wheel robot based on the inverted pendulum concept by using PID controller is the aim of this research. In order to achieve this aim, the objectives as follows are formulated:

i. To design and develop the prototype for two-wheel balancing robot with PID controller.
ii. To evaluate the performance of the developed self-balancing robot using a standard approach.

LITERATURE REVIEW

2.1 Two wheel self balancing robots

Wheeled inverse pendulum model has gained lots of interest recently with the introduction of one popular commercial product (Segway) . Dual-wheel robot has two wheels and an upright body frame where all the circuits are placed. It adjusts its position when it is about to fall forward or backward to avoid instability . The advantage of a two wheel balancing robot is that there is no auxiliary wheel .

Felix Grasser , a researcher at the Swiss Federal Institute of Technology has built JOE,a prototype of a two-wheeled vehicle as shown in Figure 1. The main objective of this robot is to balance its driver on two coaxial wheels. Each of the coaxial wheels is coupled to a dc motor. The vehicle is controlled by applying a torque to the corresponding wheels and it able to do stationary U-turns due to its configuration. A linear state space controller utilising sensory information from a gyroscope and motor encoders is used to stabilise the system.

Figure 1: JOE developed by Grasser

Dean L. Kamen has invented SEGWAY PT as human transporter applications from two wheel robot. The model is illustrated in Figure 2. This robot is an electric, self-balancing human transporter with a complex, computer controlled gyroscopic stabilization and control system. The device balances on two parallel wheels and is controlled by moving body weight. By utilizing this technology, the user can traverse in small steps or curbs and allow easy navigation on various terrains. This innovation uses five gyroscopes and a collection of other tilt sensors to keep itself upright. Only three gyroscopes are needed for the whole system, the additional sensors are included as a safety precaution. Meanwhile, because of its commercial value, Segway Inc. has developed and marketed a series of two-wheel personal transporter and achieved great commercial success since 2003.

Figure 2: Segaway PT

2.2 Fundamentals of Inverted Pendulum

The inverted pendulum system is a classic control problem that is used in the field of Control Theory and Engineering to understand it's dynamic. The system consists of an inverted pole with mass, m, hinged by an angle θ from vertical axis on a cart with mass, M, which is free to move in x directions as shown in Figure 3. A force, F is required to push the cart horizontally

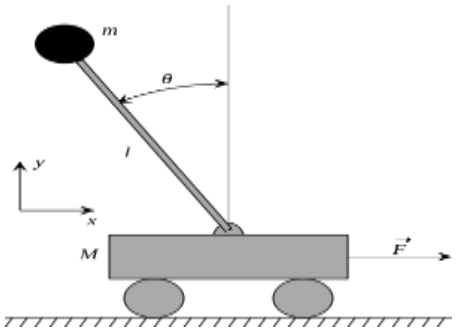


Figure 3 Free body diagram of inverted pendulum system

In order to obtain the system dynamics, the following assumptions have to be made:

i. The system starts in equilibrium state and the initial conditions are assumed to be zero.
ii. The pendulum does not move more than a few degrees away from the vertical to satisfy a linear model.

iii. A step input (displacement of the pendulum, θ) is applied to the system.

2.3 Types of controller to balance two wheel robot

The robot is inherently unstable and without external control it would roll around the wheels rotation axis and eventually fall . Various types of controllers were implemented on two wheeled balancing robot such as Linear Quadratic Regulator (LQR), Pole-Placement Controller and Fuzzy Logic Controller (FLC) and Sliding Mode Controller (SMC). Though many papers have simulation results, experimental results are particularly lacking for the most of the linear and nonlinear controller. There are many problems implementing them in practice especially in LQR, SMC, FLC though simulations can be used to show robustness to disturbances and model uncertainties. It is also uncertain how the implementation of the controller can be done in practice and there is also a lack of comparison between different controller strategies.

On the basis of research and data analysis on IEEE Xplore PID control theory is the most efficient technique to balance the robot.PID control library is available in the Arduino and all of its function are available on the Arduino website.

2.4 Balancing robot with PID

PID is perhaps the most used controller, as stated by VanDoren "More than 60 years after the introduction of proportional-integral-derivative controllers, they remain the workhorse of industrial process control." The algorithm is described by the following equation:

$$\boldsymbol{u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{d}{dt} e(t)}$$

Where, u(t) is the output of the controller, e(t) is the error and K_p, K_i and K_d are the tuning parameters. It is relatively easy to implement and does not require a model of the system. Tuning of the parameters can be done with trial and error.

2.5 Tilt angle estimation

In order to maintain the robot upright, knowing the tilt angle is imperative. There are a wide array of sensors that can be used, such as inclinometers, light sensors, accelerometer or gyroscopes. However, each these sensors have their shortcomings, the inclinometer takes a long time to converge to the angle it is currently at, light sensors are highly susceptible to background noise (ambient light and the reflective index of the surface it is operating in), gyroscopes have a bias and accelerometers are relatively noisy.

Most often, a combination of a gyroscope and an accelerometer are used.For this purpose a commonly available IMU (Inertial Measurement Unit) MPU 6050 is used.

2.6 PID Control algorithm

The control algorithm that is used to maintain its balance position on the self-balancing two wheel robot was the PID controller. The proportional, integral and derivative (PID) controller is well known as a three term controller. The Proportional Integral Derivative (PID) controller is a control loop feedback mechanism that is widely used in the industry. The controller attempts to adjust and correct the error between the measured process and the desired process and output corrective measures to adjust the process accordingly. This controller must be executed frequently enough and at the same time within the controllable range of the system.

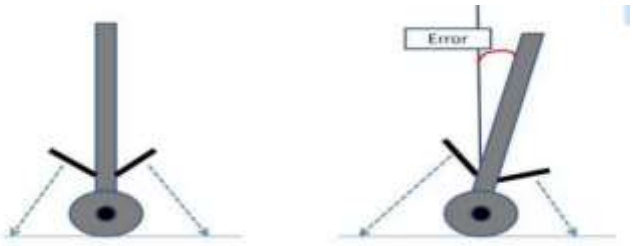


Figure 4: Set point and actual tilt angle of robot

Figure 4 shows the set point and actual tilt angle of the two wheeled robot. The error is the difference between the actual tilt angle and the desired tilt angle (set point). As its name suggests, the PID controller contains of three parts, which are the proportional term, the integral term and the derivative term. These terms have different effect on the response of the DC motor. In order to balance the robot, the set point of the robot must be 0°.

The equation is simplified as below when the robot tilts to the front side as depict in Figure 4.

Error = Current Front Sensor Reading – Front Sensor Setpoint

The equation for the error is illustrated as below when the robot tilt to the back side.

Error = Back Sensor Setpoint – Current Back Sensor Reading

Below are the equations involved in calculating the output PID:

*Output Proportional Term = Kp*Error*

*Output Integral Term = KpKi * Summation of Error * T*

= KpKiT (Summation of Error)*

*·Output Differential Term = Kp*Kd*(Error – Previous Error)/T*

*=(Kp*Kd/T)*(Error – Previous Error)*

The simplification of the formula is as below.

*Output Proportional Term = Kp*Error*

Output Integral Term = Ki (Summation of Error)*

Output Differential Term = Kd(Error – Previous Error)*

Overall, the output PID controller for balancing control system will be:

Output PID controller = Output Proportional Term + Output Integral Term + Output Differential Term

The actual angle is the instantaneous angle of the robot from time to time. This actual angle is measured by the Inertial Measuring Unit (IMU), which produces digital output signals. By comparing with the desired set point, we obtained the error, the difference between desired set point and the actual angle is obtained. The error will then be fed into the PID controller. The PID controller will process, calculate and generate the corresponding speed output to control the DC motor, in order to achieve balance in up right manner.

IMPLEMENTATION

3.1 Hardware Implementation

The design of the hardware system is crucial in bringing the mechanism and software to work together. The main components in the circuit of the balancing robot are the inertial measurement unit (IMU), the Arduino controller and the DC servo motor. Figure 5 shows the overall block diagram of the electronic system for the balancing robot. The IMU is used to measure the acceleration and the angular rate of the robot and the output is processed into digital form. The raw inputs from the IMU are further processed to obtain the tilt angle of the robot. This tilt angle is then fed into the PID controller algorithm to generate the appropriate speed to the DC motor in order to balance the robot.

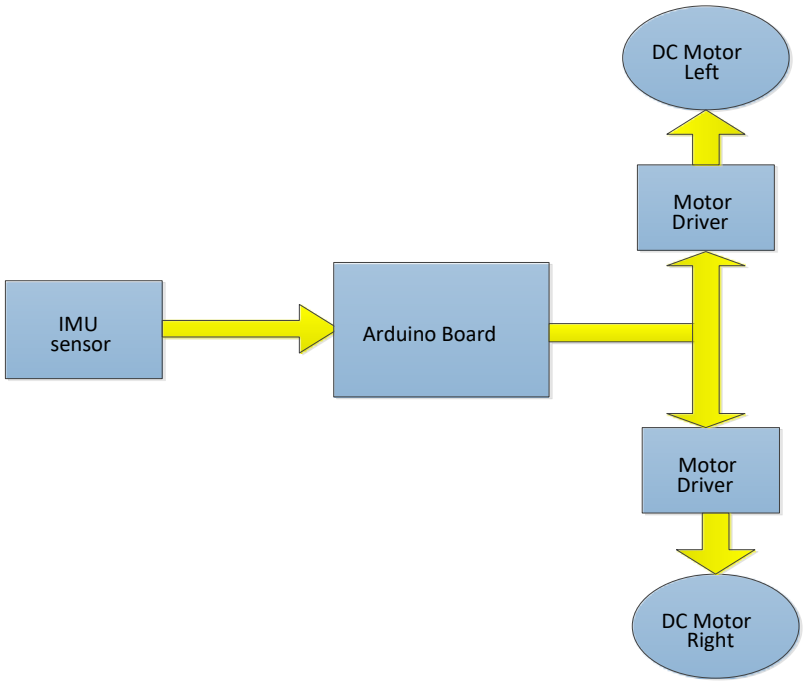


Figure 5: Block diagram of robot.

3.2.1 IMU sensor

In order to obtain the tilt angle of the balancing robot; the Six Degree of Freedom Inertial Measurement Unit (IMU) is used as in Figure 6. The MPU 6050 is a 6 DOF which means that it gives six values as output. The value consists three values from the accelerometer and three from the gyroscope. This chip uses I2C (Inter Integrated Circuit) protocol for communication. The module has on board Digital Motion Processor (DMP) capable of processing complex 9-axis Motion-Fusion algorithms. The SDA and SCL pins are used to establish a connection with the Arduino pins A4 and A5 to receive the accelerometer and gyroscope data. The interrupt pint (INT) is to instruct the Arduino when to read the data from the module and this pin instruct the Arduino only when the values change.

Figure 6: IMU Gyro + Acclerometer MPU 6050

IMU	Advantage	Disadvantage
Accelerometer	No bias	Affected by object's acceleration
Gyroscope	Unaffected by object's acceleration	Accumulated bias

Table 1: Comparison between Accelerometer and Gyroscope in IMU

The biggest advantage of the DMP is that it eliminates the need to perform complex calculations on the Arduino side. The DMP combines the raw sensor data and performs some calculations onboard to minimize the errors in each sensor. Accelerometers and gyros have different inherent limitations when used on their own as indicate in Table 1.

3.2.2 Arduino board

The main controller chosen for the balancing robot is the Arduino Uno as shown in Figure 7. It can be considered as the brain of the balancing robot and is connected to the IMU to process the tilt angle information. After processing, it will communicate with the motor driver in order to adjust the speed and direction of the motor.

Figure 7: Arduino UNO Board

3.2.3 DC Geared motors

Figure 8 shows the DC gear motor used as the actuator of the two wheel balancing robot. The motor is used to generate torque so that the robot could balance itself and stay in the upright position. For this motor, the 10A Dc motor driver is used as the motor controller.

Figure 8: Dc geared motor

For the DC motor used, below are the features of the DC motor:

i. Rated speed is 130rpm. The robot requires an average rpm so that it could counter the balancing error in a suitable speed. Low speed might not be able to balance the robot properly. Therefore, a higher rpm is chosen.

ii. Rated torque is 127.4 mN.m. The torque of the motors must be carefully chosen because a low torque might not be capable to balance the robot. The torque does not necessarily be too high. The torque required is based on the formula, Torque = Force x Distance

3.2.4 Mechanical design

Mechanical structure of robot is build from the following material:

1. Arcylic Sheets
2. Brass male female spacer screws
3. Glue sticks
4. Card board

Figure 9: Spacer Screws , Arcylic Sheet and Glue Sticks.

3.2.5 Circuit Schematic

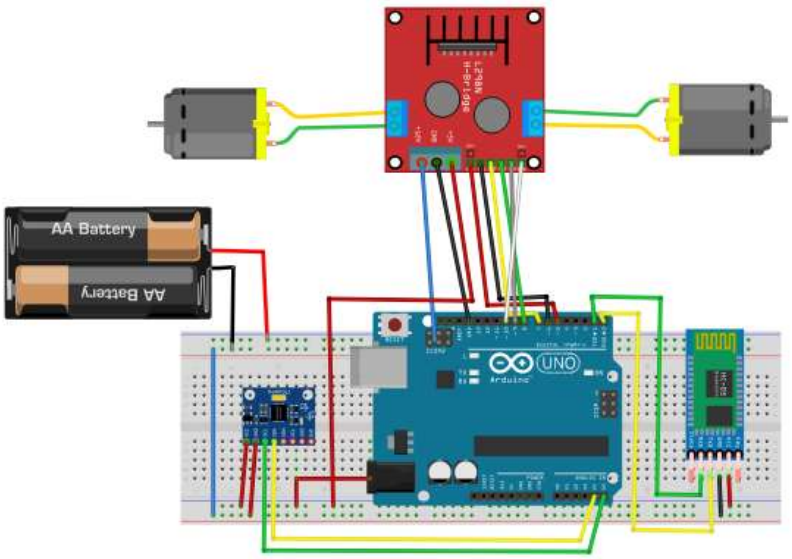
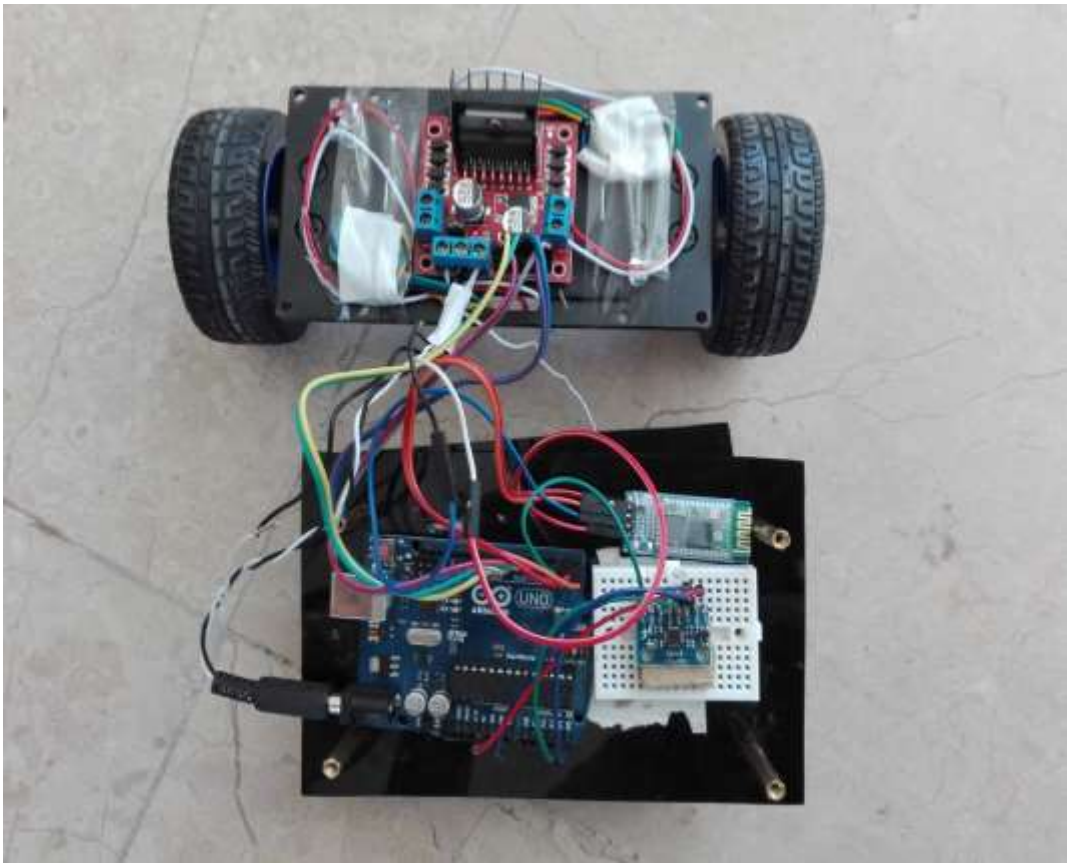


Figure 10: Circuit Schematic of Robot.

3.2.5 Robot Snaps



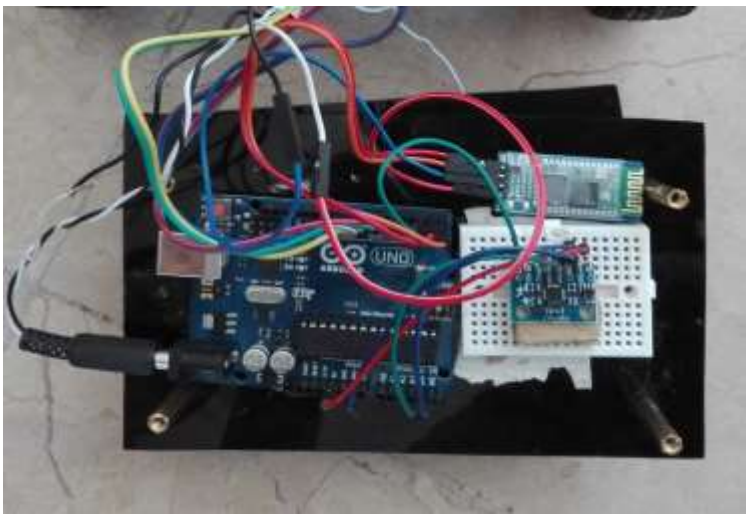


Figure 11: Different views of robots.

3.2.6 Arduino Code and libraries

The code for this project uses following libraries,

1) "PID_v1"
2) "LMotorController"
3) "I2Cdev"
4) "MPU6050_6Axis_MotionApps20"
5) "Wire"

These libraries are attached in zip. file format in separate folder.

The code is given below.

//~~~~~~~~~~~~~~~~~~~ Self Balancing Robot Code ~~~~~~~~~~~~~~~~~~~~//

```
// This code contains Bluetooth interfacing part as well which is selected by defining MOVE_BACK_FORTH high in code.
// Manual tuning for PID constants is done by defining LOG_PID_CONSTANTS high.
#include <PID_v1.h>
#include <LMotorController.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
 #include "Wire.h"
#endif

#define MIN_ABS_SPEED 18

MPU6050 mpu;

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

// orientation/motion vars
Quaternion q; // [w, x, y, z] quaternion container
VectorFloat gravity; // [x, y, z] gravity vector
float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

//PID
double originalSetpoint = 172.50;
double setpoint = originalSetpoint;
double movingAngleOffset = 3.0;
double input, output;
char moveState='0'; //0 = balance; 1 = back; 2 = forth
#define MOVE_BACK_FORTH 0      //~~~~~~~~~~Bluetooth Controlling must be 1~~~~~~~~~~//
#define LOG_PID_CONSTANTS 0    //~~~~~ MANUAL_TUNING must be 1 for Kp, Ki, Kd  ~~~~//
//adjust these values to fit your own design
double Kp = 100.04;
double Kd = 4.0;
double Ki =470.26 ;
double prevKp, prevKi, prevKd;
PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);

double motorSpeedFactorLeft = 0.40;
double motorSpeedFactorRight = 0.39;

//MOTOR CONTROLLER
int ENA = 5;
int IN1 = 6;
int IN2 = 7;
```

```
int IN3 = 9;
int IN4 = 8;
int ENB = 10;
LMotorController motorController(ENA, IN1, IN2, ENB, IN3, IN4, motorSpeedFactorLeft, motorSpeedFactorRight);

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

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

 mpu.initialize();

 devStatus = mpu.dmpInitialize();

 // supply your own gyro offsets here, scaled for min sensitivity
 mpu.setXGyroOffset(220);
 mpu.setYGyroOffset(530);
 mpu.setZGyroOffset(-76);
 mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

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

 // enable Arduino interrupt detection
 attachInterrupt(0, dmpDataReady, RISING);
 mpuIntStatus = mpu.getIntStatus();

 // set our DMP Ready flag so the main loop() function knows it's okay to use it
 dmpReady = true;

 // get expected DMP packet size for later comparison
 packetSize = mpu.dmpGetFIFOPacketSize();

 //setup PID
 pid.SetMode(AUTOMATIC);
 pid.SetSampleTime(10);
 pid.SetOutputLimits(-255, 255);
 }
 else
 {
 // ERROR!
 // 1 = initial memory load failed
 // 2 = DMP configuration updates failed
 // (if it's going to break, usually the code will be 1)
 Serial.print(F("DMP Initialization failed (code "));
 Serial.print(devStatus);
 Serial.println(F(")"));
 }
 Serial.begin(9600);
}
```

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

 // wait for MPU interrupt or extra packet(s) available
 while (!mpuInterrupt && fifoCount < packetSize)
 {
 //no mpu data - performing PID calculations and output to motors

 pid.Compute();
 motorController.move(output, MIN_ABS_SPEED);
 #if LOG_PID_CONSTANTS
   setPIDTuningValues();
 #endif
#if MOVE_BACK_FORTH      ////////////~~~~~~~~~~~~~~~~~~~~~~//////////////
  if(Serial.available())
   moveState=Serial.read();
  moveBackForth();
  if(moveState=='2') motorController.turnRight(MIN_ABS_SPEED*5,1);
  else if(moveState=='4') motorController.turnLeft(MIN_ABS_SPEED*5,1);
  else;
#endif
 }

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

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

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

 // otherwise, check for DMP data ready interrupt (this should happen frequently)
 }
 else if (mpuIntStatus & 0x02)
 {
 // wait for correct available data length, should be a VERY short wait
 while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

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

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

 mpu.dmpGetQuaternion(&q, fifoBuffer);
 mpu.dmpGetGravity(&gravity, &q);
 mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
 input = ypr[1] * 180/M_PI + 180;
 }
}
void moveBackForth()
{
   if (moveState == '1')
     setpoint = originalSetpoint - movingAngleOffset;
   else if (moveState == '3')
```

```
    setpoint = originalSetpoint + movingAngleOffset;
  else
    setpoint = originalSetpoint;
}
void setPIDTuningValues()
{
 int potKp=analogRead(A0);
 int potKi=analogRead(A1);
 int potKd=analogRead(A2);
  Kp = map(potKp, 0, 1023, 0, 25000) / 100.0; //0 - 250
  Ki = map(potKi, 0, 1023, 0, 100000) / 100.0; //0 - 1000
  Kd = map(potKd, 0, 1023, 0, 500) / 100.0; //0 - 5
//   Serial.print("Kp  ");
//   Serial.print(Kp);
//   Serial.print("  Ki  ");
//   Serial.print(Ki);
//   Serial.print("  Kd  ");
//   Serial.println(Kd);
  if (Kp != prevKp || Ki != prevKi || Kd != prevKd)
  {
    pid.SetTunings(Kp, Ki, Kd);
    prevKp = Kp; prevKi = Ki; prevKd = Kd;
  }
}
```

//~~~~~~~~~~~~~~~~~ CODE EndS ~~~~~~~~~~~~~~~~~//

DISCUSSION AND CONCLUSION

During the project a robot has been designed and built from scratch. Mechanically it looks and works as planned. The PID controller was successfully implemented on the robot. The implemented PID controller were also able to handle a bottle of water. The robot is also small and autonomous.The total cost is 5000 /- Rs which is reasonable. It could beshown that the sensors combined could be used for some interaction and also for diferent control actions if movement was successfully implemented. To further improve this work the following suggestions can be done. For the problem with the calculations in Arduino a fixed point arithmetic routine could be writtento improve speed. This must be written from scratch since there currently exist no libraries for the Arduino doing this. To solve the movement without an observer two encoders could be implemented. However then also a small circuit would be required to do the calculations for the encoder counts. When the movement is solved the filtering of the IMU sensor could be improved. This would possibly require to either solve the movement with encoders and a PID or implement fixed point arithmetics to improve calculation speed enough. If the fltering is improved however the sensor could use the advantage of all eight pixels to command the control signal to steer the robot and also use more advanced interaction. Lastly if the movement is solved another plexiglass plane with two more edge detection sensors could be mounted and implemented to allow the robot to move in both directions without risk to fall.