PID Controller based Self-balancing robot

Hemant Kuruva[1]

Electrical and Computer Engineering, Queen's University

hemant.kuruva@queensu.ca

Abstract – This project implements an inverted pendulum concept to build a self-balancing robot running on a single axis. An Inertial Measurement Unit is used to detect the angle of inclination and these values were continuously fed to a PID Controller which would then provide the amount of torque that needs to be applied in the direction of fall to keep the robot balanced. Further, an Ultrasonic sensor is also equipped on-board to detect the obstacles and avert them by changing the direction of the robot accordingly.

I. Introduction

The concept of inverted pendulum is not new to the humans. Notwithstanding the dynamics involved in it, we have always tried to balance a stick on our hand. This is a classic and well-known example of an inverted pendulum. What makes the inverted pendulum idea so important is the plethora of applications it has. Rocket propellers and self-balancing Segway robots are some great examples of Inverted pendulum.

In this project, I measure the tilt obtained from an unstable single-axle robot, use it as the error component, and iteratively try to nullify the tilt error with respect to the ideal vertical position of the robot. The sensor that is used to measure the tilt is MPU6050 Inertial Measurement Unit which has both accelerometer and gyroscope embedded in it. I make use of both accelerometer and gyroscope readings to arrive at an accurate angle of inclination.

The angle of inclination is treated as an error as the vertical position with zero inclination is the desired position and any deviation from it is treated as an error which is to be removed by producing the torque on the motors accordingly. The exact amount of torque that needs to be applied can be calculated by using a PID controller which implements a parallel combination of proportional, derivative, and integral components of the error to obtain a desired torque that can compensate the tilt. Further, an Ultrasonic sensor is used to detect an obstacle and safely turn the direction of the robot so as to avert the collision.

To complement all these, the robot chassis is sketched, modelled, and 3D printed to get the best custom fit and finish.

II. Background

The use of MPU6050 is to retrieve both the accelerometer and gyroscope values of the current position of the robot. The use of both the equipment is justified because of the fact that the accelerometer readings are influenced by sudden horizontal movement and the gyroscope readings gradually drift away from the actual value. By using a filter to amalgamate both these values, we can arrive at a near-perfect estimate.

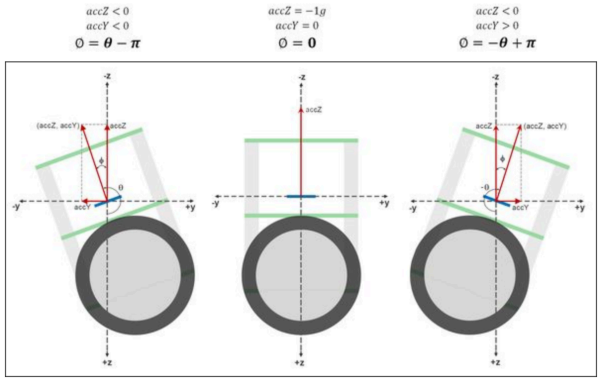

Fig 1: Angle variation during the tilt of the robot

As the gyroscope and accelerometer values are gathered, I passed the gyroscope values through a high pass filter and the accelerometer values through a low pass filter, and combining them to arrive at an inclination value.

Initially, the readings obtained from the IMU have offset errors. These were removed by trial and error method by placing the robot on a flat pedestal and checking the readings obtained from the robot. These errors need to be removed before designing the PID Controller for the robot.

The equation governing the PID Controller is as follows:

$$Output = Kp * e(t) + K_i \int e(t).dt + Kd \frac{d}{dt} e(t)$$

where e(t) is the difference in input and the setpoint. The use of Ultrasonic sensor in this project is just to avert the obstacles. The programming was done in such a way that the robot takes a right turn whenever an obstacle is detected closer than 20 cm. This avoids any damage to the chassis and other crucial equipment.

II. Equipment Used

A. *Hardware*

1. Arduino Mega 2560 microcontroller
2. MPU-6050 Inertial Measurement Unit
3. US-020 Ultrasonic sensor
4. L298N Motor Driver
5. 3 x 9V Batteries (2 for powering the motors, and 1 for powering the Arduino Mega)
6. 16x2 LCD Display
7. Lulzbot Mini 3D printer
8. Orange PLA Filament
9. Trimming Potentiometer
10. Miscellaneous tools

B. *Software*

1. Fritzing software – To build the block diagram
2. Arduino IDE – To program, flash and debug the program on the microcontroller
3. Autodesk Fusion 360 – To make the 3D model of the chassis and extract the stem file.
4. Cura Software – To 3D Print the Chassis.

III. Hardware Design

A. *Block diagram*

Fritzing was the software used to design the following block diagram. All the parts are mentioned with labels and the connections are properly given. The diagram misses out on the power source for the Arduino Mega 2560 due to the fact that it can either be USB Power Supply during the debugging process or a 9V battery during the real-time running.

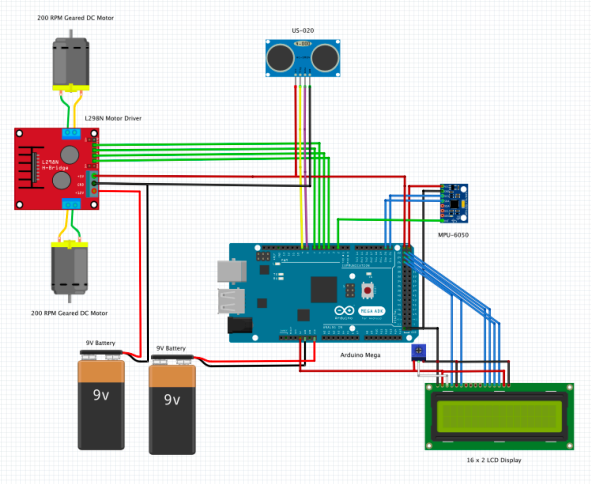

Fig 2: The block diagram showing the project's circuit connection

IV. Software Design

A. *3D Modeling the robot chassis*

All the 3D models were made using the software Autodesk Fusion 360. It is the software which can be used to 3D model either by using parametric modeling technique or freeform modeling technique. In this project, the parametric method was used as the project involves the robot's chassis with high precision and definiteness.

Fig 3: Top panel of the robot chassis

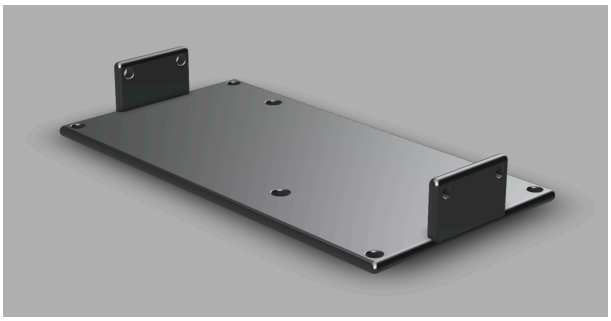

Fig 4: Bottom panel of the robot chassis

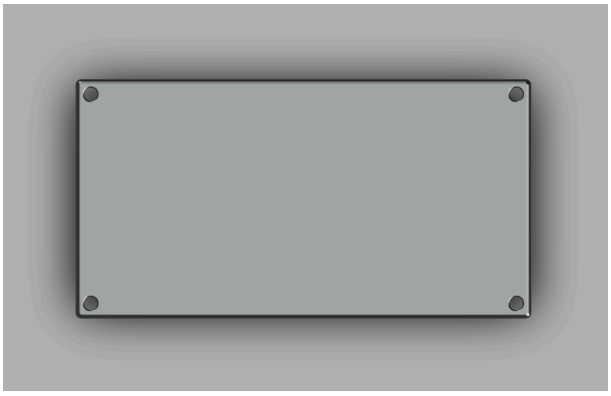

Fig 5: Center panel of the robot chassis

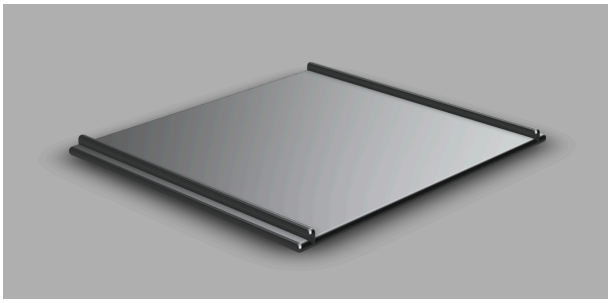

Fig 6: Front panel of the robot chassis

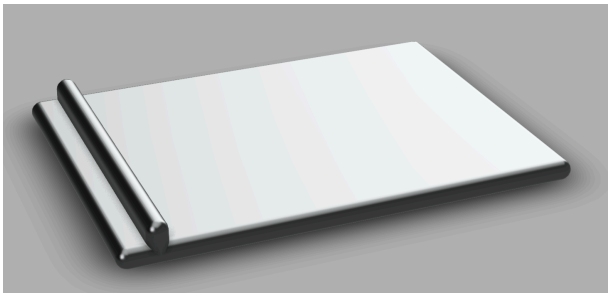

Fig 7: Side panel of the robot chassis

B. Source Code

Please go through the appendix for the Source code of this project.

RESULTS

The robot was built as expected and the performance of the robot was also satisfactory. A few challenges were faced while building the robot such as unusual behaviour from motors, problems with Hitachi LCD driver, and some usual hardware and software issues.

The project was successfully completed with a scope for further improvement. This project can be further improvised in such a way that a remote controller is used to steer the robot in any direction physically possible for the robot. It can also be embedded with additional sensors to perform a specialized task. The chassis can also be further improved to resist the damage incurred by the equipment during the falls due to imbalance.

Following are the photographs of the Self-balancing robot built. A video of its performance was also recorded and will be provided on demand.

Fig 8: Diagonal view of the robot

Fig 9: Front view of the LCD Display

Fig 10: Top view carrying developer's information

APPENDIX

Source Code

```
//          EE503 Course Project            //
//     CODE DEVELOPED BY HEMANT KURUVA      //
//                 20077636                 //
//                                          //
```

```
#include "Wire.h"
#include "math.h"
#include "I2Cdev.h"
#include "MPU6050.h"
#include <NewPing.h>
#include <LiquidCrystal.h>

const int rs = 27, en = 26, d4 = 25, d5 = 24, d6 = 23, d7 = 22;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

#define leftMotorB   6
#define leftMotorA   7
#define rightMotorB  5
#define rightMotorA  4

#define TRIGGER_PIN 9
#define ECHO_PIN 8
#define MAX_DISTANCE 75

#define Kp  60
#define Kd  0.1
#define Ki  10
```

```
#define sampleTime  0.005
#define targetAngle -2.5

MPU6050 mpu;
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

int16_t acceleroY, acceleroZ, gyroX;
volatile int motorPower, gyroRate;
volatile float accAngle, gyroAngle, currentAngle, prevAngle=0, error,
prevError=0, errorSum=0;
volatile byte count=0;
int distanceCm;

void setMotors(int leftMotorSpeed, int rightMotorSpeed) {
  if(leftMotorSpeed >= 0) {
    analogWrite(leftMotorB, leftMotorSpeed);
    digitalWrite(leftMotorA, LOW);
  }
  else {
    /////Direction Reversal
    analogWrite(leftMotorB, 255 + leftMotorSpeed);
    digitalWrite(leftMotorA, HIGH);
  }
  if(rightMotorSpeed >= 0) {
    analogWrite(rightMotorB, rightMotorSpeed);
    digitalWrite(rightMotorA, LOW);
  }
  else {
    /////Direction Reversal
    analogWrite(rightMotorB, 255 + rightMotorSpeed);
    digitalWrite(rightMotorA, HIGH);
  }
}



/////INITIALIZING PID CONTROLLER/////

void init_PID() {
  // initialize Timer1
  cli();          // Disabling the global interrupts
  TCCR1A = 0;     // Setting the Timer/Counter Control Register A to 0
  TCCR1B = 0;     // Setting the Timer/Counter Control Register B to 0
  // Setting Compare Match Register to set sample time 5ms
  OCR1A = 9999;
  // Turn on CTC mode (Clear Timer on Compare mode)
  TCCR1B |= (1 << WGM12);
  // Set CS11 bit for prescaling by 8
  TCCR1B |= (1 << CS11);
  // Enable timer compare interrupt
  TIMSK1 |= (1 << OCIE1A);
  sei();          // Enabling global interrupts
}

void setup() {

  /////INITIALIZING THE LCD DISPLAY/////
```

```
  lcd.begin(16, 2);
  lcd.setCursor(0,0);
  lcd.print("     Hello!     ");
  lcd.setCursor(0,1);
  lcd.print("   Dr. Sydney   ");

  // SETTING THE MOTOR PINS TO OUTPUT MODE/////
  pinMode(leftMotorB, OUTPUT);
  pinMode(leftMotorA, OUTPUT);
  pinMode(rightMotorB, OUTPUT);
  pinMode(rightMotorA, OUTPUT);


  pinMode(13, OUTPUT); // Status LED as Output



  /////INITIALIZING THE MPU6050 AND SETTING THE OFFSET VALUES
  /////I have found out these offset values from repeated tests and checking
the Serial Monitor to know the readings

  mpu.initialize();
  mpu.setYAccelOffset(-630);
  mpu.setZAccelOffset(518);
  mpu.setXGyroOffset(23);
  init_PID(); // Initializing the PID sampling loop
}

void loop() {
  // Reading the acceleration and gyroscope values
  acceleroY = mpu.getAccelerationY();
  acceleroZ = mpu.getAccelerationZ();
  gyroX = mpu.getRotationX();
  motorPower = constrain(motorPower, -255, 255); // Setting the motor power
after constraining it
  setMotors(motorPower, motorPower);
  // Measure distance every 100 milliseconds
  if((count%20) == 0){
    distanceCm = sonar.ping_cm();
  }
  if((distanceCm < 20) && (distanceCm != 0)) {
    setMotors(-motorPower, motorPower); //Changing the direction of the robot
due to the obstacle
  }
}


// The ISR will be called every 5 milliseconds
ISR(TIMER1_COMPA_vect)
{
  // Calculate the angle of inclination
  accAngle = atan2(acceleroY, acceleroZ)*RAD_TO_DEG; //From the mathematical
equation
  gyroRate = map(gyroX, -32768, 32767, -250, 250);
  gyroAngle = (float)gyroRate*sampleTime;
  currentAngle = 0.9934*(prevAngle + gyroAngle) + 0.0066*(accAngle);
//Estimation of the current angle
```

```
  error = currentAngle - targetAngle;
  errorSum = errorSum + error;
  errorSum = constrain(errorSum, -300, 300);
  //Calculating the PID Controller output by using the Kp, Ki, Kd values
  motorPower = Kp*(error) + Ki*(errorSum)*sampleTime - Kd*(currentAngle-
prevAngle)/sampleTime;
  prevAngle = currentAngle;
  count++;

  if(count == 200)  {
    count = 0;
    digitalWrite(13, !digitalRead(13)); // Toggling the LED on pin13 every
second
  }
}
```