**MECT 613 – MICROCONTROLLERS – FALL 2017**

**MIDTERM PROJECT**

SELF-BALANCING ROBOT

|  |  |
| --- | --- |
| **PREPARED BY** | |
| Mohd Saiful Akmal Bin Razali | 171021 |
| Gabriel de Brito Silva | 171020 |
| Hedaya Ali | 171103 |
| Berit Adina Haendel | 171018 |

**Prepared for:**

Prof. Dr. Mahmoud El-Samanty

**SCHOOL OF ENGINEERING & APPLIED SCIENCES (EAS)**

**DATE OF SUBMISSION:**

**‏05‏/12‏/2017**

Table of Contents

[Introduction 3](#_Toc500277277)

[Objective 3](#_Toc500277278)

[1 Theoretical Analysis 3](#_Toc500277279)

[1.1 Inverted pendulum 3](#_Toc500277280)

[2.2 Calculation of angle 5](#_Toc500277281)

[2 Materials and Components 5](#_Toc500277282)

[2.1 Acrylic sheet 5](#_Toc500277283)

[2.2 Microcontroller - Arduino Mega 2560 5](#_Toc500277284)

[2.3 Geared DC Motor with Wheel 6](#_Toc500277285)

[2.4 H-Bridge Module L298N 7](#_Toc500277286)

[2.5 Internal Measurement Unit (IMU) 8](#_Toc500277287)

[2.6 Batteries 8](#_Toc500277288)

[3 Hardware and System Design 9](#_Toc500277289)

[3.1 Frist design approach 12](#_Toc500277290)

[3.3 Reworked: Second design approach 12](#_Toc500277291)

[4 Code 12](#_Toc500277292)

[5 Others 16](#_Toc500277293)

[5.1 Descriptions of attachments 16](#_Toc500277294)

[5.2 Used references 16](#_Toc500277295)

[References 16](#_Toc500277296)

# Introduction

Nowadays, self-balancing robots have been a topic of interest of not only researchers, but also students and hobbyists worldwide. Fundamentally, self-balancing robot is an inverted pendulum on wheels, a derivative of the inverted pendulum on a cart. In comparison with traditional robots which are in a constant state of equilibrium, self-balancing robot is an unstable system in a natural manner [1]. The design itself is more complex due to the necessity of to be actively controlled maintain the upright position. However, self-balancing robot benefits itself from being able to turn on the spot. Commonly, the main practical application of a self-balancing robot is a transportation of human from a location to another location which was introduced and popularised by the release of the Segway Personal Transporter (PT) by Segway Inc. of New Hampshire, USA which is the manufacturer of a two-wheeled, self-balancing electric vehicle [2]. Segway is basically a homophone of segue (a smooth transition, which literally translated from Italian as follow). Lately, it is used in many industries such as inside the factory floors, in the airport and also for recreation in the park. It is more attractive compared to four wheeled vehicles or three wheeled vehicles as it can take sharp turns & navigate in tighter spaces easily [3]. Therefore, self-balancing robot for this particular project can be defined as a two-wheeled vehicle which able to balance itself and automatically correct its position on disturbance.

# Objective

There are several objectives for this particular project which are described as follows:

i. To design a two-wheeled self-balancing robot.

ii. To fabricate a two-wheeled self-balancing robot.

iii. To demonstrate a two-wheeled self-balancing robot that can stay upright.

# 1 Theoretical Analysis

## 1.1 Inverted pendulum

The self-balancing robot can be described as an inverted pendulum. As a simplification we assume that the robot is mainly unstable in rotation around the wheel axis. Therefore, the model becomes 2-diminsional (z- and x-axis) and 2-DOF (position on x-axis and inclination angle).

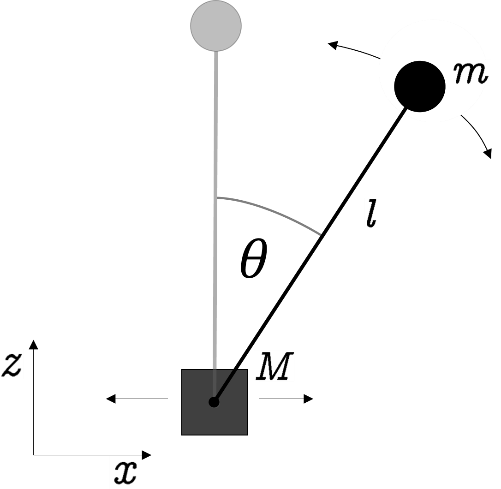
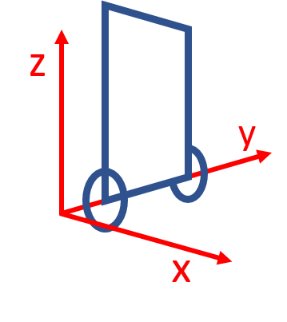


Fig. 1 Coordinate system for self-balancing robot. Relates to a fixed point on the ground. Assuming that robot will not rotate around z-axsis.

The pendulum consists of a mass point *m* at () which is connected to the base’s mass point *M* at (). The pendulum can freely rotate around the joint point and *M* can be move along the *x*-axis. To obtain the equations of motion for the system, we first determine the Lagrange equation consisting of the kinetic energy *T* and the potential energy *V* of the system:

kin. energy of mass m

kin. energy of mass M

rot. energy of mass m

pot. energy of mass m

With the inertia moment of the point mass *m* rotating around point *M*. Since the coordinates are given through

, , ,

we can derivate them

, , ,

and insert the results into the Lagrange equation. Summarized we obtain:

Now, the relevant equation of motion for the system can be obtained by applying the following Euler-Lagrange equation.

damping of the pendulum

As a result, the angular acceleration can be expressed through the angle, the angle velocity and the acceleration in *x*-direction:

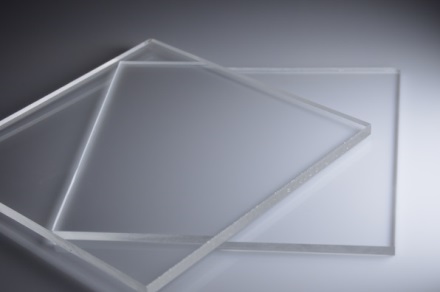
## 2.2 Calculation of angle

# 2 Materials and Components

For this project, we are using several components & materials that are required in order for us to design and fabricate a proper two-wheeled self-balancing robot which described as follows:

## 2.1 Acrylic sheet

We are using Acrylic sheet as the materials to build our chassis. The thickness of the sheet itself is 5mm. The main reason why we choose Acrylic instead of wood or other materials due to the Acrylic is light weight & high impact strength. The analysis of deformation on the chassis of this self-balancing robot is not being carry out as in our opinion, the weight of the all component will not a critical point & main focus of this project.



## 2.2 Microcontroller - Arduino Mega 2560

The microcontroller used in this project is Arduino Mega 2560 which is based on ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. The board has a relatively small size, maintaining the robot as small as possible.



The main advantage of the Arduino is the IDE and the large community. By implementating IDE enables fast software development due to the extensive collection of libraries and sample code. The large community is helpful in the case where a problem is encountered, there is a higher chance that someone else has found a solution and it is visible in one of the many forums available.

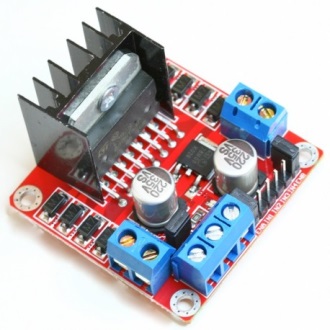
## 2.3 Geared DC Motor with Wheel



Geared DC Motor has been chosen over a DC Motor as Geared DC Motor coupled with a transmission. This Geared DC Motor adds mechanical gears in order to change or alter the speed & torque of the motor for an application. Besides, Geared DC Motor have a better power/weight ratio, greater efficiency and therefor, more compact, robust & reliable. For wheels itself, we have to ensure that the material of the wheels is exceptionally good grip which, make us decided to use a rubber, type of material for it. The specifications of this Geared DC Motor & Wheel are as follows:

|  |  |  |
| --- | --- | --- |
| **No.** | **Description** | **Detail** |
| 1 | Name | * Plastic Tire Wheel With DC 3-6v Gear Motor For Arduino Smart Car |
| 2 | Specification of the Wheel | * Centre hole: 5.3mm x 3.66mm * Wheel size: 65mm x 26mm |
| 3 | Specification of the Geared DC Motor | * Voltage: DC 3V-6V * Current: 100 mA-120mA * Reduction rate: 48: 1 * RPM (With tire): 100-240 * Tire Diameter: 65mm * Motor Weight: 29g * Motor Size: 70mm X 22mm X 18mm * Noise: <65dB |

## 2.4 H-Bridge Module L298N



This motor controller from Tronixlabs Australia is based on the L298N heavy-duty dual H-bridge controller, which can be used to drive two DC motors at up to 2A each, with a voltage between 5 and 35V DC. The controller has fast short-circuited protection diodes, and a nice heatsink to keep the L298N happy. There is also an on-board 5V regulator which useable if we want to use between 7 and 12V DC to drive the motors. The module can also supply an Arduino with 5V DC. The specifications of this L298N H Bridge module are as follows:

|  |  |  |
| --- | --- | --- |
| **No.** | **Description** | **Detail** |
| 1 | Name | * L298N Dual H Bridge DC Motor Drive Controller Board Module For Arduino |
| 2 | Specification | * Double H bridge drive * Chip L298N (ST NEW) * Logical voltage: 5V * Drive voltage: 5V-35V * Logic current: 0mA-36mA * Drive current: 2A(Maximum single bridge) * Storage temperature -20 to +135°C * Maximum power: 25W * Weight: 30g * Size: 43mm x 43mm x 27mm * Compatible with L297/L298 driver |

## 2.5 Internal Measurement Unit (IMU)

The inertial measurement unit (IMU) is very important component in the robot as knowing the tilt angle is critical. IMUs are composed of electromechanical systems (MEMS). MEMS accelerometers and gyroscopes have the advantage of being compact, inexpensive and having low power consumption. They are however less accurate in comparison to optical devices. The InvenSense MPU-6050 sensor contains a MEMS accelerometer and a MEMS gyro in a single chip. It is very accurate, as it contains 16-bits analog to digital conversion hardware for each channel. Therefor it captures the x, y, and z channel at the same time. The sensor uses the I2C-bus to interface with the Arduino.



## 2.6 Batteries



To provide power while maintaining the self-balancing robot, a Lithium Polymer (Li-Po) battery was chosen as the power source. The specification of this battery is as follows:

|  |  |  |
| --- | --- | --- |
| **No.** | **Description** | **Detail** |
| 1 | Name | * Sanyo 18650 Battery |
| 2 | Specification | * Brand Name: For Sanyo * Model: 18650 * Type: Li-Ion * Nominal Capacity: 3400mAh * Nominal Voltage: 3.7V * Color: Red * Weight: 38g * Rechargeable Battery: Yes * Rechargeable Times: Up to 500 times * Size: 65mm x 18mm (Diameter) * Place of Origin: Japan |

# 3 Hardware and System Design

The overall design of this self-balancing robot is shown below. The modular design allowed the layers to be adjusted as needed. The battery is the component with the greatest value of density. The placement of the battery at the top is to make sure that the centre of mass is located above the pivot point. All the other components are placed close to each other to avoid interference that comes with longer wires.

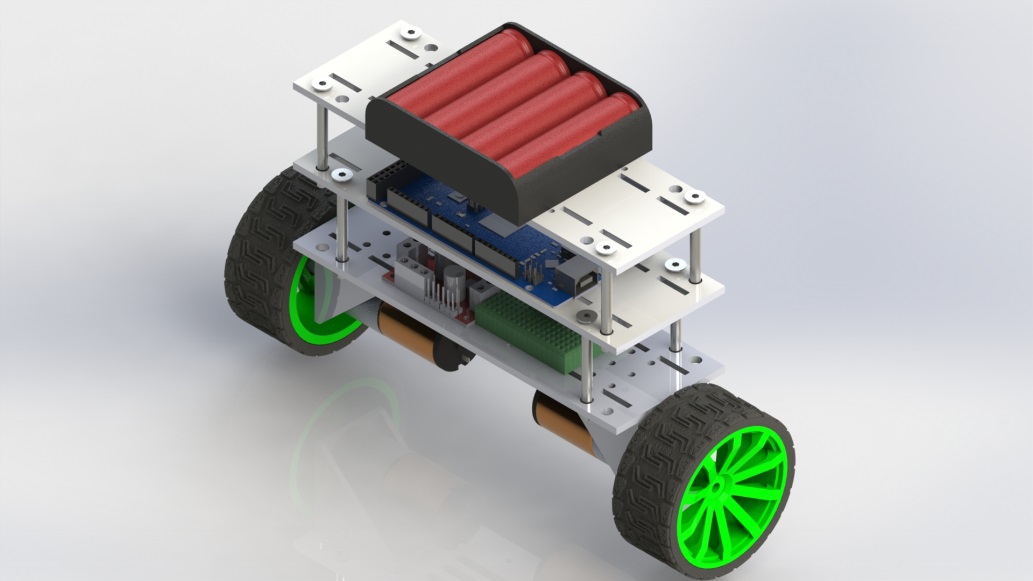


Figure: Overall view of our self-balancing robot.

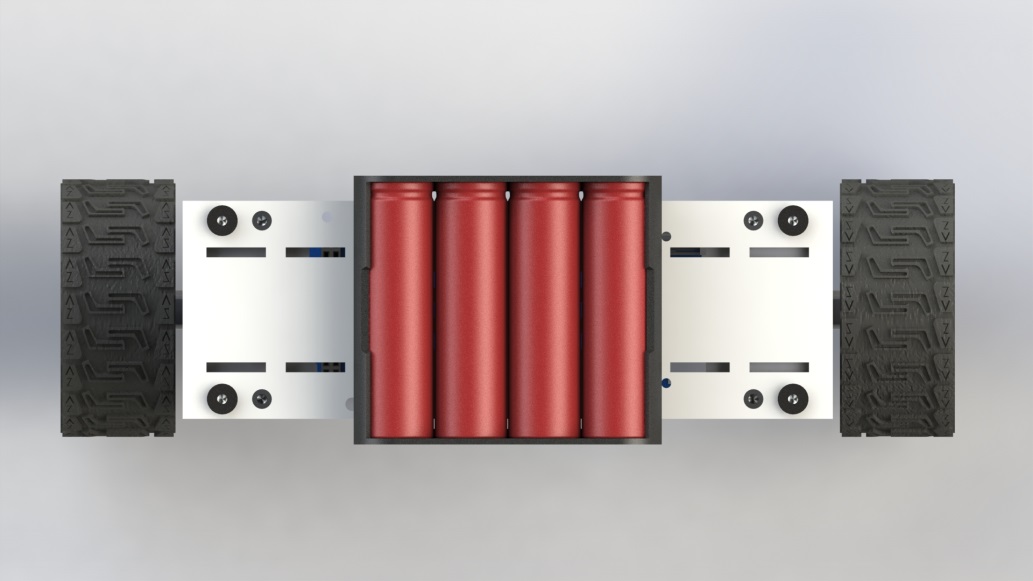


Figure: Top view of our self-balancing robot.

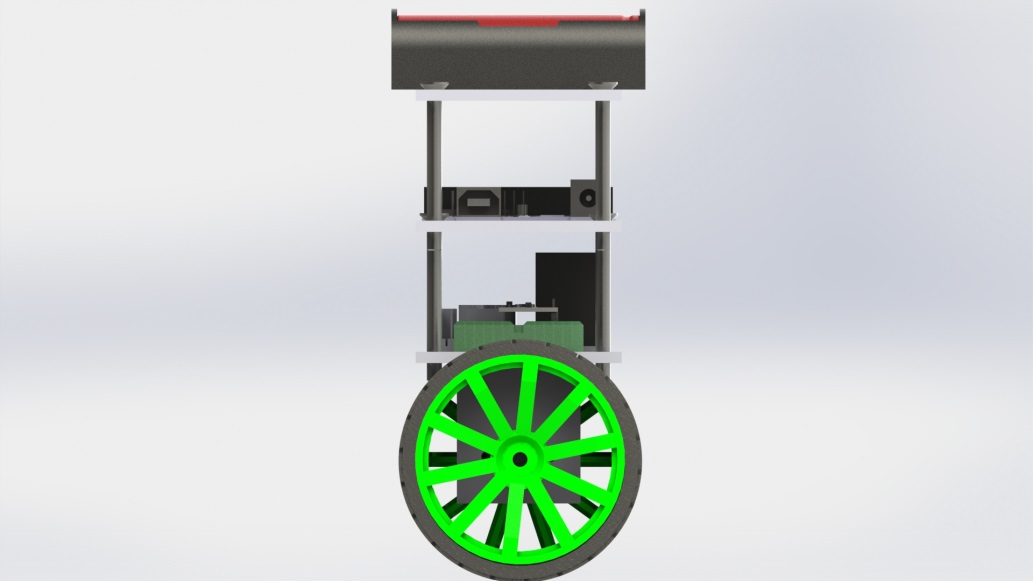


Figure: Side view of our self-balancing robot.

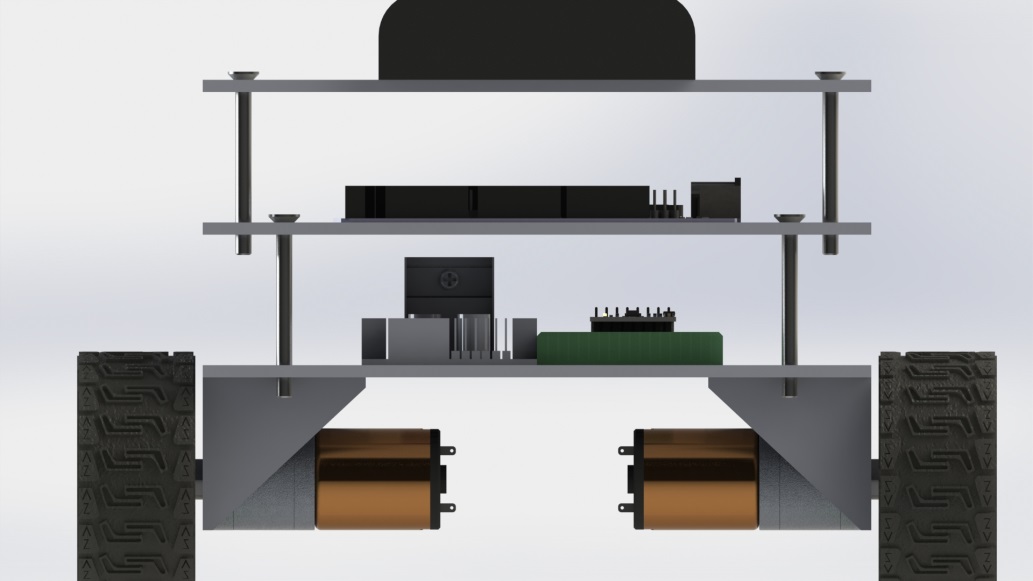
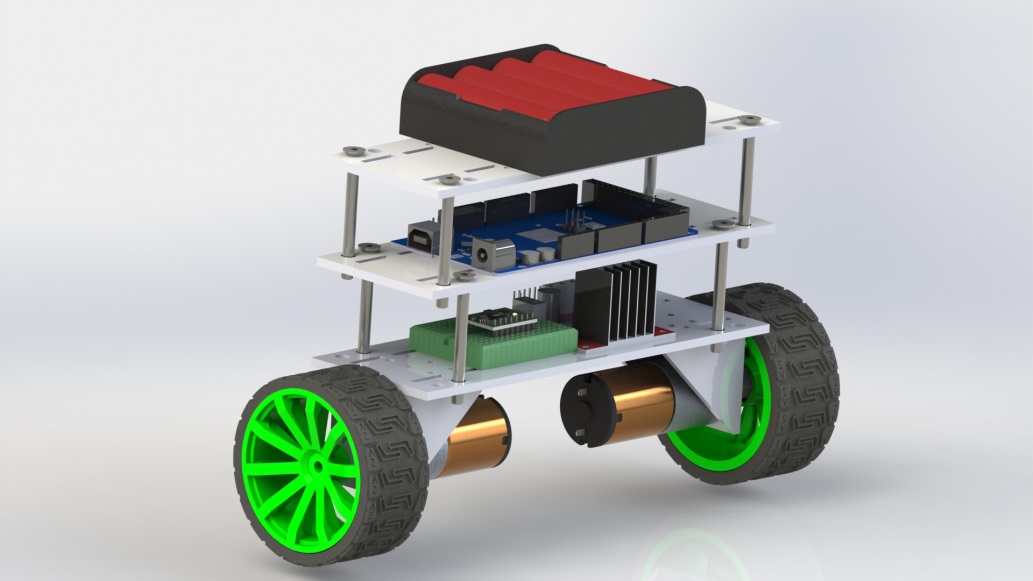
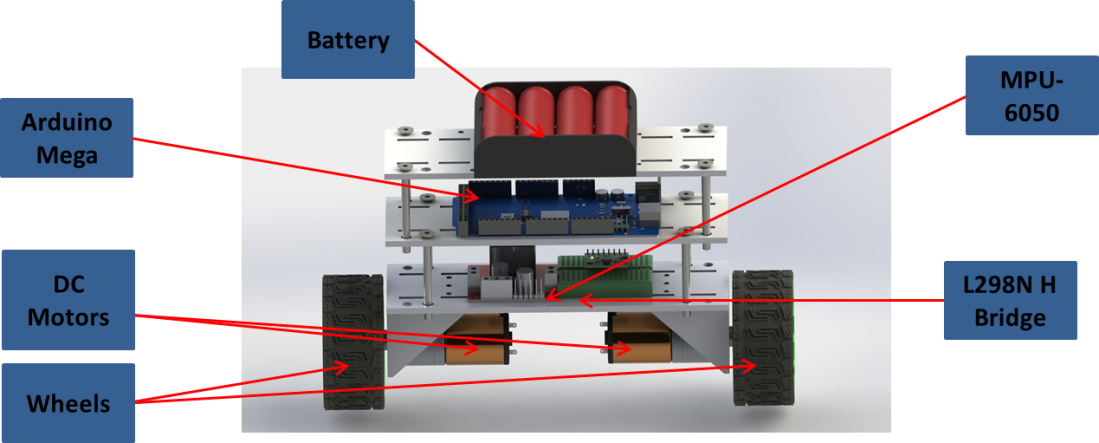


Figure: Front view of our self-balancing robot.



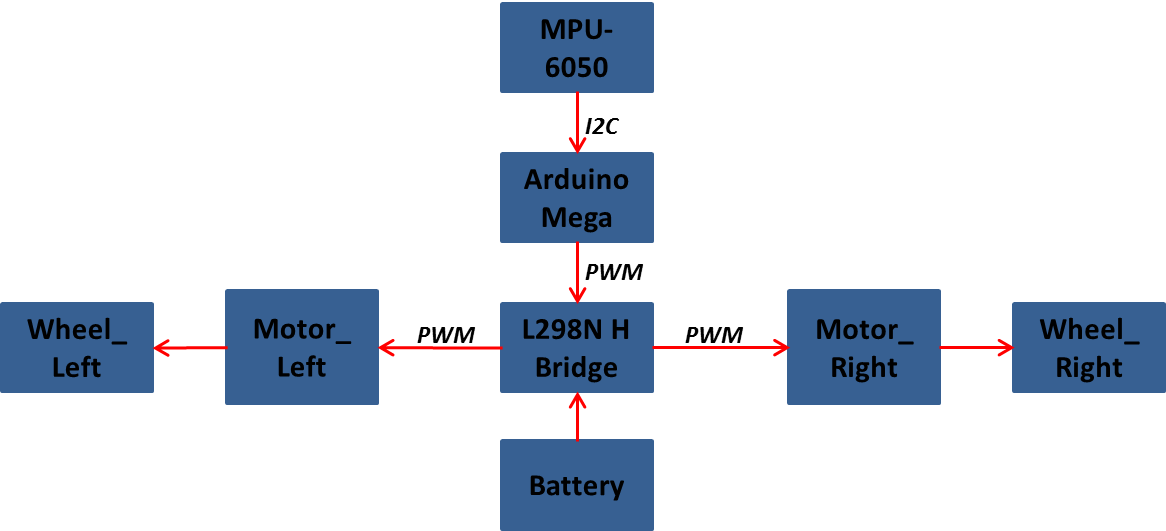


SCHEMATIC DIAGRAM

Below is the schematic diagram of our self-balancing robot with its used hardware & communication. In general, the system can be divided into 3 main sub-system which are microcontroller, measurement sub-system & motor driver sub-system.

For this project, we are using Arduino Mega as the microcontroller. Arduino Mega will read a raw data from the measurement sub-system which in this case is MPU-6050 via I2C protocol. Then, Arduino Mega reads raw data from the measurement sub-system and sends PWM signals to the motor driver to be converted to mechanical motion.

The measurement sub-system consists of MPU-6050 that has a single-axis gyroscope & 2-axis accelerometer for attitude determination. When fresh IMU data is received, the MPU-6050 filters out sensor noise, and then fuse the data from the two sensors together to produce a single reading of tilt. Once the current tilt is known, the MPU-6050 calculates the error from the desired tilt, in this case the vertical, and then uses PID to control the PWM output to the motor driver sub-system. The motor driver subsystem consists of a motor driver, L298N H Bridge connected to two DC gear motors. The l298N H Bridge is powered by 4 Li-Po AA batteries connected in series for a total of 12 volts.

****

## 3.1 Frist design approach

## 3.3 Reworked: Second design approach

# 4 Code

|  |
| --- |
| #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 20 |
| #define DEBUG 0 |
|  |
| 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 = 182; |
| double setpoint = originalSetpoint; |
| double movingAngleOffset = 0.15; |
| double input, output; |
|  |
| //adjust these values to fit your own design |
| double Kp = 55; //60 //32 |
| double Kd = 1.4; //1.4 |
| double Ki = 32; //50 //60 |
| PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT); |
|  |
| double motorSpeedFactorLeft = 0.6; |
| double motorSpeedFactorRight = 0.6; |
| //MOTOR CONTROLLER |
|  |
| int ENA = 44; |
| int IN1 = 51; |
| int IN2 = 52; |
| int IN3 = 49; |
| int IN4 = 48; |
| int ENB = 46; |
|  |
| //int ENA = 3; |
| //int IN1 = 4; |
| //int IN2 = 5; |
| //int IN3 = 8; |
| //int IN4 = 9; |
| //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(76); |
| mpu.setZGyroOffset(-85); |
| 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); |
| 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(")")); |
| } |
| } |
|  |
| 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(); |
|  |
| if(DEBUG == 0) motorController.move(output, MIN\_ABS\_SPEED); |
| } |
|  |
| // 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; |
| } |
| } |

# 5 Others

## 5.1 Descriptions of attachments

# References

|  |  |
| --- | --- |
| [1] | C. a. T. F. Sudin, "Autonomous balancing robot," Chalmers University of Technology, 2013. |
| [2] | G. a. B. G. Welch, "An Introduction to the Kalman Filter," 2001. [Online]. Available: http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001\_CoursePack\_08.pdf. |
| [3] | B. Bonafilia, N. Gustafsson, P. Nyman and S. Nilsson, "Self-balancing two-wheeled robot," Chalmers University of Technology. |
| [4] | Ogata, Modern Control Engineering, Fifth Edition ed., Prentice Hall, 2010. |