Project Name:

Arduino Based Single Wheel Self Balancing Robot

Required Equipments:

1.Arduino Mega 2560

2.L298N Motor Driver Board

3.MPU-6050 Accelerometer and Gyroscope 3-axis Module

4.Gear motor

5.Lipo Battery

6.Wheel

7.Switch

8.Connecting Wire

Description Of Construction:

Controller: The controller that I have used here is Arduino Mega, why because it is simply easy to use.

Motors: The best choice of motor for a self balancing robot, without a doubt will be Stepper motor. But To keep things simple I have used a DC gear motor.

Motor Driver: For DC gear motors we use the L298N driver module

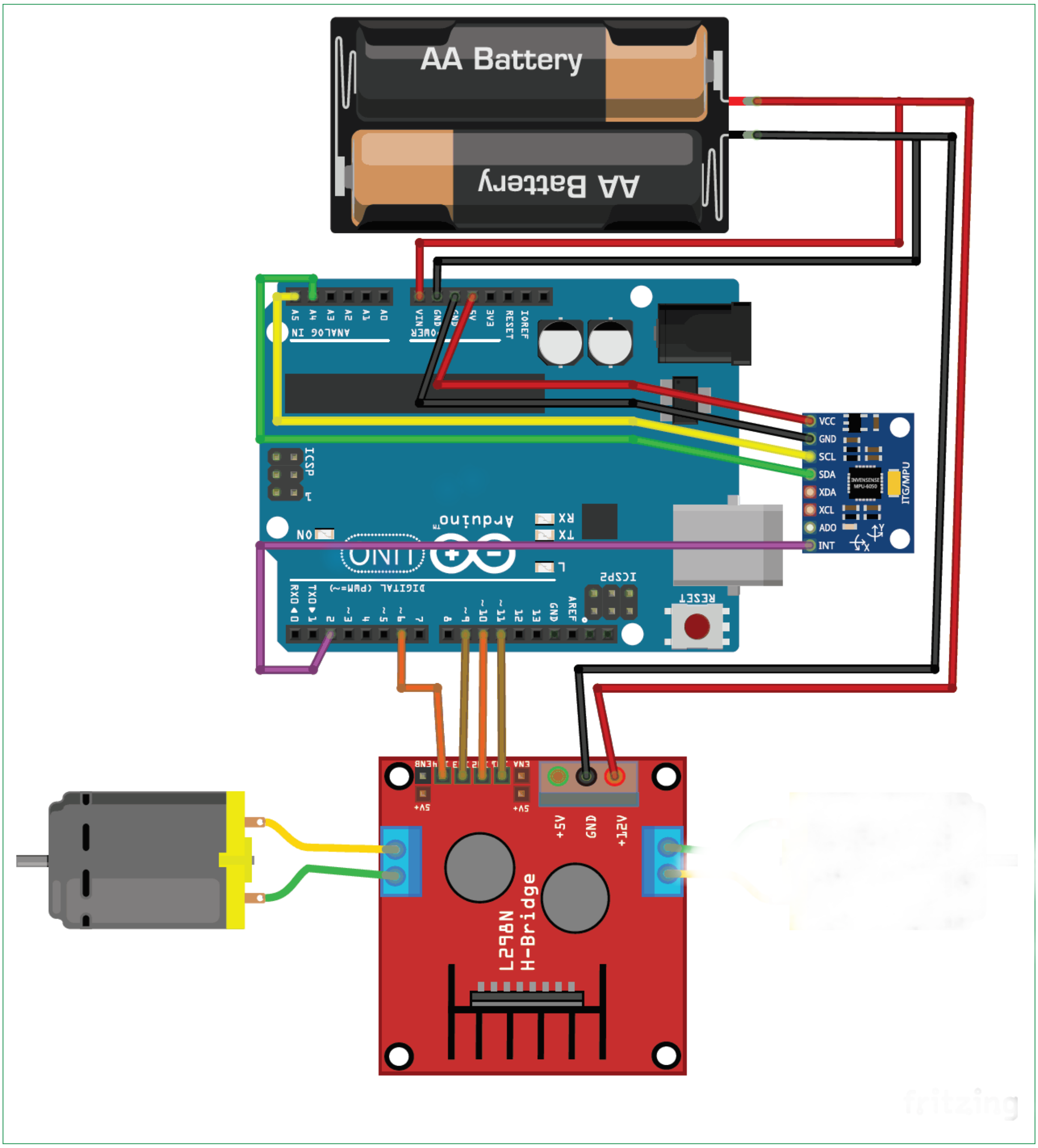
Wheels: We make sure Our wheels have good grip over the floor we are using.

Accelerometer and Gyroscope: The best choice of Accelerometer and Gyroscope we use the MPU6050.

Battery: We need a battery that is as light as possible and the operating voltage should be more than 5V so that we can power our Arduino directly without a boost module. So the ideal choice will be a 11.1V Li-polymer battery.

Chassis: Another place where we should not compromise is with our bots chassis.we use aluminium structured body and we balance the weight of both sides.

CIRCUIT CONNECTION:



Programming the Self Balancing Robot:

Code:

For calibration:

```
#include "I2Cdev.h"

#include "MPU6050.h"


// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation

// is used in I2Cdev.h

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE

    #include "Wire.h"

#endif


// class default I2C address is 0x68

// specific I2C addresses may be passed as a parameter here

// AD0 low = 0x68 (default for InvenSense evaluation board)

// AD0 high = 0x69

MPU6050 accelgyro;

//MPU6050 accelgyro(0x69); // <-- use for AD0 high



const char LBRACKET = '[';

const char RBRACKET = ']';

const char COMMA    = ',';

const char BLANK    = ' ';

const char PERIOD   = '.';
```

```
const int iAx = 0;

const int iAy = 1;

const int iAz = 2;

const int iGx = 3;

const int iGy = 4;

const int iGz = 5;


const int usDelay = 3150;   // empirical, to hold sampling to 200 Hz

const int NFast =  1000;    // the bigger, the better (but slower)

const int NSlow = 10000;    // ..

const int LinesBetweenHeaders = 5;

      int LowValue[6];

      int HighValue[6];

      int Smoothed[6];

      int LowOffset[6];

      int HighOffset[6];

      int Target[6];

      int LinesOut;

      int N;


void ForceHeader()

  { LinesOut = 99; }


void GetSmoothed()

  { int16_t RawValue[6];

    int i;
```

```
  long Sums[6];
  for (i = iAx; i <= iGz; i++)
   { Sums[i] = 0; }
//   unsigned long Start = micros();

  for (i = 1; i <= N; i++)
   { // get sums
     accelgyro.getMotion6(&RawValue[iAx], &RawValue[iAy], &RawValue[iAz],
                          &RawValue[iGx], &RawValue[iGy], &RawValue[iGz]);
     if ((i % 500) == 0)
       Serial.print(PERIOD);
     delayMicroseconds(usDelay);
     for (int j = iAx; j <= iGz; j++)
       Sums[j] = Sums[j] + RawValue[j];
   } // get sums
//   unsigned long usForN = micros() - Start;
//   Serial.print(" reading at ");
//   Serial.print(1000000/((usForN+N/2)/N));
//   Serial.println(" Hz");
  for (i = iAx; i <= iGz; i++)
   { Smoothed[i] = (Sums[i] + N/2) / N ; }
 } // GetSmoothed

void Initialize()
 {
  // join I2C bus (I2Cdev library doesn't do this automatically)
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
```

```
      Wire.begin();
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
      Fastwire::setup(400, true);
    #endif

    Serial.begin(9600);

    // initialize device
    Serial.println("Initializing I2C devices...");
    accelgyro.initialize();

    // verify connection
    Serial.println("Testing device connections...");
    Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection failed");
  } // Initialize

void SetOffsets(int TheOffsets[6])
  { accelgyro.setXAccelOffset(TheOffsets [iAx]);
    accelgyro.setYAccelOffset(TheOffsets [iAy]);
    accelgyro.setZAccelOffset(TheOffsets [iAz]);
    accelgyro.setXGyroOffset (TheOffsets [iGx]);
    accelgyro.setYGyroOffset (TheOffsets [iGy]);
    accelgyro.setZGyroOffset (TheOffsets [iGz]);
  } // SetOffsets

void ShowProgress()
  { if (LinesOut >= LinesBetweenHeaders)
```

```
    { // show header
      Serial.println("\tXAccel\t\t\tYAccel\t\t\t\tZAccel\t\t\tXGyro\t\t\tYGyro\t\t\tZGyro");
      LinesOut = 0;
    } // show header
   Serial.print(BLANK);
   for (int i = iAx; i <= iGz; i++)
    { Serial.print(LBRACKET);
      Serial.print(LowOffset[i]),
      Serial.print(COMMA);
      Serial.print(HighOffset[i]);
      Serial.print("] --> [");
      Serial.print(LowValue[i]);
      Serial.print(COMMA);
      Serial.print(HighValue[i]);
      if (i == iGz)
        { Serial.println(RBRACKET); }
      else
        { Serial.print("]\t"); }
    }
   LinesOut++;
  } // ShowProgress

void PullBracketsIn()
  { boolean AllBracketsNarrow;
   boolean StillWorking;
   int NewOffset[6];
```

```
Serial.println("\nclosing in:");
AllBracketsNarrow = false;
ForceHeader();
StillWorking = true;
while (StillWorking)
 { StillWorking = false;
  if (AllBracketsNarrow && (N == NFast))
   { SetAveraging(NSlow); }
  else
   { AllBracketsNarrow = true; }// tentative
  for (int i = iAx; i <= iGz; i++)
   { if (HighOffset[i] <= (LowOffset[i]+1))
     { NewOffset[i] = LowOffset[i]; }
    else
     { // binary search
      StillWorking = true;
      NewOffset[i] = (LowOffset[i] + HighOffset[i]) / 2;
      if (HighOffset[i] > (LowOffset[i] + 10))
       { AllBracketsNarrow = false; }
     } // binary search
   }
  SetOffsets(NewOffset);
  GetSmoothed();
  for (int i = iAx; i <= iGz; i++)
   { // closing in
    if (Smoothed[i] > Target[i])
     { // use lower half
```

```
          HighOffset[i] = NewOffset[i];
          HighValue[i] = Smoothed[i];
        } // use lower half
      else
        { // use upper half
          LowOffset[i] = NewOffset[i];
          LowValue[i] = Smoothed[i];
        } // use upper half
    } // closing in
    ShowProgress();
  } // still working

} // PullBracketsIn

void PullBracketsOut()
{ boolean Done = false;
  int NextLowOffset[6];
  int NextHighOffset[6];

  Serial.println("expanding:");
  ForceHeader();

  while (!Done)
  { Done = true;
    SetOffsets(LowOffset);
    GetSmoothed();
    for (int i = iAx; i <= iGz; i++)
```

```
  { // got low values
    LowValue[i] = Smoothed[i];
    if (LowValue[i] >= Target[i])
      { Done = false;
        NextLowOffset[i] = LowOffset[i] - 1000;
      }
    else
      { NextLowOffset[i] = LowOffset[i]; }
  } // got low values

SetOffsets(HighOffset);
GetSmoothed();
for (int i = iAx; i <= iGz; i++)
  { // got high values
    HighValue[i] = Smoothed[i];
    if (HighValue[i] <= Target[i])
      { Done = false;
        NextHighOffset[i] = HighOffset[i] + 1000;
      }
    else
      { NextHighOffset[i] = HighOffset[i]; }
  } // got high values
ShowProgress();
for (int i = iAx; i <= iGz; i++)
  { LowOffset[i] = NextLowOffset[i];   // had to wait until ShowProgress done
    HighOffset[i] = NextHighOffset[i]; // ..
  }
```

```
   } // keep going
  } // PullBracketsOut

void SetAveraging(int NewN)
  { N = NewN;
    Serial.print("averaging ");
    Serial.print(N);
    Serial.println(" readings each time");
   } // SetAveraging

void setup()
  { Initialize();
    for (int i = iAx; i <= iGz; i++)
      { // set targets and initial guesses
        Target[i] = 0; // must fix for ZAccel
        HighOffset[i] = 0;
        LowOffset[i] = 0;
      } // set targets and initial guesses
    Target[iAz] = 16384;
    SetAveraging(NFast);

    PullBracketsOut();
    PullBracketsIn();

    Serial.println("-------------- done --------------");
  } // setup
```

FOR BALANCE THE ROBOT:

Main code:

```
#include "I2Cdev.h"

#include <PID_v1.h>

#include "MPU6050_6Axis_MotionApps20.h"

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




/*********Tune these 4 values for your BOT*********/

double setpoint= 176; //set the value when the bot is perpendicular to ground using serial monitor.

//Read the project documentation on circuitdigest.com to learn how to set these values

double Kp = 21; //Set this first
```

```
double Kd = 0.8; //Set this secound

double Ki = 140; //Finally set this

/******End of values setting*********/


double input, output;

PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);




volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high

void dmpDataReady()

{

   mpuInterrupt = true;

}


void setup() {

  Serial.begin(115200);


  // initialize device

    Serial.println(F("Initializing I2C devices..."));

    mpu.initialize();


     // verify connection

    Serial.println(F("Testing device connections..."));

    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050
connection failed"));


    // load and configure the DMP
```

```
devStatus = mpu.dmpInitialize();



// supply your own gyro offsets here, scaled for min sensitivity
mpu.setXGyroOffset(29);
mpu.setYGyroOffset(21);
mpu.setZGyroOffset(13);
mpu.setZAccelOffset(1761);

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

  // enable Arduino interrupt detection
  Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
  attachInterrupt(0, dmpDataReady, RISING);
  mpuIntStatus = mpu.getIntStatus();

  // set our DMP Ready flag so the main loop() function knows it's okay to use it
  Serial.println(F("DMP ready! Waiting for first interrupt..."));
  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(")"));
  }

//Initialise the Motor outpu pins
  pinMode (6, OUTPUT);
  pinMode (9, OUTPUT);
  pinMode (5, OUTPUT);


//By default turn off both the motors
  analogWrite(6,LOW);
  analogWrite(9,LOW);
```

```
}



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();

    //Print the value of Input and Output on serial monitor to check how it is working.
    Serial.print(input); Serial.print(" =>"); Serial.println(output);

    if (input>150 && input<200){//If the Bot is falling

    if (output>0) //Falling towards front
    Forward(); //Rotate the wheels forward
    else if (output<0) //Falling towards back
    Reverse(); //Rotate the wheels backward
    }
    else //If Bot not falling
    Stop(); //Hold the wheels still
```

```
}

// 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); //get value for q
        mpu.dmpGetGravity(&gravity, &q); //get value for gravity
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity); //get value for ypr

        input = ypr[1] * 180/M_PI + 180;

   }
}

void Forward() //Code to rotate the wheel forward
{
    analogWrite(5,output);
    digitalWrite(6,HIGH);
    digitalWrite(9,LOW);

    Serial.print("F"); //Debugging information
}

void Reverse() //Code to rotate the wheel Backward
{
    analogWrite (5, (output)*(-1));
     digitalWrite(6,LOW);
    digitalWrite(9,HIGH);
```

```
   Serial.print("R");
}

void Stop() //Code to stop both the wheels
{
  analogWrite(5,0);
   digitalWrite(6,LOW);
   digitalWrite(9,LOW);
   Serial.print("S");
}
```

COST ESTIMATION:

Product Name	Price(taka)
1.Arduino Mega	800
2.l298N Motor Driver	300
3.MPU-6050	250
4.Gear motor	500
5.wheel	200
6.Lipo Battery(1000mah)	850
7.Structure	250
9.Connecting Wire	80
Total	3230