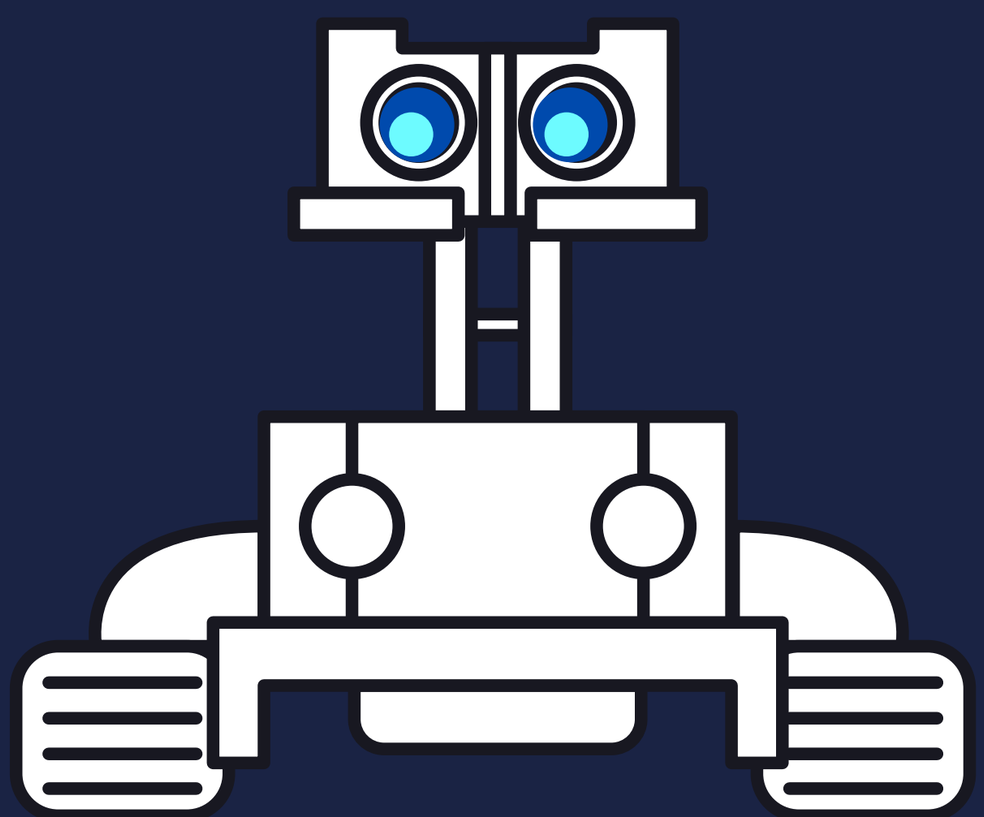
PROJECT REPORT ON

SELF BALANCING ROBOT



FROM ERROR TO ERROR ONE DISCOVER ENTIRE TRUTH

ABSTRUCT

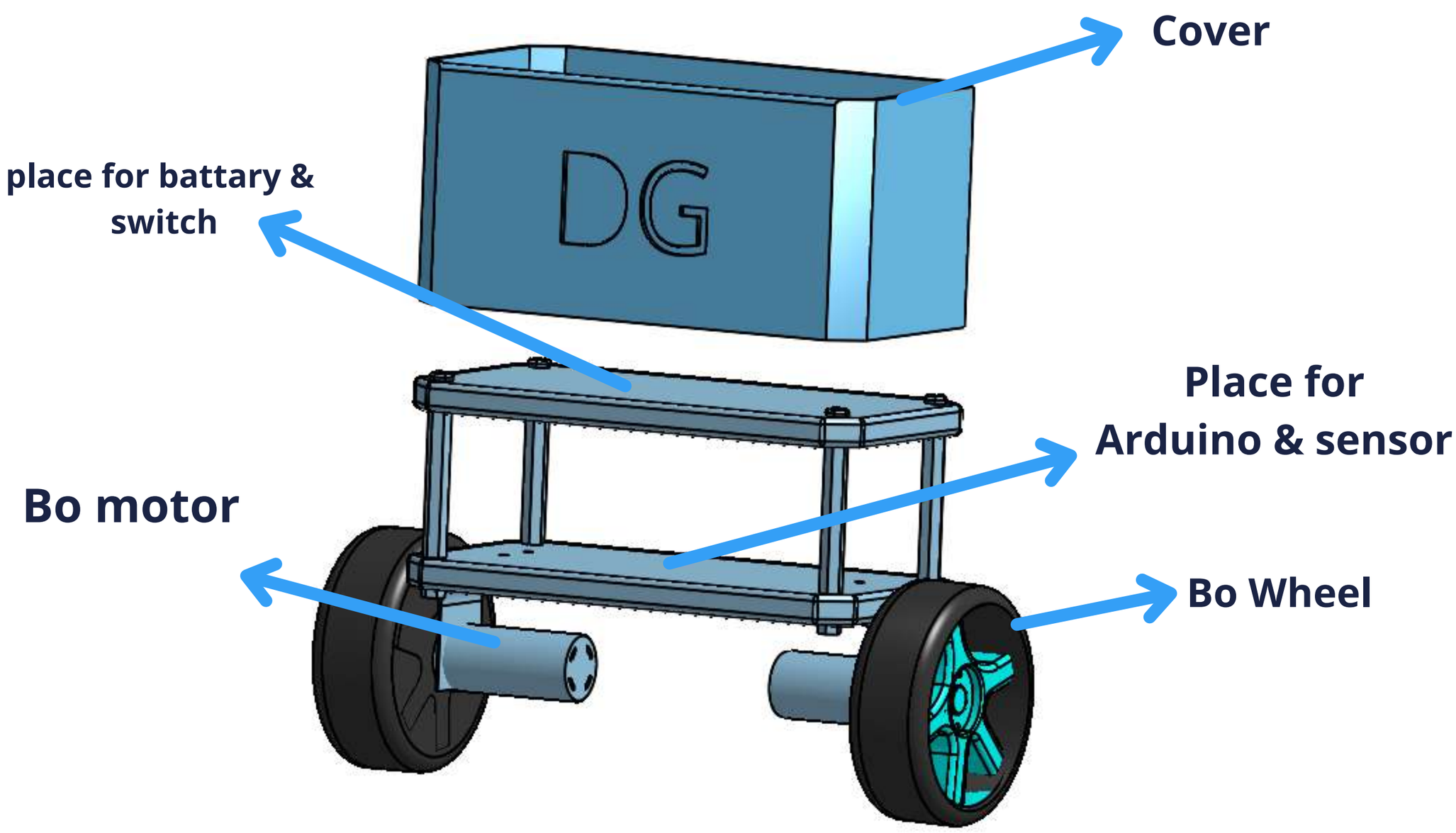
The researches on two-wheeled self-balancing robots have gained momentum in recent decade around the world. This report describes the kinematics model of a two-wheeled self-balancing robot. After mechatronics system design of the robot is completed, the analysis of the whole kinematics model can be divided into two wheels and a body. Then the velocity decompositions of robot wheel and body are analyzed respectively. After the left and right wheel kinematic model is established based on above velocity decomposition method, and the kinematic model of self-balancing robot's body is also calculated by this method. The whole kinematics model of the two-wheeled self-balancing robot system is then established. The effectiveness of the kinematics model is testified by the simulation analysis on ADAMS and experimental validation

INDEX

INTRODUCTION

BACKGROUND :

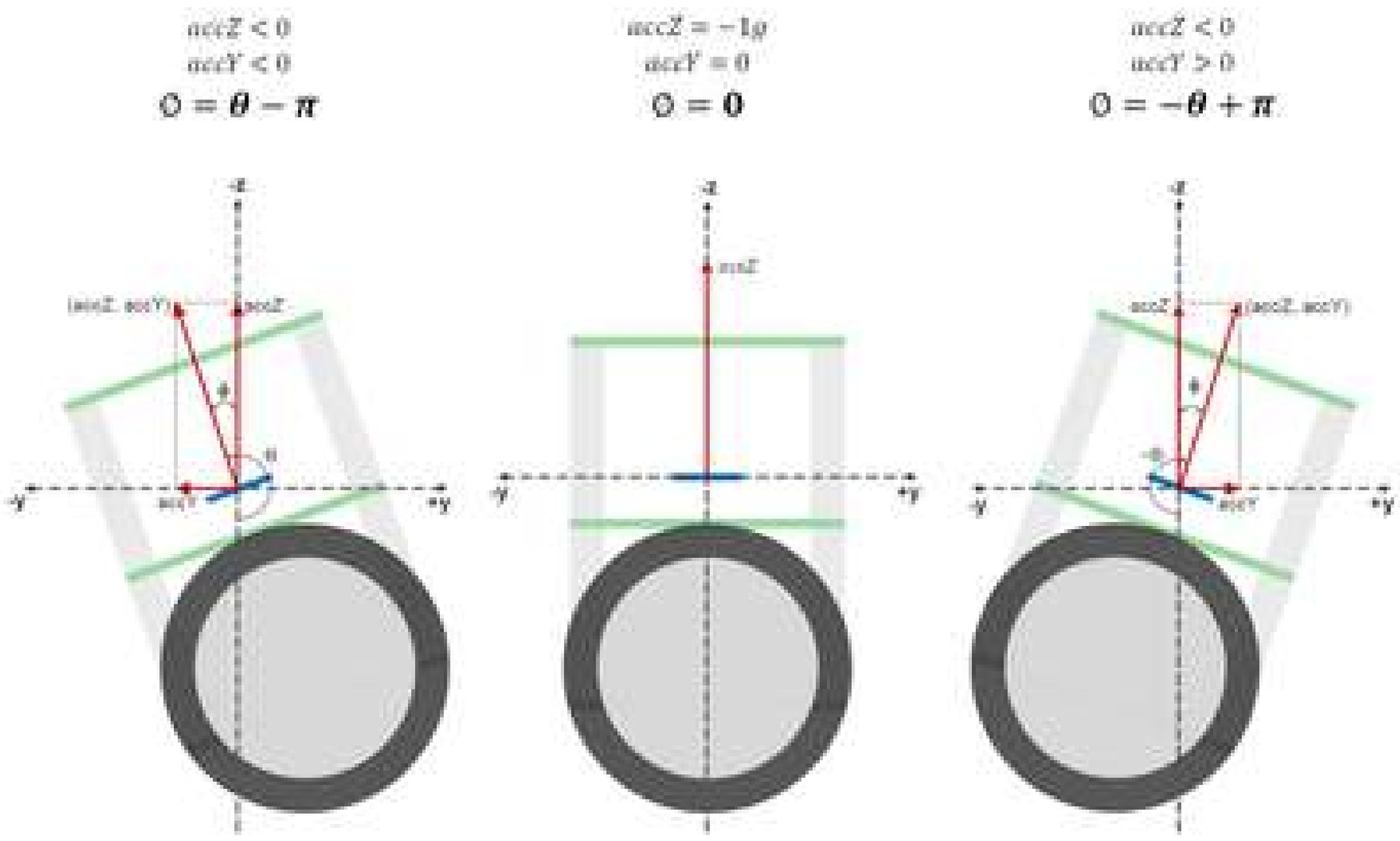
Inverted pendulum applications are plenty; for example the human body is an inverted pendulum balancing the upper body around our ankle joints in every step. In the recent decade Segways, making use of bodily movements for steering, have emerged on the market. In common, these applications share that their centre of mass is located above their pivot points, thus requiring active control in order to balance. [9] The open source community is full of instructions and code snippets, many making use of the open source micro controller Arduino for control algorithms. An example is the open source kit Balanduino as seen in Figure



In order to balance a two-wheeled inverted pendulum robot it is necessary to have accurate information of the current tilt angle from using a measurement unit. Furthermore a controller needs to be implemented to compensate for said tilt. A PID-controller is able to control the pendulum angle, since it is a SISO - SingleInput Single-Output system. If the robot should be able to be controlled in regard to position, x, as well as the angle, it is a MIMO Multiple-Input Multiple-Output system and one PID-controller is not enough. Controlling multiple states is conveniently made through a state space controller.

PURPOSE :

Prepare the demonstrator for a state space controller to be implemented, that can control the angle deviation, ψ and position, x. See Figure . A state space description is based on a model of reality. Several assumptions and simplifications must be made. The core part of this project will be to create a model for that purpose and validate whether the model is good enough for implementation of a state space control, this will be done by answering the question: Using a manually tuned PID-controller, what conclusions can be made regarding the reliability of the model? From answering this question it can be concluded if the model is accurate enough for future state space control.

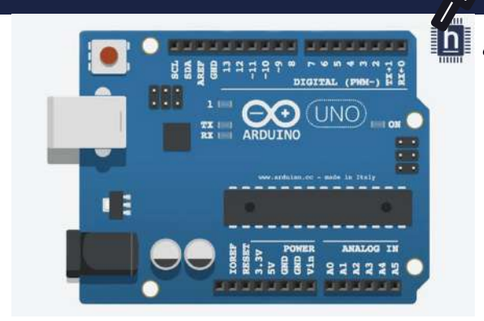


SCOPE :

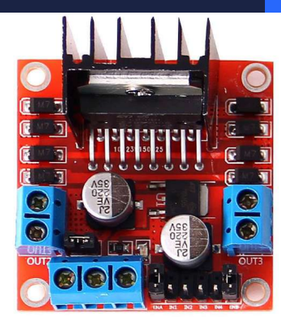
The process of making a balancing robot is widely documented and open-source code is available. In some parts of this project open-source code has been used and modified to fit the hardware. The given project requirements involved using the micro controller board Arduino Uno. The robot will only be run and tested indoors on flat surfaces. The pendulum is assumed to have one degree of freedom. It will therefore only be controlled in one direction, regarding both the angle of tilt and the position of the robot.

• LIST OF COMPONENT

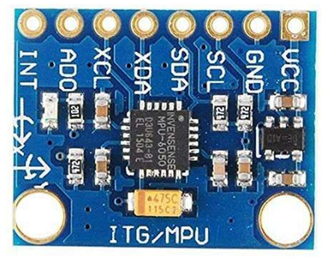
ARDUINO UNO :



L298N MOTOR DRIVER :



MPU6050 ,ACCELROMETER AND GYROSCOPE

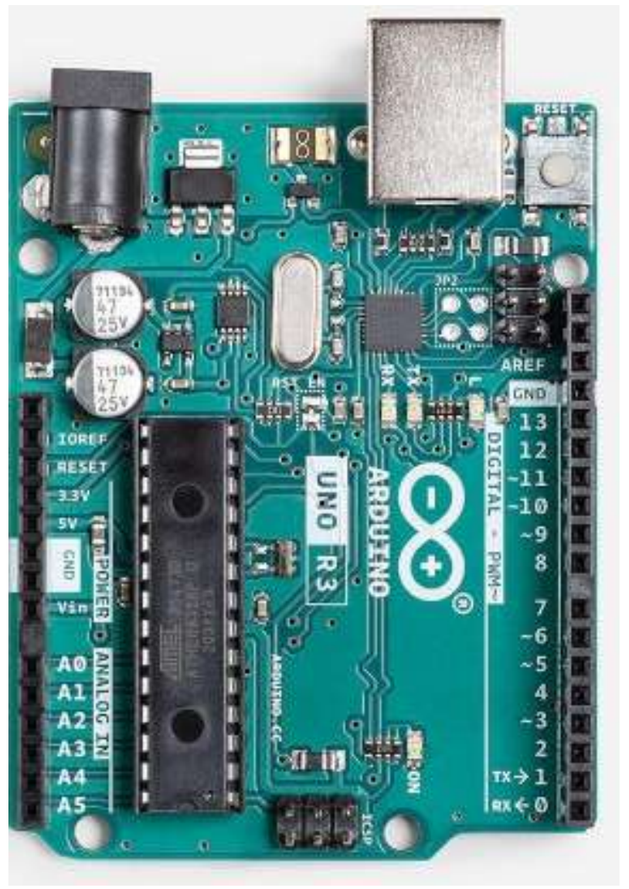


BO MOTOR AND BO WHEEL

BODY PARTS

COMPONENT DISCREAPTION

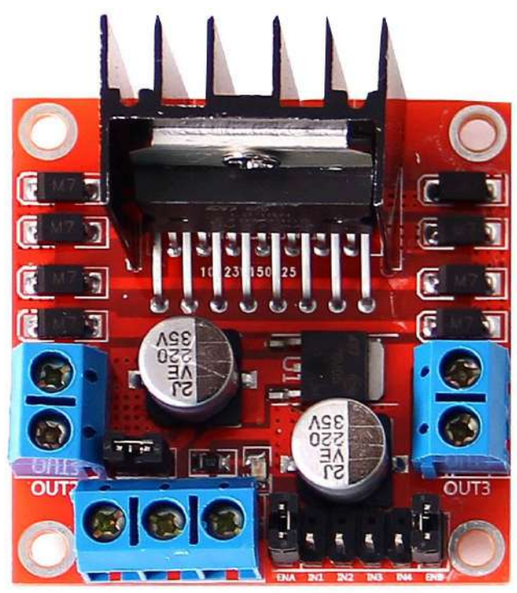
ARDUINO UNO :



Arduino Uno is a microcontroller board based on the ATmega328P (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started..

"Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform; for an extensive list of current, past or outdated boards see the Arduino index of boards.

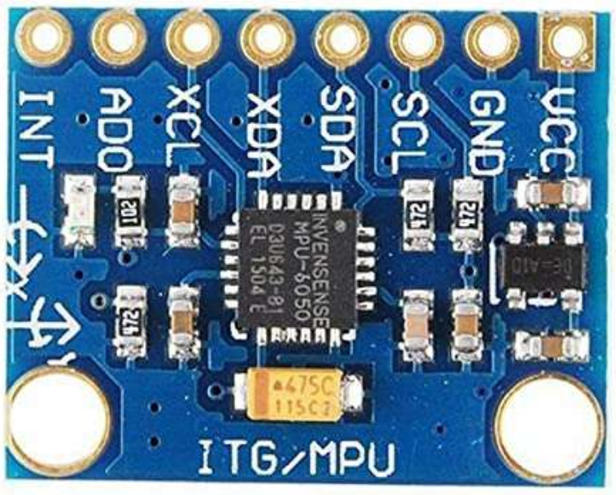
L298N MOTOR DRIVER :



The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

This depends on the voltage used at the motors VCC. The module have an onboard 5V regulator which is either enabled or disabled using a jumper. If the motor supply voltage is up to 12V we can enable the 5V regulator and the 5V pin can be used as output, for example for powering our Arduino board. But if the motor voltage is greater than 12V we must disconnect the jumper because those voltages will cause damage to the onboard 5V regulator. In this case the 5V pin will be used as input as we need connect it to a 5V power supply in order the IC to work properly.

MPU 6050 , 3 AXIS ACCELEROMETER AND GYROSCOPE :



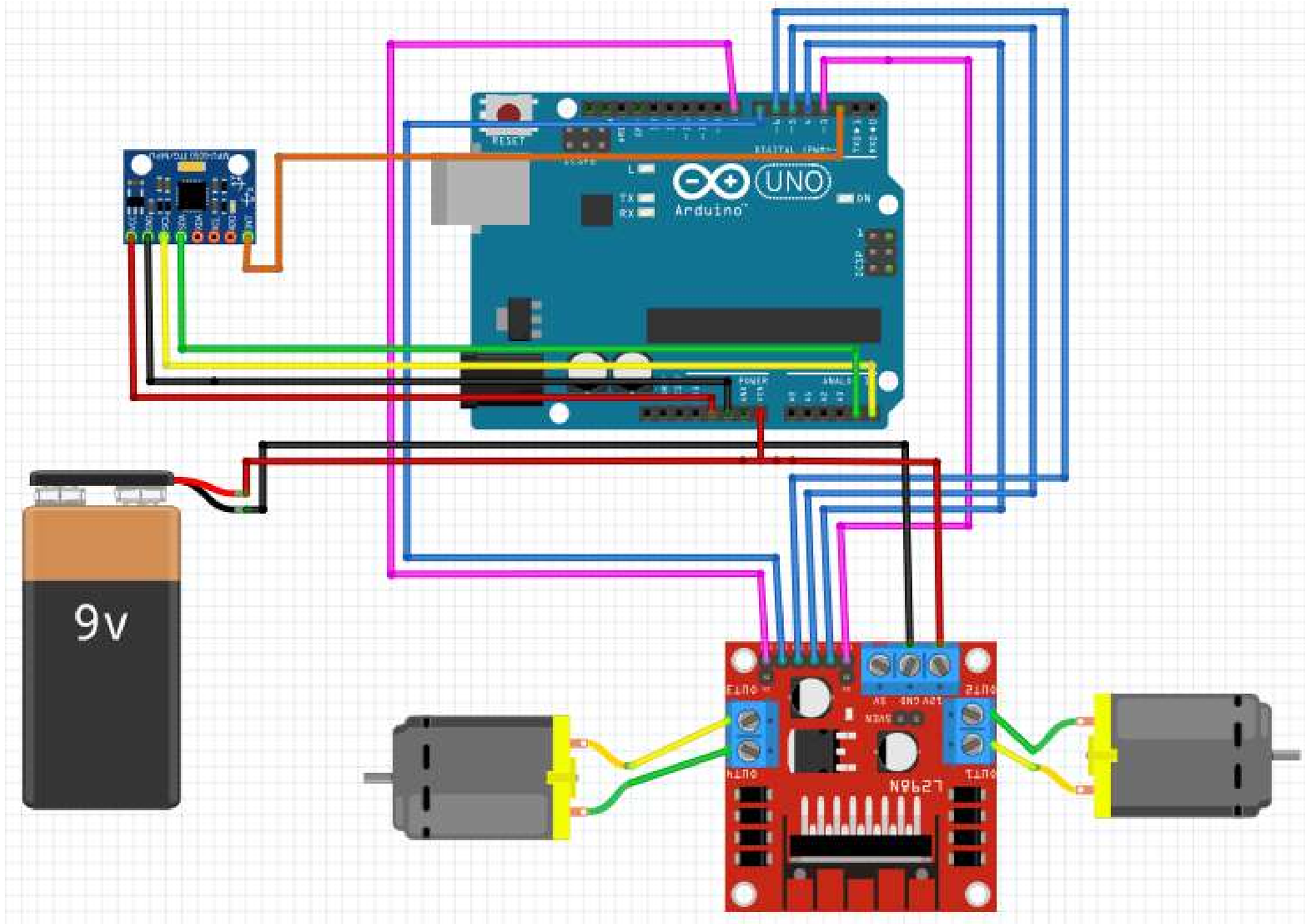
MPU6050 is basically a sensor for motion processing devices. It is the world first six dimension motions tracking device. It was designed for low cost and high performances smartphones, tablets and wearable sensor. It is capable of processing nine-axis algorithms, it captures motion in X, Y and Z axis at the same time.MPU6050 is used in different industrial projects and electronic devices to control and detect the 3-D motion of different objects. In today's post, we will have a look at its working, pinout, protocol, it's interfacing with Arduino, features, applications,

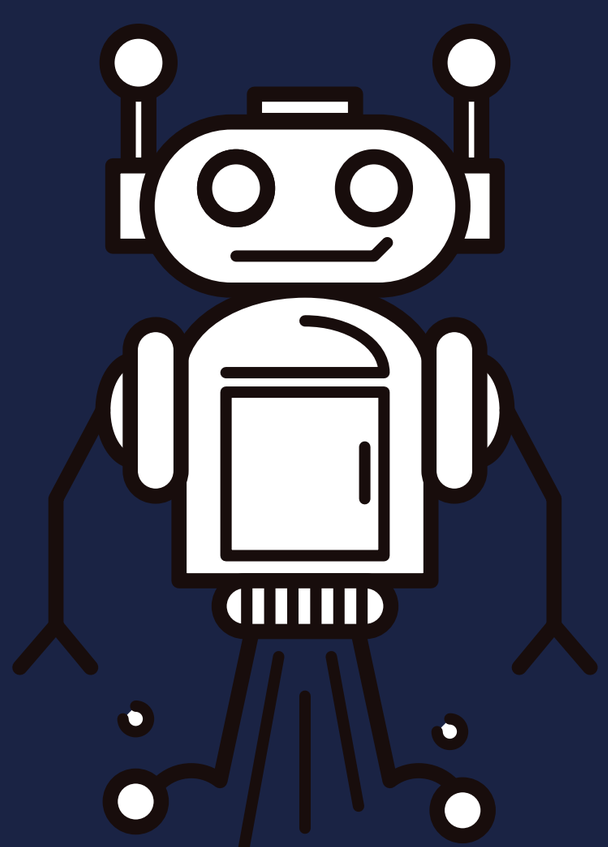
Pin Number	Pin Name	Description
1	Vcc	Provides power for the module, can be +3V to +5V. Typically +5V is used
2	Ground	Connected to Ground of system
3	Serial Clock (SCL)	Used for providing clock pulse for I2C Communication
4	Serial Data (SDA)	Used for transferring Data through I2C communication
5	Auxiliary Serial Data (XDA)	Can be used to interface other I2C modules with MPU6050. It is optional
6	Auxiliary Serial Clock (XCL)	Can be used to interface other I2C modules with MPU6050. It is optional
7	AD0	If more than one MPU6050 is used a single MCU, then this pin can be used to vary the address
8	Interrupt (INT)	Interrupt pin to indicate that data is available for MCU to read.

BO MOTOR AND BO WHEEL :

BO (Battery Operated) light weight DC geared motor which gives good torque and rpm at lower voltages. and bo wheel is simply made by pastic , it only fit in bo motor

CIRCUIT DIAGRAM





BLOCK DIAGRAM

BO MOTOR (LEFT)

L298 MOTOR DRIVER MODULE

BO MOTOR (RIGHT)

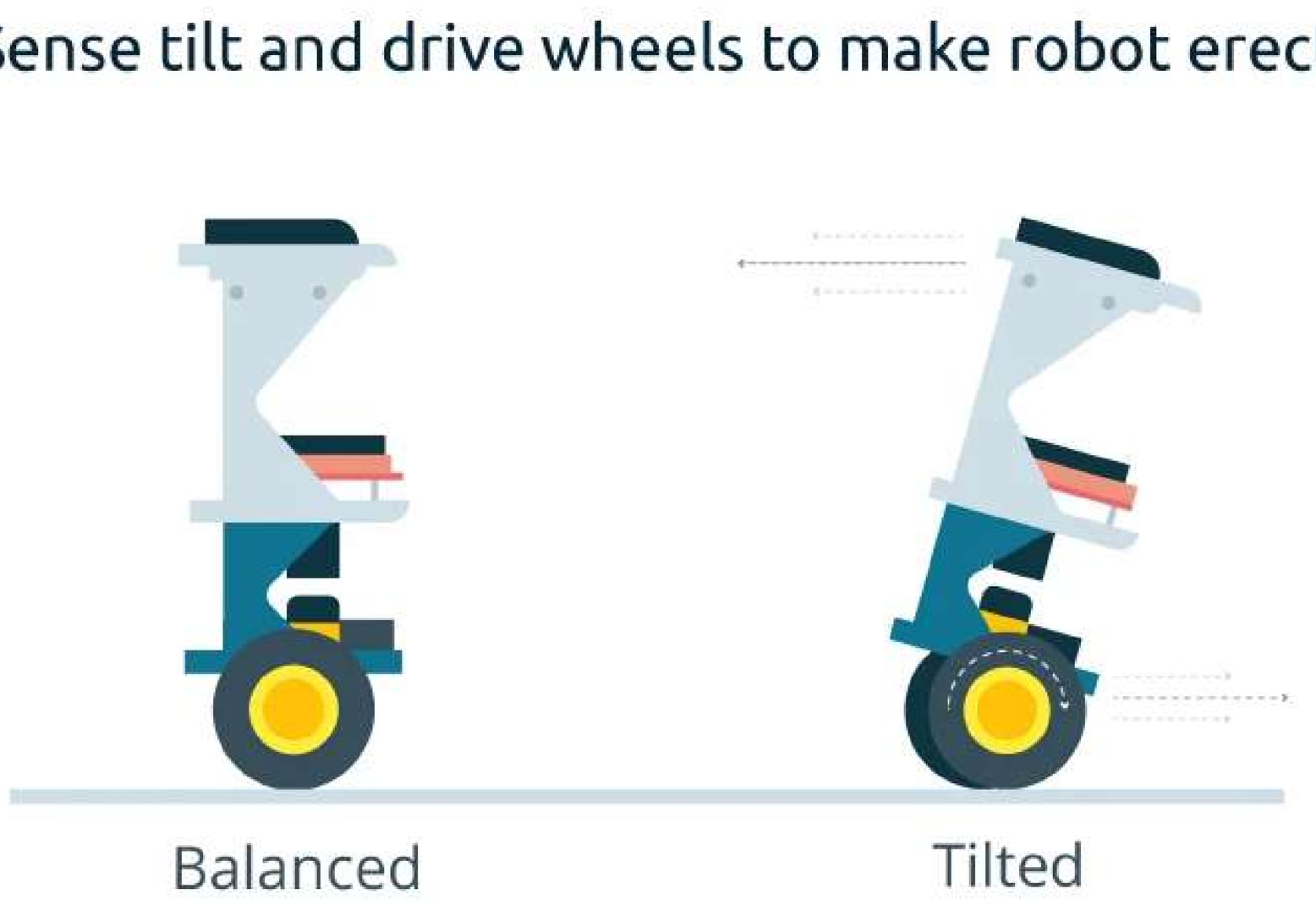
5V BATTARY

ARDUINO UNO (ATMEGA 328P)

ACCELEROMETER + GYROSCOPE MPU6050

• WORKING

To keep the robot balanced, the motors must counteract the robot falling. This action requires feedback and correcting elements. The feedback element is the MPU6050 gyroscope + accelerometer, which gives both acceleration and rotation in all three axes. The Arduino uses this to know the current orientation of the robot. The correcting element is the motor and wheel combination.



we only need angular position or gyro of the wheel's axis. Gyro readings drift over time, requiring recalibration quite often. Therefore, to get correct angular position, gyro readings are corrected with the help of a neighbouring accelerometer. Once the angular position is achieved, traction motors push the cart towards the direction of falling. The greater the angle of shift, the greater the speed with which the traction motor pushes the cart. As the angle of shift wrt vertical position reduces to zero, the speed reduces. Thus, the top of the cart moves like a pendulum, maintaining the balance. The author's prototyp

APPLICATION

1)Self-balancing unicycle, a self-powered unicycle that balances itself in three dimensions

2) Monocycle Is The World's First Self Balancing Bike: The Monocycle is a perfect city bike that can be used for short distance travels and can be parked in the least available of spaces. Moreover, the bicycle comes with a strong electric motor to support the riders, so they need not push too hard to speed up the bike.

3)Electric self-balance bike scooter/with Remote switch/balance bike: it's a self balancing electric bike which balance itself on its two wheels.

MERITS

ConseptOf self Balancing robot Using In Segway , THE two-wheeled design of the self-balancing Segway Personal Transporter significantly increases its maneuverability, because it reduces the turn radius to zero. The vehicle can rotate in place to instantly change its direction of motion and precisely navigate tight spaces that a three or four-wheeled robot cannot.

DEMERITS

Two Wheeled robots can not navigate well over obstacles, and this is the main drawback of this type, depending on the terrain, such as rocky terrain, sharp declines or areas with low friction, There are some situations where the wheels are not the best choice, The robot needs to pass small or large obstacles, Wheeled robots are useless in terrains that are not flat & have low friction coefficient.

CONCLUSION

After dooing this project we know that use of MPU6050 sensor , we know about accelerometer and gyroscop , use of motor driver , in this project we are trying balance robot with 100% efficiency but we can't do this because of occilation of robot .this concepte widely use in self balanced bike or scoter as show in application section ,so this thing need more improvement in future.

BLBLIOGRAPHY

-- for code and some theory
https://create.arduino.cc/projecthub/
-- for drowing circuit diagram
fritzing software (self draw)
-- component details
https://www.theengineeringprojects.com

APPENDICES

APPENDIX 1 : CODE FOR ARDUINO

```
#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

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 = 175.8;
double setpoint = originalSetpoint;
double movingAngleOffset = 0.1;
double input, output;
int moveState=0; //0 = balance; 1 = back; 2 = forth
double Kp = 50;
double Kd = 1.4;
double Ki = 60;
PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);

double motorSpeedFactorLeft = 0.6;
double motorSpeedFactorRight = 0.5;
//MOTOR CONTROLLER
int ENA = 5;
int IN1 = 6;
int IN2 = 7;
int IN3 = 8;
int IN4 = 9;
int ENB = 10;
LMotorController motorController(ENA, IN1, IN2, ENB, IN3, IN4, motorSpeedFactorLeft,
motorSpeedFactorRight);

//timers
long time1Hz = 0;
long time5Hz = 0;

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

// initialize serial communication
// (115200 chosen because it is required for Teapot Demo output, but it's
// really up to you depending on your project)
Serial.begin(115200);
while (!Serial); // wait for Leonardo enumeration, others continue immediately

// 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
    Serial.println(F("Initializing DMP..."));
    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
        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(")"));
 }
}
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);
 }
 // 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);
 #if LOG_INPUT
```

```
Serial.print("ypr\t");
 Serial.print(ypr[0] * 180/M_PI);
 Serial.print("\t");
 Serial.print(ypr[1] * 180/M_PI);
 Serial.print("\t");
 Serial.println(ypr[2] * 180/M_PI);
 #endif
 input = ypr[1] * 180/M_PI + 180;
 }
```

ES102