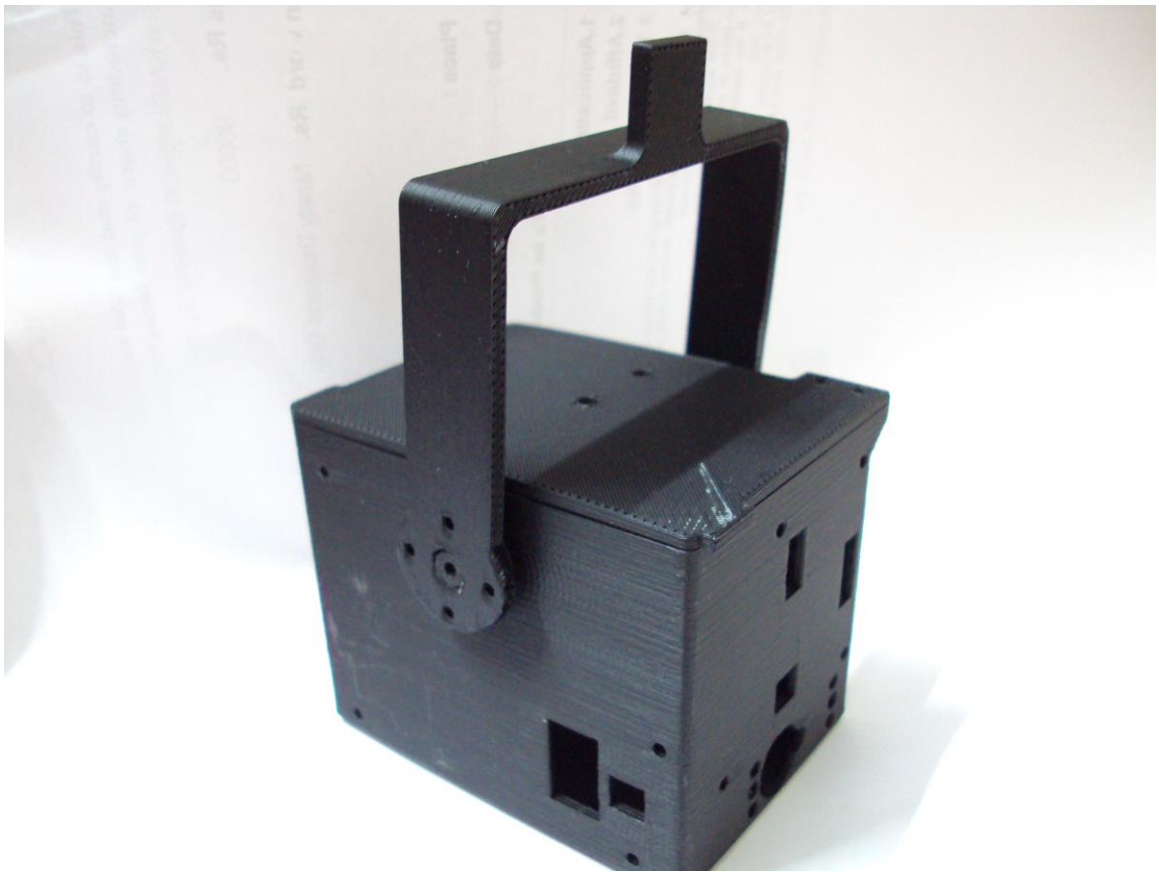


# LEECH ROBOT

A ROBOT CAPABLE OF REPLICATING LEECH MOTIONS

# Leech robot

## Submitted to

Prof. Prasanna Gandhi,

Suman mashruwala lab,

Mechanical department,

IIT-Bombay.

Summer internship 2014 by team of four members of B.Tech second year



| Sr.no | Name | Branch | College |
|---|---|---|---|
| 1. | Anshul .k. Paigwar | Mech | VNIT, Nagpur |
| 2. | Bhoumik Shah | Mech | IIT-Bombay |
| 3. | Deepak Miglani | Mech | IIT-Bombay |
| 4. | Gagan Makija | Mech | IIT-Bombay |

# Acknowledgment

*I hereby express my sincere thanks to respected Prof. Prasanna Gandhi, for giving me the chance to be the part of this project.*

*I also take an opportunity to express my deep sense gratitude to IIT Bombay for providing all the required facilities and inspiration in bringing out this project work.*

# INDEX

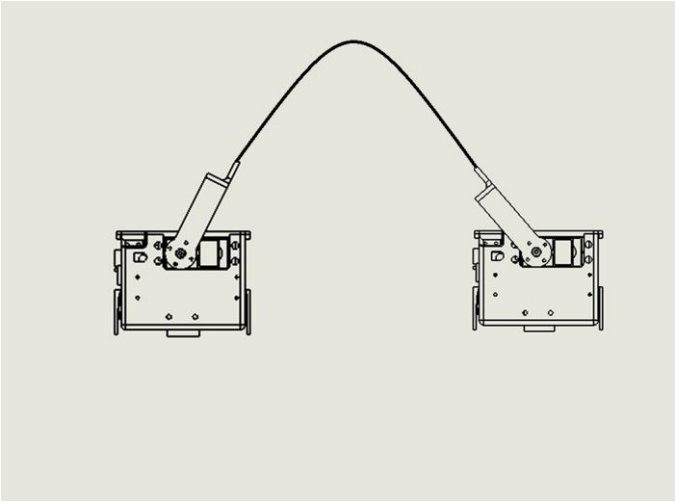
# Abstract

The special engineering project based around development of robotic platform for the use in analysis of flexible beams. This report deals with the development of the Sensing and Control system to allow the robot to move around its environment, using Inertial measurement unit to detect its orientation of its legs and performing the various motions .

# Introduction

This report outlines the initial work performed on design, fabrication and control system for the robot. Research into existing technologies is outlined, with appropriate references. The testing performed to validate the technologies used is described. The overall design, at the time of writing, is described, along with the functional description of each module.

The robot consist of a flexible beam and two modules attached each at the ends of the beam. These modules consist of servo motor, battery, circuit board, electromagnet, Inertial measurement unit, xbee module. The ends of beam are connected to the servo motor. The motor at the one end provide the necessary torque to lift the module at the other end. The following image show the arrangement of the robot:

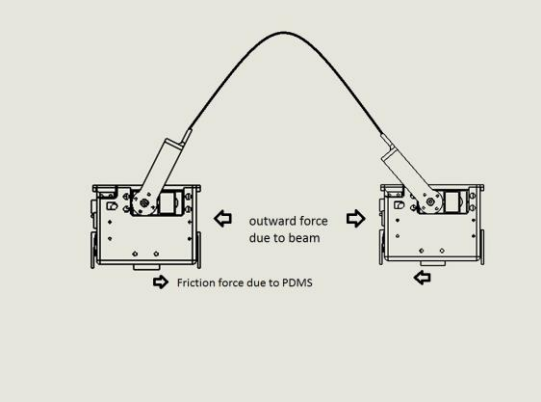


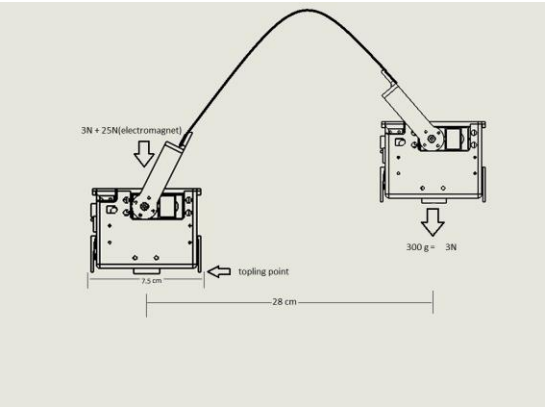
# Hardware and Design

The robot is to designed so as to move by lifting one leg while keeping the other one on the ground to achieve these following problems where faced:

## Problems faced

- While both the leg are on ground beam attached to them is in bent position, as the beam tries to push both the module in outward direction thus preventing modules to stand on their base.
- While lifting the other module the center of mass of robot lies somewhere between the two module thus creating a toppling force on the module on the ground and making it practically impossible to lift the other module.
- Another goal robot is to take turn, in order to achieve that robot module should be rotated about their axis but because of the use of PDMS on the base rotation is restricted due to high friction.




## Their proposed solutions

- The first problem was solved with the use of PDMS on the base of two module Polydimethylsiloxane (PDMS) belongs to a group of polymeric organ silicon compounds

that are commonly referred to as silicones. PDMS is the most widely used silicon-based organic polymer, and is particularly known for its unusual rheological (or flow) properties. PDMS provides the high friction and impact shock absorption thus prevent the modules to slide outwards due to reaction of beam.

- The toppling problem can only be solved if we could provide force in opposite direction. This force was generated using the electromagnet at the base. The electromagnet would stick to metal base and would prevent toppling. The electromagnet of lift capability of 25N is placed at the center of the base of both the modules. To prevent toppling the moment of the force generated by electromagnet should be greater than the moment due to toppling force about the toppling point.

  Moment due to toppling force : 3N x (28-3.75) = 72.75N-cm

  Moment due to force of electromagnet and weight of module: (25 N x 3.75 cm) +(3 N x 3.75) = 105N-cm

  Hence the toppling problem was solved with the use of electromagnets.

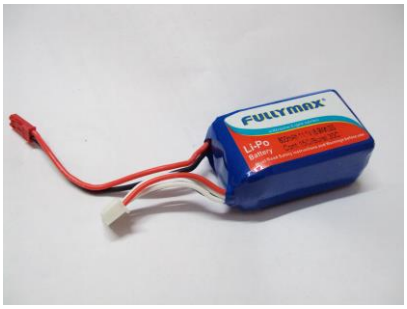
- To make the robot modules to be able to rotate there should not be any contact of PDMS with ground. To achieve this a cam design was proposed. Cams were used on both side of module as wheels working as differential drive mechanism. When the cams are at their lowest position contact is lost and module can be rotated. When the module is rotated cams again reaches high position and contact is established.

## Components

| Sr.no | Components | Qty. Per module | specification | weight |
|---|---|---|---|---|
| 1 | Servo motor | 1 | 15 kg-cm | 56g |
| 2 | Micro servo motor | 2 | 2 kg-cm | 12g |
| 3 | Electromagnet | 1 | 25N lift | 20g |
| 4 | Lipo battery | 1 | 11.1V ,800mAh | 70g |
| 5 | IMU MPU6050 | 1 | Accelero+gyro | 6g |
| 6 | Xbee series 2 | 1 | | 5g |
| 7 | Main circuit + Side circuit 1 + Side circuit 2 | 1 + 1 + 1 | | 10g +10g +5g |
| 8 | shell | 1 | | 30g |
| 9 | lid | 1 | | 20g |
| 10 | Battery bracket + battery bracket 2 | 1 + 2 | | 20g |
| 11 | cam | 2 | | 10g |
| 12 | Copper beryllium | 1 | | 10g |
| | Total weight | | | 284g |

1. Servo motor and micro servo motor


2. Lipo battery and electromagnet




3. Copper beryllium beam

Copper beryllium (CuBe), is a copper alloy with 0.5—3% beryllium. Beryllium copper combines high strength with non-magnetic and non-sparking qualities. It has excellent metalworking, forming and machining qualities. It has many specialized applications in tools for hazardous environments, musical instruments, precision measurement devices, bullets, and aerospace. Beryllium-containing alloys create an inhalation hazard during manufacturing due to their toxic properties. Beryllium copper is a non-ferrous alloy used in springs, spring wire, load cells and other parts that must retain their shapes during periods in which they are subjected to repeated stress and strain.

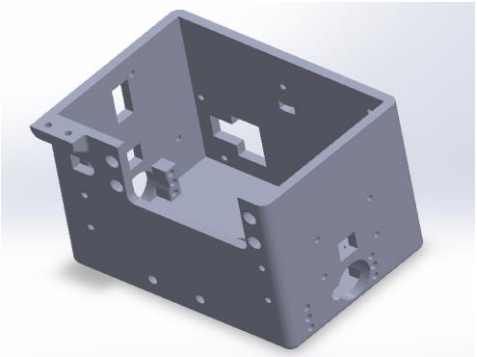
- A formula was derived to find the minimum length of the beam so that for given mass of modules to achieve the touch down condition of both the modules.

  L = 2.321*sqrt(EI/(mg)) Taking thickness = 1 mm, width= 30mm, E= 135 GPa we have, EI=0.3375 (in SI), mass =300g length of beam is calculated.

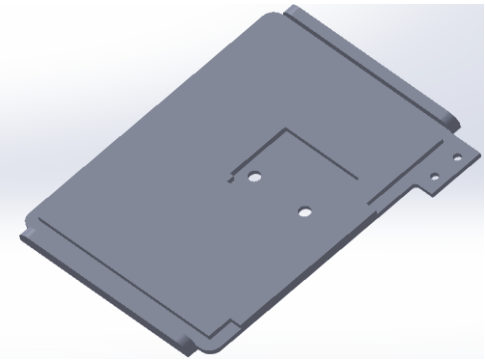
## Mechanical design

- The requirement of the leech robot was to be light and compact. The modules were designed to fit all the component described in most efficient and practical way.
- Various factors had to be taken in consideration while designing the enclosure for all of these components such as:
    1. Different types of gaits to be implemented.
    2. Impact on the enclosure while performing these gaits.
    3. Weight of robot.
    4. Easy fitting and removal of components
    5. Circuit board size and placement of components on it.
    6. Holes for things to be plugged into the bot eg. Programmer, lipo battery charger, reset button, nut bolt for fitting the components.
    7. Power supply
    8. Wireless transmission of data form IMU to computer for analysis.
- After considering all the above factors enclosure were designed in SOLIDWORKS and fabricated using 3D printing technology.
- Parts were so designed to avoid the overhang problem while 3d printing. Material used for manufacturing was ABS plastic.

1. Cad-model of shell



2. Cad-model of lid



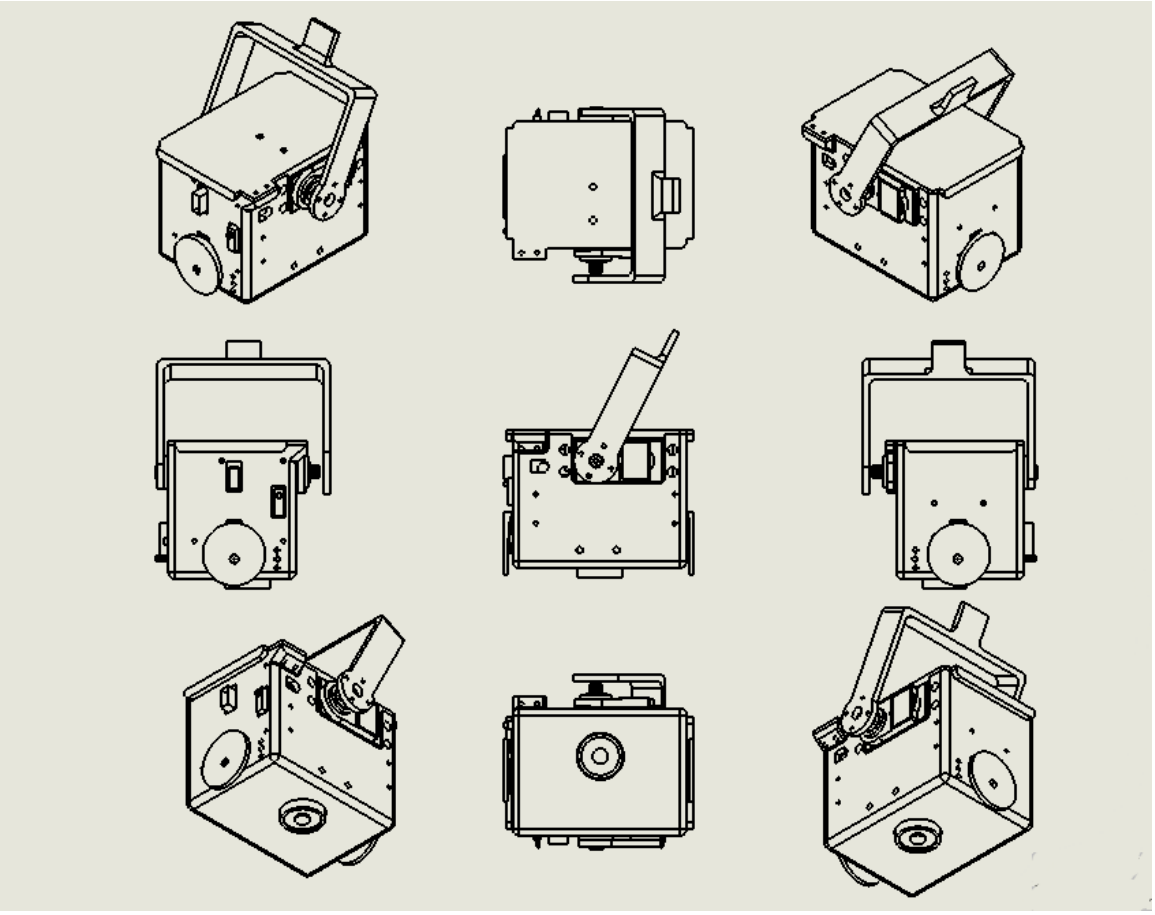
3. Actual shell

4. Actual lid

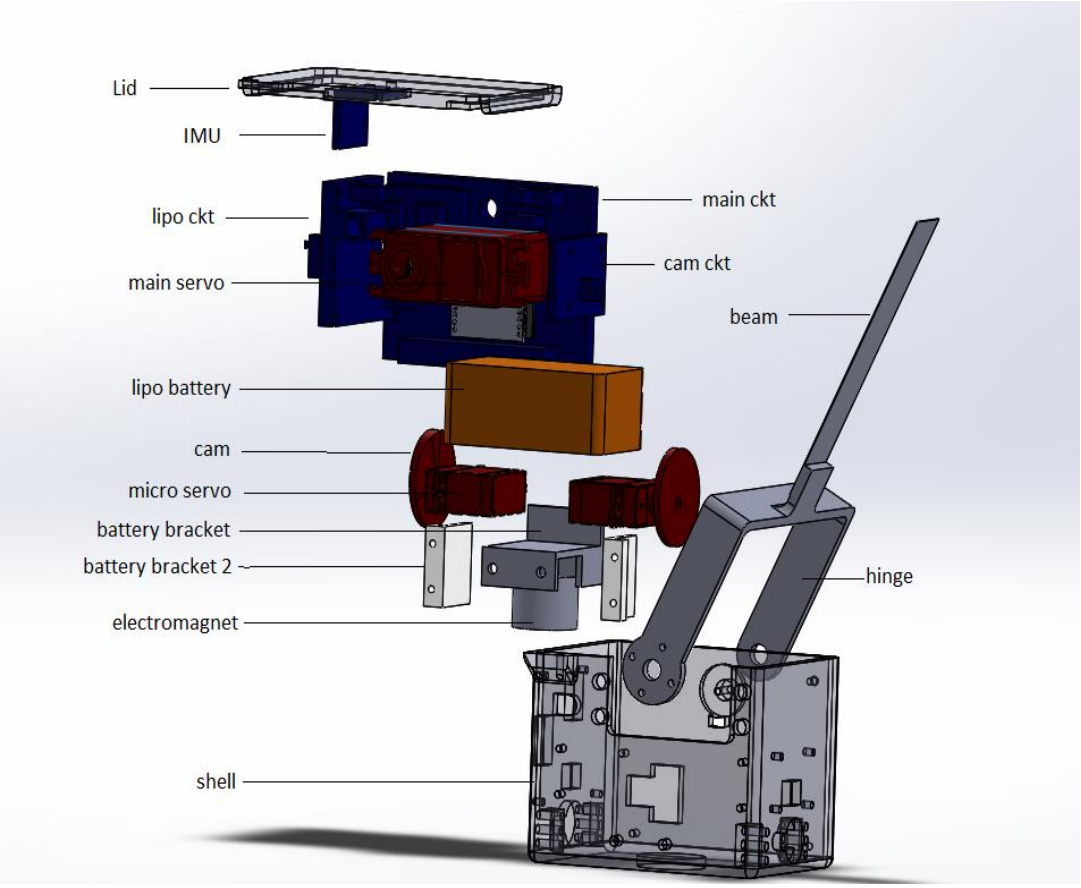
5. actual shell with battery brackets

6. Actual hinge

- Different views of the robot can be seen in the figure below



- Fitting arrangement of all the components is shown using the exploded view of Cad-model below:




## Electronics and circuit design

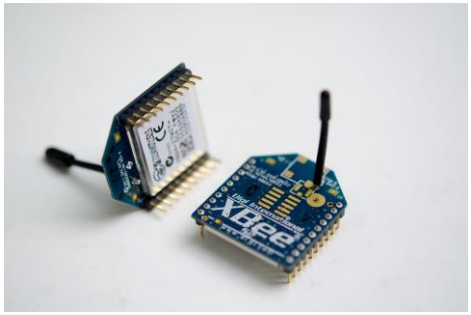
- Electronic circuit was designed in two parts. Main circuit and a side circuit.
- Main circuit is where the microcontroller is present and consist almost everything needed to control the robot.

1. AVR atmega-328p is used as main microcontroller. It the same microcontroller used in arduino uno boards. AVR microcontroller are widely used by the people in the field of robotics and have huge community.

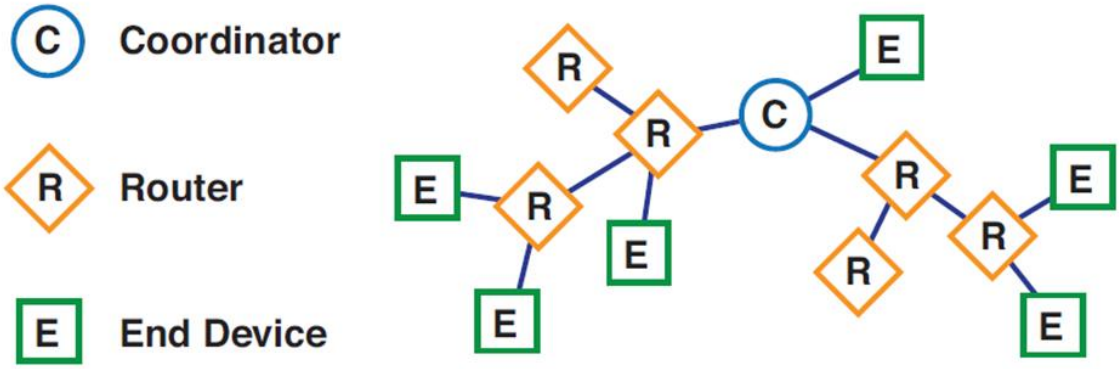
2. Atmega-328p is used since we can put arduino bootloader in it and can easily program IC using arduino IDE and all the arduino libraries can be used for easy and efficient programing. Tutorial on putting arduino bootloader can be found at the following links:
   http://arduino.cc/en/Tutorial/ArduinoToBreadboard
   http://arduino.cc/en/Main/Standalone

3. For programing Atmega-328 board have reset button and female header box to connect an ISP programmer.
4. Texas instrument DC-DC switch mode regulator PTN78060 V is used to convert 12V from lipo battery to regulated 5V power supply for microcontroller and the servo motors and other components. PTN78060 C can take up to 30 V as input and have adjustable output by adjusting resistor value across its pins. For 5 V value of resistor is 21 kilo- ohm. It has max output current value of 3 A.
5. JST connectors of 2mm and 2.5 mm pitch used to connect various components to circuit board. These connectors are small in size and have only one way to insert thus consumes lesser space than normal jumpers and eliminates risk of reverse connection and lose connections.
6. Board also consist of xbee module for wireless data transmissions. xbee is a short range low data rate wireless communication system. It works on mesh networking. Unlike other systems it transfers data from one node to other node to reach its destination node. Current manufacturers for XBEE's are DIGI. The speed is 250 Kbps but with noise it gives speed upto 25 Kbps. The IEEE standard for it is 802.15.4.
7. Xbee explorer is used to interface xbee with computer. We have used 3 xbee first xbee at the computer working as co-ordinator other xbee each in one module would act as router and the circuit board and IMU would act as end devices.

1. Xbee and xbee Explorer

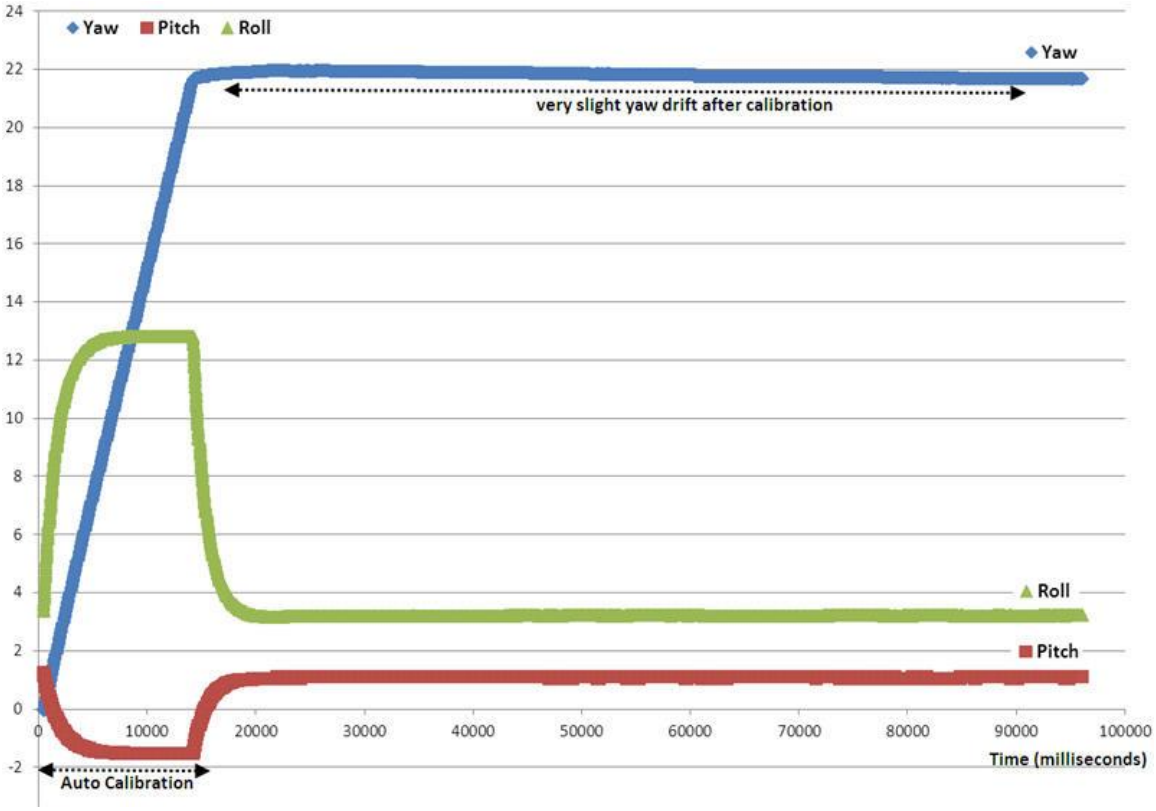


2. Xbee mesh networking




8. Since xbee works on 3.3 V Texas instrument linear voltage converter LP2950 is used convert 5 V to 3.3 V to power the xbee.
9. For communication through xbee RX and TX pin of Atmega has to be connected to TX and RX pin of xbee respectively. But since xbee works on 3.3 V and micro controller on 5 V TX pin of atmega328 cannot be directly connected to xbee RX pin so tri-state non inverting buffer from Texas instrument SN74LVC1G125 is used in between to reduce down the 5 V signals to 3.3 V to be feed to xbee.
10. For communicating between two modules two pins of Atmega-328 are reserved to be used as RX and TX pins using software serial library of arduino.

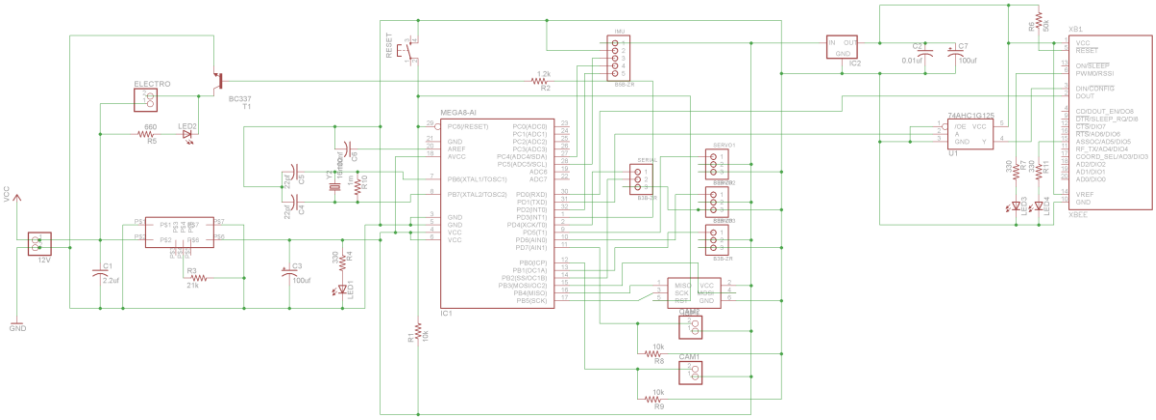
11. Electromagnet is controlled using BC337 NPN transistor as a switch.
12. IMU MPU6050 is used for knowing the orientation of the module. MPU6050 consist of accelerometer and gyroscope to detect its orientation. It also consist of Digital Motion Processing unit (DMP) which is an onboard chip that performs the calibration and calculation for filtering the raw data form accelerometer and gyroscope and gives the output in the form of yaw, pitch and roll or Euler angles. Arduino libraries for MPU6050 can be found I2Cdevil website and the sample codes are available on GIT-hub. MPU6050 uses I2C protocol to communicate with the microcontrollers. Auto calibration using DMP is shown in graph below:




Tutorials on interfacing MPU6050 with an arduino board can be found at following link:

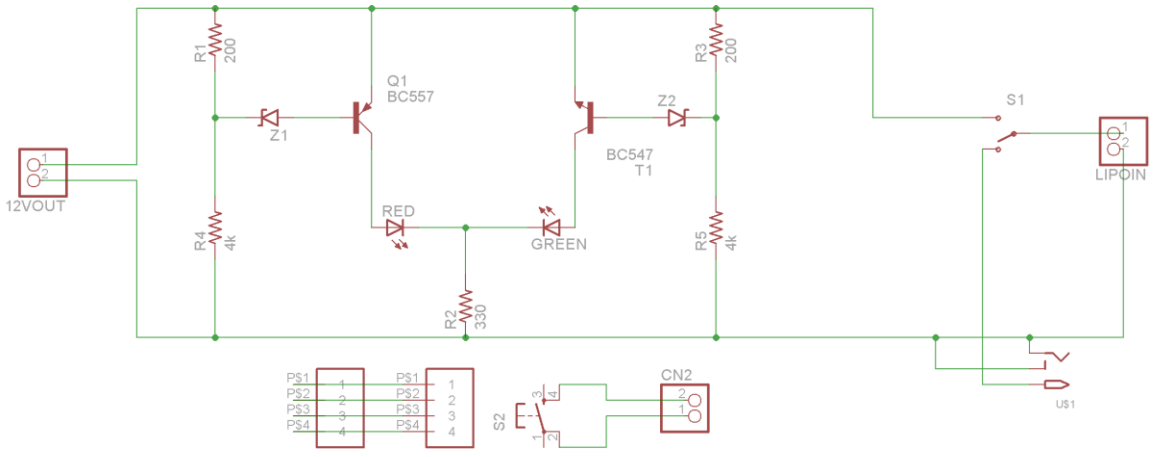
http://42bots.com/tutorials/arduino-uno-and-the-invensense-mpu-6050-6dof-imu/

- Schematic for the main circuit board is shown below:



- The side circuit consist of lipo protection circuit and have provision for charging lipo battery. Lipo battery is pluged into this circuit power is then supplied to main circuit using JST connectors. Indicator Leds are used to for indicating charge in battery. Led led glows if battery voltage goes below 10 V indicating low battery and green led glows if the battery voltage is greater than 12 v indicating full charge or over charge.

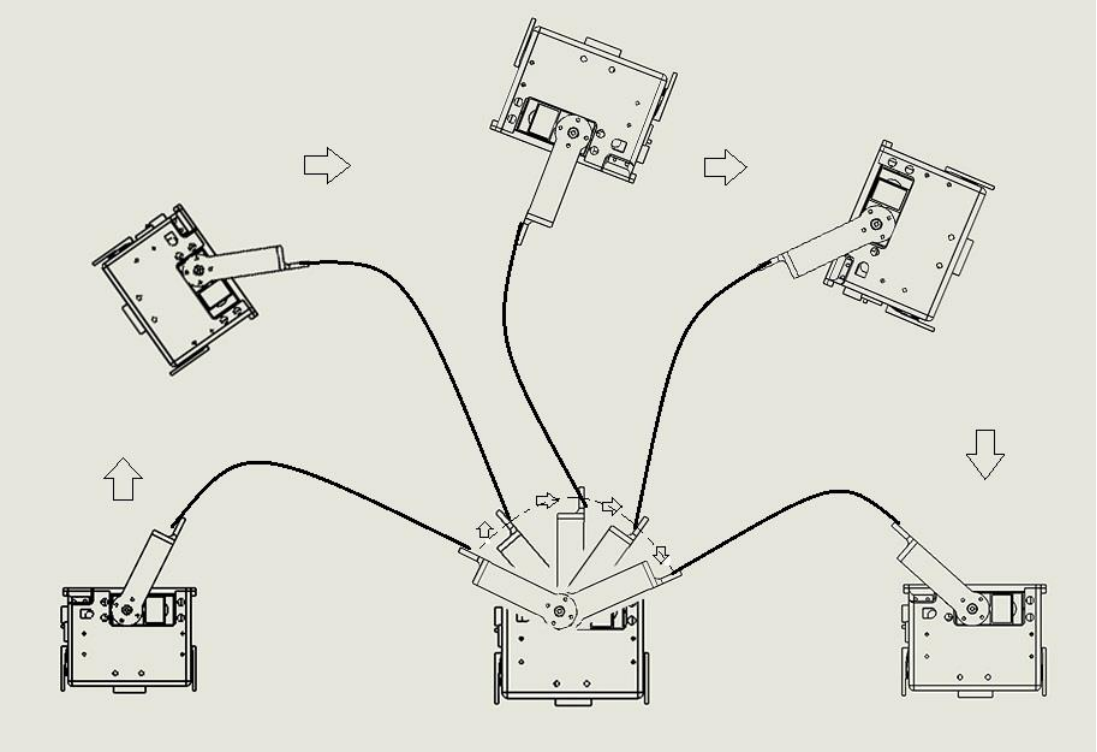
- The schematic for side circuit is shown below:



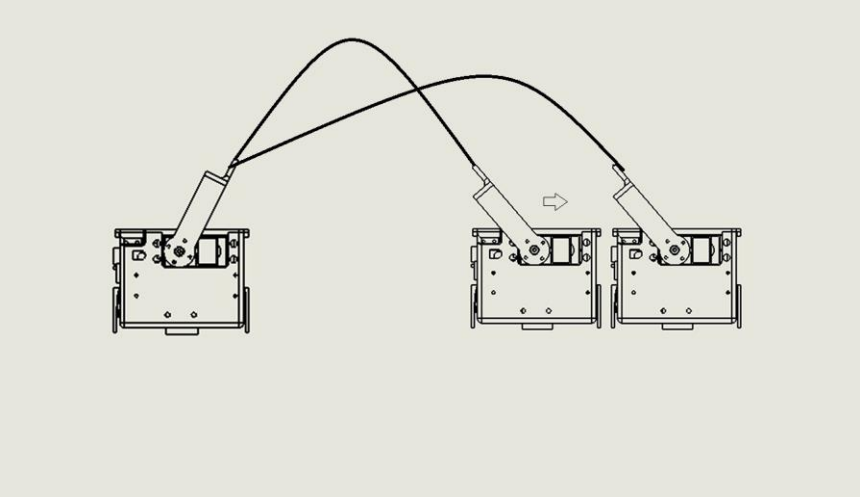
## Gaits and robot kinematics

Gaits is pattern of movements of limbs of animal during locomotion. The bot is programmed to perform following gaits:

- Forward motion:

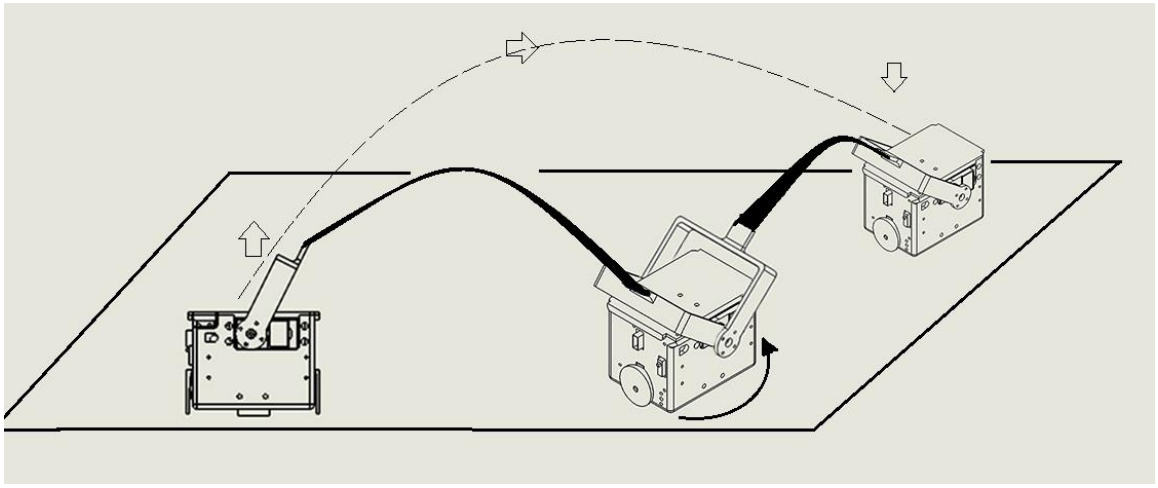


- Leech motion:

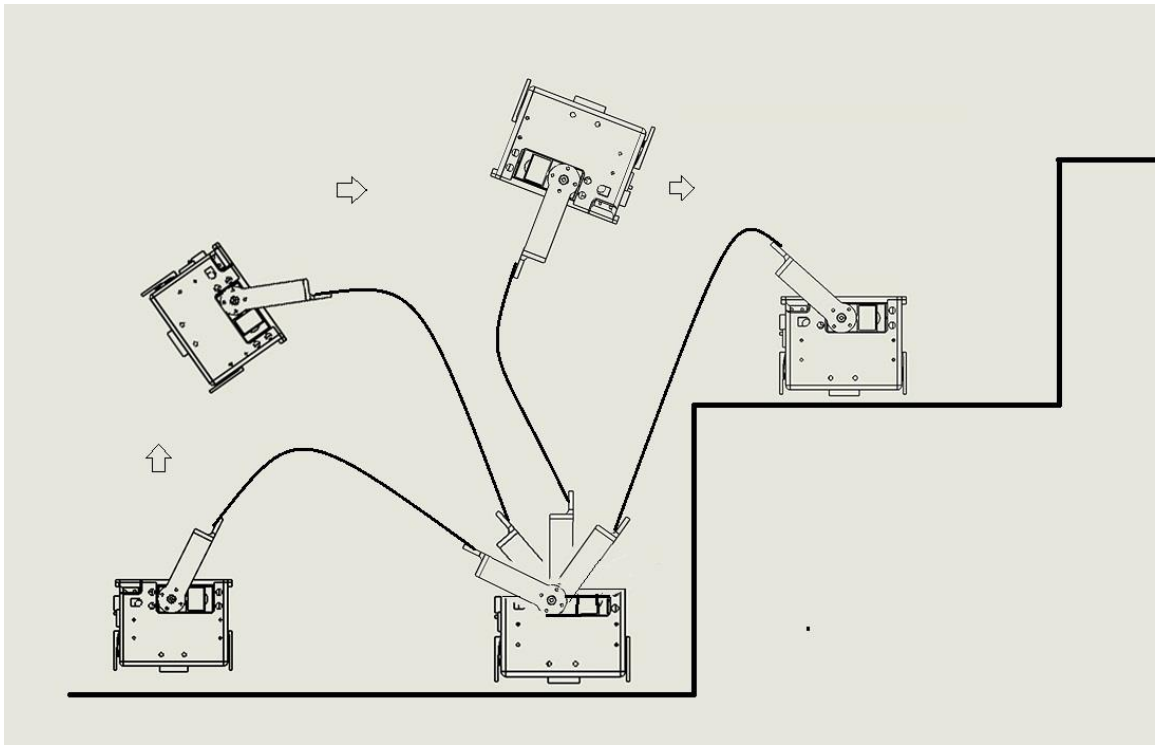


.

- Turning motion:



- Stair climbing:



## Conclusion

The robot System produced as a result of this project is capable of performing various gaits and can be used in various research projects. Robot body and chasis and Circuit Board was designed and manufactured. Atmega-328 was succesfully interfaced with IMU and with servo motor and with the use of xbee wireless communication with bot was established.

## Reference


- Paper On the Dynamics and Multiple Equilibria of an Inverted Flexible Pendulum With Tip Mass on a Cart by prof. Prasanna Gandhi and prof. Ojas Patil.
- M-Blocks: Momentum-driven, Magnetic Modular Robots by John W. Romanishin, Kyle Gilpin and Daniela Rus.


## Code

```
#include <Servo.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
  #include "Wire.h"
#endif
MPU6050 mpu;
#define OUTPUT_READABLE_YAWPITCHROLL
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

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




// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

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




// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================
```

```
void setup() {



    myservo.attach(9);

    // join I2C bus (I2Cdev library doesn't do this automatically)



Servo myservo;  // create servo object to control a servo
                // a maximum of eight servo objects can be created

int pos = 0;
    #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(9600);
    while (!Serial);

    // 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"));

    // wait for ready
    //Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    //while (Serial.available() && Serial.read()); // empty buffer
    //while (!Serial.available());                 // wait for data
    //while (Serial.available() && Serial.read()); // empty buffer again

    // 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();
    } 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(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}
```

```
float yawpitchroll(int j){
  // if programming failed, don't try to do anything
   if (!dmpReady) ;

   // wait for MPU interrupt or extra packet(s) available
   while (!mpuInterrupt && fifoCount < packetSize) {
      // other program behavior stuff here
      // .
      // .
      // .
      // if you are really paranoid you can frequently test in between other
      // stuff to see if mpuInterrupt is true, and if so, "break;" from the
      // while() loop to immediately process the MPU data
      // .
      // .
      // .
   }

   // 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;

    #ifdef OUTPUT_READABLE_QUATERNION
        // display quaternion values in easy matrix form: w x y z
        mpu.dmpGetQuaternion(&q, fifoBuffer);
```

```
    Serial.print("quat\t");
    Serial.print(q.w);
    Serial.print("\t");
    Serial.print(q.x);
    Serial.print("\t");
    Serial.print(q.y);
    Serial.print("\t");
    Serial.println(q.z);
#endif

#ifdef OUTPUT_READABLE_EULER
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q);
    Serial.print("euler\t");
    Serial.print(euler[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(euler[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(euler[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
```

```
    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

#ifdef OUTPUT_READABLE_REALACCEL
    // display real acceleration, adjusted to remove gravity
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    Serial.print("areal\t");
    Serial.print(aaReal.x);
    Serial.print("\t");
    Serial.print(aaReal.y);
    Serial.print("\t");
    Serial.println(aaReal.z);
#endif

#ifdef OUTPUT_READABLE_WORLDACCEL
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
```

```
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            Serial.print("aworld\t");
            Serial.print(aaWorld.x);
            Serial.print("\t");
            Serial.print(aaWorld.y);
            Serial.print("\t");
            Serial.println(aaWorld.z);
        #endif

        #ifdef OUTPUT_TEAPOT
            // display quaternion values in InvenSense Teapot demo format:
            teapotPacket[2] = fifoBuffer[0];
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

return ypr[j] * 180/M_PI;
}
```

```
}

// ================================================================
// ===                    MAIN PROGRAM LOOP                     ===
// ================================================================

void loop() {

   for(pos = 0; pos < 180; pos += 1)  // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
   myservo.write(pos);              // tell servo to go to position in variable 'pos'
   delay(15);                       // waits 15ms for the servo to reach the position
  }
  for(pos = 180; pos>=1; pos-=1)     // goes from 180 degrees to 0 degrees
  {
   myservo.write(pos);              // tell servo to go to position in variable 'pos'
   delay(15);                       // waits 15ms for the servo to reach the position
  }
  delay(100);
  float k;
  k = yawpitchroll(2);
  Serial.print(k);
  if ((k > 20)||(k < -20)){
   blinkState = HIGH;
       digitalWrite(LED_PIN, blinkState);
     }
  else
```

```
{
  blinkState = LOW;
        digitalWrite(LED_PIN, blinkState);
      }

}
```