Foldable Micro Aerial Vehicle
A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
In partial fulfillment of the requirements for the
Degree in Bachelor of Science
in
Robotics Engineering
By
Fuchen Chen
Weijia Tao
Dabai Zhang

Data: 2016/04/28
Project Advisor: Professor Cagdas Onal
Abstract

Folding is an efficient way of constructing 3D structures. It requires less energy to manufacture and less space for storage. This paper presents a robot fabrication method inspired by origami. Crease pattern is laser cut onto polyester sheets to become the basic parts that form the structure of robot. Such parts are then folded and assembled together. To demonstrate the functionality and reliability of this fabrication method, two kinds of micro aerial vehicles, a quadcopter and a vertical take-off and landing (VTOL) aircraft, are designed and tested. Test flights are performed on both vehicles.
# Table of Contents

Foldable Micro Aerial Vehicle........................................................................................................... i

Abstract ................................................................................................................................................ ii

Table of Contents ................................................................................................................................... iii

Table of Figures ....................................................................................................................................... iv

1 Introduction ......................................................................................................................................... 1

2 Background ......................................................................................................................................... 2

2.1 Quadcopter ..................................................................................................................................... 2

2.2 VTOL aerial vehicle ......................................................................................................................... 3

2.3 Foldable robot ................................................................................................................................. 5

3 Methodology ....................................................................................................................................... 7

3.1 Modular Concept ............................................................................................................................ 7

3.2 Design Flow ..................................................................................................................................... 8

3.3 Quadcopter Design ......................................................................................................................... 9

3.4 VTOL Aircraft Design ...................................................................................................................... 10

4 Results and Discussion ....................................................................................................................... 12

5 Conclusions ........................................................................................................................................ 15

6 Bibliography ....................................................................................................................................... 16

7 Appendix A – Connector Crease Patterns .......................................................................................... 17

8 Appendix B – VTOL Flight Controller Code ....................................................................................... 19
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crease Pattern and Folded Quadcopter Frame</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Foldable and Self-Deployable Pocket Sized Quadrotor</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>DJI Spreading Wings S1000</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Remote Control VFO</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>FireFly6</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>Foldable Hexapod robot</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>3-DOF Foldable Hexapod Robot</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>Basic Elements of Modular Concept</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>Design Flow of Foldable Robot Fabrication Method</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>Quad-copter Design and Finished Prototype</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>VTOL Aircraft Design and Finished Prototype</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>Quad-copter Test Flight GPS Position Data</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>Quad-copter Test Flight GPS Velocity Data</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>Quad-copter Test Flight GPS Altitude Data</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>Screenshots of VTOL Aircraft Hovering (Seconds)</td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>Screenshots of VTOL Aircraft Take-Off (Seconds)</td>
<td>14</td>
</tr>
</tbody>
</table>
1 Introduction

Folding is an efficient way of constructing three dimensional objects. This method manufactures the main parts in a planar fashion. Then, the parts are folded up and locked together to become structures and mechanism. This project utilized an origami inspired fabrication method to build robot body. This fabrication method requires less energy to manufacture, less space for storage, and less cost for raw materials.

However, the previous effort on this robot fabricating technique has some disadvantages. First of all, this method requires a new design for every new robot. There is a lack of design rules and basic components. Next, the finished robot body is not strong and reliable. Finally, it is also difficult to build complicated structures and mechanism because of the previous reasons.

In this paper, a systematic design and fabrication method of foldable robot is presented. This new system is based on a modular concept, which proposes that robot body can be built with just beams and connectors. Design flow of this system is demonstrated. Design of the basic parts are discussed. In order to show the feasibility and reliability of this unique method. Two micro aerial vehicles, a quad-copter and a vertical take-off and landing (VTOL) aircraft, are designed and built. Test flights are performed on both aircrafts.
2 Background

In this section, background information of quadcopter and VTOL aerial vehicle are discussed.

2.1 Quadcopter

Quadcopter, also known as quadrotor, is an aerial vehicle with four vertically oriented rotors placed to form a square. Its lift is generated by the propellers pushing the air downward. A quadcopter usually consists of a frame, four motors and propellers, a flight controller, and battery. Its frame can be made of different materials such as metal, plastic, carbon-fiber, or foam. The motors can either be brushed or brushless motors. The flight controller is in charge of the local motion control of the quadcopter. It usually has an inertial measurement unit as feedback. Electronic speed controllers are needed for flight controllers to control the speed of the motors. If the quadcopter is remotely controlled by users, a receiver and transmitter are also needed.

There are already some projects focusing on foldable quadcopter. Figure 1 shows a folded frames for quadcopter [1]. The fabrication method is similar to what this project uses. The crease pattern on the left can be automatically generated by the scripts written in Python given the desired dimension. Their quadcopter utilizing such frame can hover with a standard deviation around 1 cm. The disadvantages of their design is that it does not have any protection for the propellers which could be detrimental when quadcopter hits objects.

Another quadcopter that utilizes folding is shown in Figure 2 [2]. During transportation,
the arms of the quadcopter can be folded around the main frame. Before flight, the arm self-deploy in 0.3 seconds due to the torque generated by the propellers. Although it is fabricated differently from our design, the concept of reducing storage space by folding is the same. The same concept is also used in DJI Spreading Wings S1000, Figure 3, which is not self-deployable [3].

Figure 2 Foldable and Self-Deployable Pocket Sized Quadrotor

Figure 3 DJI Spreading Wings S1000

2.2 VTOL aerial vehicle

VTOL aerial vehicle are the ones that can take off and land from the same place without
using long run way. Up till now, there are three configurations, wing type, helicopter type, and ducted type configuration [4]. The flight control and stability of VTOL is very difficult and is the primary area of research in this field.

Figure 4 shows a remote control vertical flying object (VFO) [5]. It fits into the VTOL category. It is difficult to control manually but an experienced player could fly it. It is similar to what we are building. However, instead of manual control, local control is implemented to reduce the difficulty of controlling the VTOL.

![Figure 4 Remote Control VFO](image)

Besides the fixed wing remote control VTOL aircraft, hobbyist has also built a tiltrotor that is able to vertical take off and land as a rotary wing aircraft and fly as a fixed wing aircraft shown in Figure 5 [6]. During forward flight mode, the main lift is generated by the airstream flowing over the airfoil-shape wing. This design requires 6 motors in total which is not very efficient and increase the complexity of the system. Nevertheless, it is more efficient in forward flight mode.
2.3 Foldable robot

Foldable robot in this report refers to robots that are fabricated by folding plastic sheet. The plastic sheet is made of polyester. Laser cutter is used to cut out the crease pattern on the sheet [7] [8] [9]. Fasteners are included into the crease pattern so that no screws or glue is needed for building the frame. Figure 6 and Figure 7 are examples of such kind of foldable robots. Similar design patterns of these two robots are also implemented in the final design of FMAVs.
Figure 7 3-DOF Foldable Hexapod Robot
3 Methodology

In this part of the report, details of the fabrication method is discussed. The modular concept is explained and illustrated. General design flow of the foldable robot fabrication method is shown. To better demonstrate the capability of this design system, designs and finished prototypes of the quad-copter and VTOL aircraft are discussed.

3.1 Modular Concept

Previous method focuses on designing the entire structure of the robot together. The design tries to incorporate all functions at the same time, which could be very tricky and requires lots of iterations.

In the newly proposed method, basic elements that form the robot body are studied first. Folded triangular beam and connector are identified as the two basic elements. It is stated that that by using these two elements, lots of complicate and strong robot body can be made, satisfying all kinds of applications. Two types of cross sections for triangular beam are frequently used, equilateral triangle and isosceles right triangle. Connector is the part that can connect multiple beams together with each beams at different orientations. Figure 8 shows the most basic beam and connector. The topmost parts are the crease pattern. The middle ones are the sketch of the folded structure. The bottom one shows that these two are able to be connected together. Instead of the key lock design presented in [7] [8] [9], a new method is designed to connect beam and connector. The little rectangle located on the folding lines of the connector can be pressed down when the beam is folded, which will overlap with the same rectangles on the triangular beam. In this way, two parts are locked together since the side edge of it will prevent the beams from sliding out of the connector. Such design also allows easy disassembling. Crease patterns of all connectors are attached in Appendix A.
3.2 Design Flow

Given the modular concept, the design flow is shown in Figure 9. The requirements of the robot body are first identified. Then, robot body is designed in SolidWorks in terms of beams and connectors. Examples are shown in Figure 10 and 11. After that, dimensions of the predesigned beams and connectors are modified according to the actual size of the robot. To be noted, iterations happen between second and third steps since some desired structure is very difficult to build using triangular beams. Either an alternative structure is used or a new connector design is come up to realize the specific structure. New connector design may be useful for future need as well. Finally, all parts are printed out, folded up, and assembled.
3.3 Quadcopter Design

To design the quad-copter, requirements of the structure are listed.

1. Four motor holders.
2. Bumper.
3. Mounting place for electronics and battery

Iterations on the actual shape of the body and bumper are performed to achieve the best strength and optimum weight. The finished prototype is shown in Figure 10. The dimension is about 320*320*100mm. Electronic parts are listed below.

- RCX H1306 3100KV Micro Outrunner Brushless Motor
- 5x3inch Propeller
- Naze32 Flight Controller
- Emax SimonK 12A Brushless ESC
- Usmile NZ GPS
- 2 Cell 1200mAh LiPo Battery
3.4 VTOL Aircraft Design

For VTOL aircraft, the design requirements are also listed. The two control surfaces are used to balance and orient the aircraft during hovering and flying horizontally.

1. Flat plate wing.
2. Two control surface with sufficient size.
3. Center body to hold electronics and battery.
4. Two motor holders.

Figure 11 shows the design sketch and finished aircraft. The dimension is 500*250mm. Parts are listed.
- RCX 1804 2400KV Micro Outrunner Brushless Motor
- 5x3inch Propeller
- RCX 10A ESC Brushless Motor Speed Controller
- Atmega128RFA1 and MPU6050 Customized Flight Controller
- Cell 900mAh LiPo Battery

In this aircraft, a customized flight controller is built (see Appendix B). MPU6050 6DOF IMU is used to obtain the orientation of the aircraft. PID controllers are performed on each axis of the aircraft. The main controller is also capable of 2.4GHz RF wireless communication, which enables wireless control using phones and PCs.
4 Results and Discussion

Test flights are performed on both aerial vehicles. Quad-copter is fully functional and the test flight is very successful. It is remote controlled to fly around a regular football field outdoor. It has a battery life of around 10 minutes. The GPS tracking data is shown in Figure 12. The velocity and altitude data is shown in Figure 13 and 14. The top speed of the test flight is about 9 miles per hour. The maximum speed is not reached mainly due to inexperienced piloting.
Test flight performed on the VTOL aircraft includes taking off vertically and hovering. Screenshots of the hovering are shown in Figure 15. Take-off is shown in Figure 16. As shown, the aircraft is able to hover for about 2 seconds. The aircraft is able to take off. However, the controller is not capable of fully stabilize the hovering and take-off process. Still, the hardware of the robot is proved to be able to satisfy the application need.
Figure 16 Screenshots of VTOL Aircraft Take-Off (Seconds)
5 Conclusions

This paper demonstrates a new foldable robot design and fabrication system. Following the design procedures and using the predesigned parts, complicated robot body can be built such as quadcopter and VTOL aircraft presented. The strength of the finished structure can satisfy various application need.

Both aircrafts have a functional and reliable body. The test flight of the quad-copter is successful. VTOL aircraft can only hover for a short time and take off from ground. In the future, a more advanced flight controller should be developed to be able to fully control the VTOL aircraft.

Since the system proposed here is an evolving system, more and more connector design will be developed based on the need of future robot.
6 Bibliography


## Appendix A – Connector Crease Patterns

<table>
<thead>
<tr>
<th>Crease Pattern</th>
<th>3D Model</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td>Connect two equilateral triangular beams at a modifiable angle (Modifiable).</td>
</tr>
<tr>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td>Connect one equilateral triangular beam to an isosceles right triangular beam at a modifiable angle.</td>
</tr>
<tr>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td>Connect two equilateral triangular beams at a modifiable angle (Modifiable) with one pair of the faces aligned.</td>
</tr>
<tr>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
<td>Connect an isosceles right triangular beam to another one at a modifiable angle. One beam is allowed to go through the connector.</td>
</tr>
</tbody>
</table>
A different connection orientation of the previous type.

Connect an isosceles right triangular beam to an equilateral triangular beam at a modifiable angle. One beam is allowed to go through the connector.

Cross connection between two equilateral triangular beams.

Two equilateral triangular beams connect at 60 degrees. Another equilateral triangular beam is allowed to go through the connector.
8 Appendix B – VTOL Flight Controller Code

#include <Servo.h>
#include <MPU6050_6Axis_MotionApps20.h>
#include <I2Cdev.h>
#include "TimerThree.h"
#include "Wire.h"
#include "RadioFunctions.h"

// Objects
MPU6050 mpu;
Servo servoLeft; // pin 5, MID: 1650 UP: 2150, DOWN: 1150
Servo servoRight; // pin 4 MID: 1550 UP: 1050, DOWN: 2050
Servo motorLeft; // pin 3
Servo motorRight; // pin 2

#define PID_BUFFERSIZE 5
#define SERVOLEFT_MID 1650
#define SERVORIGHT_MID 1550
#define MOTOR_MIN 1050
#define MOTOR_MAX 2000

// Message Constants
#define START_MSG 0x11
#define END_MSG 0x99
#define DATA_LENGTH 4

#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

// orientation/motion vars
Quaternion q; // [w, x, y, z] quaternion container
VectorInt16 aa; // [x, y, z] accel sensor measurements
VectorInt16 aaReal; // [x, y, z] gravity-free accel sensor measurements
VectorInt16 aaWorld; // [x, y, z] world-frame accel sensor measurements
VectorFloat gravity; // [x, y, z] gravity vector
float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

// Controller variable
int desiredPitch = 30;
int desiredRoll = 0;
int desiredYaw = 0;
int desiredThrottle = MOTOR_MIN; // 1075-2000

// orientation stored in 1/1000 rad
// buffersize 5
int pitchRecord[PID_BUFFERSIZE];
uint8_t_t pitchPtr;
int rollRecord[PID_BUFFERSIZE];
uint8_t_t rollPtr;
int yawRecord[PID_BUFFERSIZE];
uint8_t_t yawPtr;

float pitchPGain = 0.7;
float pitchIGain = 0.02;
float pitchDGain = -0.15;

float yawPGain = 0.4;
float yawIGain = 0;
float yawDGain = -0.15;

float rollPGain = 0.09;
float rollIGain = 0.02;
float rollDGain = 0.1;

int pitchError[3];
int rollError[3];
int yawError[3];

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

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

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

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

20
volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

volatile bool controlLoopFlag = false;
volatile int counter;
void callback() {
    counter ++;
    controlLoopFlag = true;
}

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

void setup() {
    Serial.begin(38400);
    //----------------------------------------------MPU6050----------------------------------------------//
    MPU6050Setup();
    //-----------------------------------------------Timer-----------------------------------------------//
    Timer3.initialize(10000);  // 10ms or 100Hz
    Timer3.attachInterrupt(callback);
    //------------------------------------------------Controller----------------------------------------------//
    pitchPtr = 0;
    rollPtr = 0;
    yawPtr = 0;
    //-----------------------------------------------Servos----------------------------------------------//
    servoLeft.attach(5);
    servoRight.attach(4);
    motorLeft.attach(3);
    motorRight.attach(2);
    // Reset pos and speed
    servoLeft.writeMicroseconds(SERVOLEFT_MID);
    servoRight.writeMicroseconds(SERVORIGHT_MID);
    motorLeft.writeMicroseconds(MOTOR_MIN);
motorRight.writeMicroseconds(MOTOR_MIN);
delay(1000);
motorLeft.writeMicroseconds(MOTOR_MIN+50);
motorRight.writeMicroseconds(MOTOR_MIN+50);
delay(1000);
motorLeft.writeMicroseconds(MOTOR_MIN);
motorRight.writeMicroseconds(MOTOR_MIN);

rfBegin(11);  // Initialize ATmega128RFA1 radio on channel 11 (can be 11-26)

// Send a message to other RF boards on this channel
rfPrint("ATmega128RFA1 Dev Board Online!\n");

void loop() {
  // Input variable
  uint8_t message_buffer[DATA_LENGTH];
  uint8_t message_start;
  uint8_t message_end;

  // 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) {
    // Control loop cycle once
    if (controlLoopFlag){
      // Reset Flag
      controlLoopFlag = false;
      // Calculate error
      calculateError();
      //printError();
      calculateOutput();
    }
    // Check input
if (rfAvailable() >= 6) {
    Serial.print("S");
    message_start = rfRead();
    // Find the start of message
    if (message_start == START_MSG){
        for (int i = 0; i < DATA_LENGTH; i++){
            message_buffer[i] = rfRead();
        }
        message_end = rfRead();
        // If the data is aligned, do something
        if (message_end == END_MSG){
            Serial.print(message_buffer[0]);
            Serial.print(message_buffer[1]);
            Serial.print(message_buffer[2]);
            Serial.println("E");
            desiredThrottle = map(message_buffer[0],0,255,MOTOR_MIN,MOTOR_MAX);
            desiredRoll = message_buffer[1]/100.0;
            desiredYaw = message_buffer[2]/100.0;
            desiredPitch = -message_buffer[3]/100.0;
            Serial.println(desiredThrottle);
        }
    }
    MPU6050DAQ();
}

void calculateError(){
    // P error = mean every error
    int sumPTemp = 0;
    int sumRTemp = 0;
    int sumYTemp = 0;
    for (int i = 0; i<PID_BUFFERSIZE; i++){
        sumPTemp = sumPTemp + (desiredPitch-pitchRecord[i]);
        sumRTemp = sumRTemp + (desiredRoll-rollRecord[i]);
        sumYTemp = sumYTemp + (desiredYaw-yawRecord[i]);
    }
pitchError[0] = sumPTemp/5;
rollError[0] = sumRTemp/5;
yawError[0] = sumYTemp/5;

// I error = total every error
pitchError[1] = sumPTemp;
rollError[1] = sumRTemp;
yawError[1] = sumYTemp;

// D error = current error - earliest error
int earliestPitchPtr = pitchPtr + 1;
int earliestRollPtr = rollPtr + 1;
int earliestYawPtr = yawPtr + 1;
if (earliestPitchPtr > PID_BUFFERSIZE-1) earliestPitchPtr = 0;
if (earliestRollPtr > PID_BUFFERSIZE-1) earliestRollPtr = 0;
if (earliestYawPtr > PID_BUFFERSIZE-1) earliestYawPtr = 0;
pitchError[2] = (desiredPitch - pitchRecord[pitchPtr])-(desiredPitch - pitchRecord[earliestPitchPtr]);
rollError[2] = (desiredRoll - rollRecord[rollPtr])-(desiredRoll - rollRecord[earliestRollPtr]);
yawError[2] = (desiredYaw - yawRecord[yawPtr])-(desiredYaw - yawRecord[earliestYawPtr]);
}

void calculateOutput(){
  int pitchOut = pitchPGain*pitchError[0]+pitchIGain*pitchError[1]+pitchDGain*pitchError[2];
  int rollOut = rollPGain*rollError[0]+rollIGain*rollError[1]+rollDGain*rollError[2];
  int yawOut = yawPGain*yawError[0]+yawIGain*yawError[1]+yawDGain*yawError[2];
  // Serial.print("pitchOut: "); Serial.print(pitchOut);
  // Serial.print(" rollOut: "); Serial.print(rollOut);
  // Serial.print(" yawOut: "); Serial.println(yawOut);

  // Servo output values
  int servoLeftOut = SERVOLEFT_MID - pitchOut;
  servoLeftOut = servoLeftOut + yawOut;
  if (servoLeftOut > SERVOLEFT_MID+500) servoLeftOut = SERVOLEFT_MID+500;
  if (servoLeftOut < SERVOLEFT_MID-500) servoLeftOut = SERVOLEFT_MID-500;

  int servoRightOut = SERVORIGHT_MID + pitchOut;
  servoRightOut = servoRightOut + yawOut;
  if (servoRightOut > SERVORIGHT_MID+500) servoRightOut = SERVORIGHT_MID+500;
  if (servoRightOut < SERVORIGHT_MID-500) servoRightOut = SERVORIGHT_MID-500;

  int motorLeftOut = desiredThrottle - rollOut;
if (motorLeftOut > MOTOR_MAX) motorLeftOut = MOTOR_MAX;
if (motorLeftOut < MOTOR_MIN) motorLeftOut = MOTOR_MIN;

int motorRightOut = desiredThrottle + rollOut;
if (motorRightOut > MOTOR_MAX) motorRightOut = MOTOR_MAX;
if (motorRightOut < MOTOR_MIN) motorRightOut = MOTOR_MIN;

servoLeft.writeMicroseconds(servoLeftOut);
servoRight.writeMicroseconds(servoRightOut);
motorLeft.writeMicroseconds(motorLeftOut);
motorRight.writeMicroseconds(motorRightOut);

}  

void MPU6050Setup(){
    Wire.begin();
    TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz). Comment this line if having compilation difficulties with TWBR.
    
    // 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(38400);
    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"));
    
    // 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
    delay(5000);
// load and configure the DMP
Serial.println(F("Initializing DMP..."));
devStatus = mpu.dmpInitialize();

// supply your own gyro offsets here, scaled for min sensitivity
// -4952 -276 1613 62 -1 -3
// -4305 -1570 1693 49 8 36
// acelX acelY acelZ giroX giroY giroZ
mpu.setXAccelOffset(-4289);
mpu.setYAccelOffset(-1604);
mpu.setZAccelOffset(1684);
mpu.setXGyroOffset(46);
mpu.setYGyroOffset(6);
mpu.setZGyroOffset(37);

// 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(4, 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);

void MPU6050DAQ(){
    // 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 & 0x01) {
        // 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;

        // get orientation in yaw pitch roll
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

        // store orientations
        yawPtr ++;
        if (yawPtr>PID_BUFFERSIZE-1) yawPtr = 0;
        yawRecord[yawPtr] = (int) (ypr[0]*1000.0);
pitchPtr ++;
if (pitchPtr>PID_BUFFERSIZE-1) pitchPtr = 0;
pitchRecord[pitchPtr] = (int) (ypr[1] * 1000.0);

rollPtr ++;
if (rollPtr>PID_BUFFERSIZE-1) rollPtr = 0;
rollRecord[rollPtr] = (int) (ypr[2]*1000.0);

//        Serial.print(counter);
//        Serial.print(" ");
//        Serial.print(yawRecord[yawPtr]);
//        Serial.print(" ");
//        Serial.print(pitchRecord[pitchPtr]);
//        Serial.print(" ");
//        Serial.println(rollRecord[rollPtr]);
// blink LED to indicate activity
blinkState = !blinkState;
digitalWrite(LED_PIN, blinkState);
}

void printError(){
    Serial.print("P Error:");
    Serial.print(pitchError[0]);
    Serial.print(" ");
    Serial.print(pitchError[1]);
    Serial.print(" ");
    Serial.print(pitchError[2]);
    Serial.print(" ");
    Serial.print(pitchError[3]);

    Serial.print(" R Error:");
    Serial.print(rollError[0]);
    Serial.print(" ");
    Serial.print(rollError[1]);
    Serial.print(" ");
    Serial.print(rollError[2]);

    Serial.print(" Y Error:");
    Serial.print(yawError[0]);
    Serial.print(" ");
    Serial.print(yawError[1]);
    Serial.print(" ");
    Serial.print(yawError[2]);
}