Self-standing Monopod

Xin Chen
Diyu Yang
Jianan Gao

TA: Luke
ECE 445

February 2017

Abstraction

This report describe our journey of building a self-standing monopod over the whole semester. To accomplish this goal, we explored the property of control system by programming in microcontroller to work together with our motor-propeller system. Although this project is not working as we expected, we could still see how much efforts we devoted into this project.

Contents

Chapter 1

Introduction

1.1 Problem Statement

When doing photography we usually needs tripods to stabilize our camera. However, the problem with a tripod is that it's heavy to carry, difficult to setup, and it does not have too many shooting angles for the camera. Plus it requires a perfectly flat ground to use tripod, which is somewhat inconvenient for photographers. Therefore, we want to design a self-standing mono-pod which is robust and stable for photographers with the technique of PD control. We also get inspiration from one youtube video to better design our project.[1]

1.2 Objectives

1.2.1 Goals

- The monopod is able to stand by itself when the control system is working.
- Weight is lighter than a tripod with same length.
- The monopod can be placed in most of the terrains.
- The monopod is able to withstand at least a weight of a camera on the top of the monopod.

1.2.2 Functions

- Balance at the vertical position when no external forces applied.
- Be able to go back to equilibrium position when constant external forces is less than 5N.
- Get back to equilibrium position within 5s when tapping force less than 7N is applied on the monopod.

1.2.3 Benefits

- It has the ability to hold heavy weight camera.
- Users do not need to manually set the monopod to center point.
- Users do not need to worry about the monopod with a camera falling down in most of the cases.

1.2.4 Features

- It will balance itself due to the technique of PD control.
- It can recover to original state when it senses any impulse from outside.
- The monopod is robust enough to withstand wind with speed less than 20 km/h.

Chapter 2

Design

2.1 Block diagram

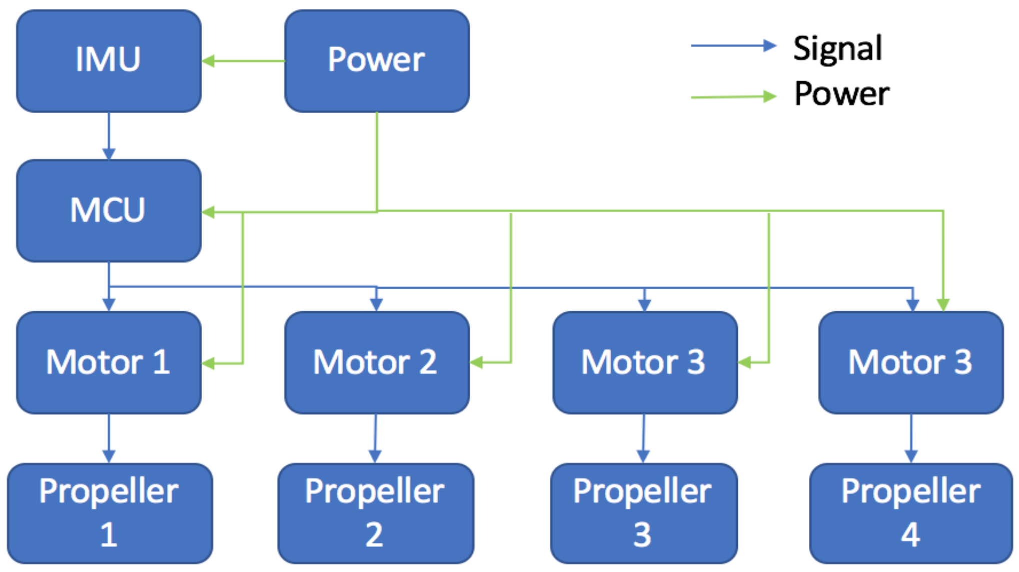


Figure 2.1: Block diagram

2.2 Block description

2.2.1 Power supply

Provide 8V voltage and up to 3A current for our whole system.

2.2.2 Brushless motor

These motors function as actuators of our system. They take PWM input from the microcontroller and spin the propeller at desired speed.

2.2.3 Propeller

These two wheels are mechanical device to generate torque to pull the monopod to its equilibirum point.

2.2.4 Inertial measurement unit(IMU)

We use IMU's accelerometer as input to our control system. We use this data to calculate the angle how much the monopod is away from vertical position.

2.2.5 Microcontroller

This works as a control unit that will take input from IMU and process the data, and output PWM signal to our actuators.

2.3 Design

2.3.1 Math model

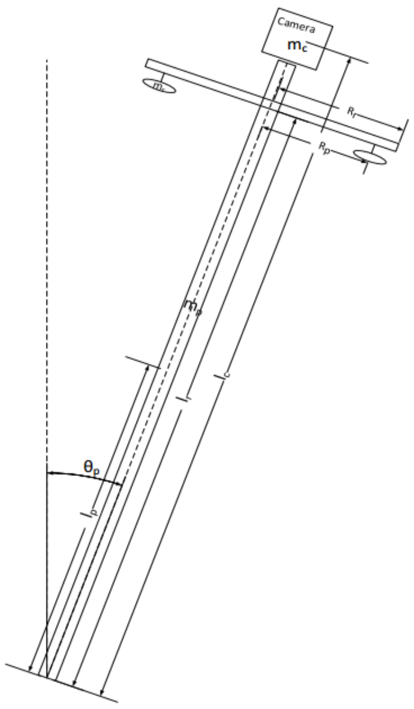


Figure 2.2: Math model

$m_p = 0.363$kg Mass of monopod

$m_r = 0.02$kg Mass of each propeller motor package

$m_c = 0.5$kg Mass of the camera

$l_m ax = 2$m Maximum length of monopod

$l_m in = 2$m Minimum length of monopod

F_r Force produced by one propeller

θ_p Distance from bottom to the center of mass of pendulum

l_r Distance from bottom to the propeller

$l_c = l + 0.1$m Distance from bottom to center of mass of camera

$R_p = 0.15$m Distance from propeller to the pod

Our PID control system takes pendulum angle θ_p as input, and output the force needed to produce for each propeller in each direction.
Net Torque added onto the system:
$\tau = (m_p l_p + 4 m_r l_r + m_c l_c) g \sin(\theta_p)$ - $F_r\ R_p$
$\tau = -\dot{L}$

Where L is the angular momentum of the pendulum-propeller system, and can be expressed as follows:

$L = J\dot{\theta}_p$

2.3.2 Microcontroller

We use a ATmega328p as our MCU. The advantage of this chip is that it allows for in-system programming (ISP) and offers support for all of the digital pulse width modulation (PWM) functionality that we required. This works as a control unit that will take input as output from IMU and process to get the duty cycles the PWM signal need for motor to balance the monopod. The schematics circuit design for our microcontroller is the following:

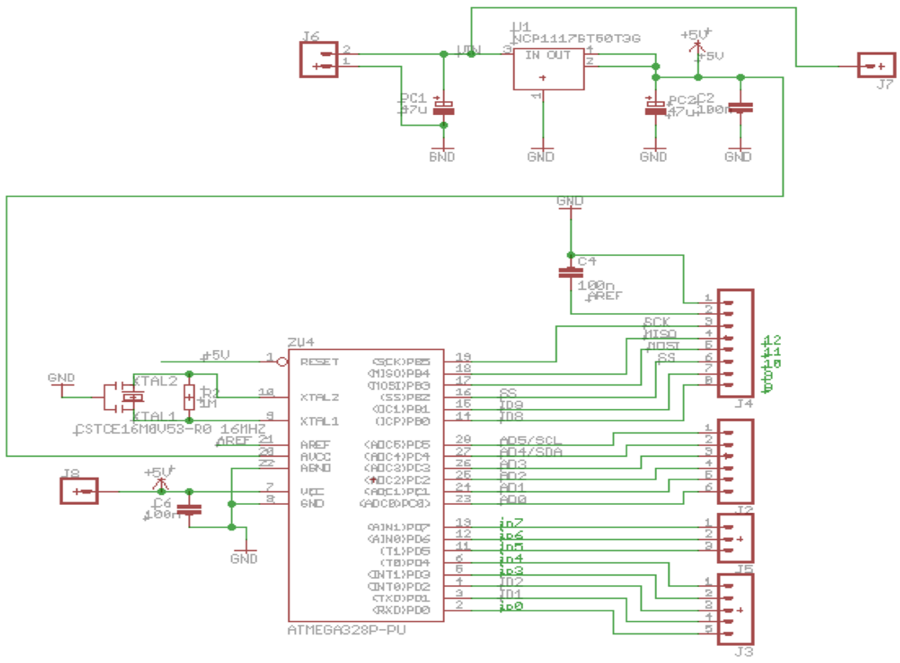


Figure 2.3: MCU schematics

In this schematic circuit, we designed one 5V output pin with the help of voltage regulator which we use as power supply for our motor drivers. In addition to 5V pin, we have 4 PWM outputs to manipulate spinning speed of motors.

The picture below is our PCB for microcontroller circuit. We have one 5V voltage regulator that provide power of the microcontroller. We also have the atmega328p sitting in the middle of the PCB on a socket(the piciture only shows the location of the chip.) The pins we use on this PCB are 2 analog input pins for communicating with IMU, and 4 digital output pins for sending PWM signal to motor drivers.

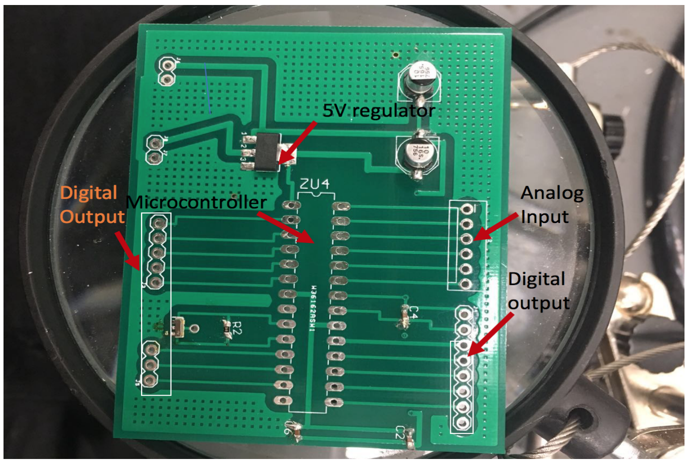


Figure 2.4: Pin assignment

2.3.3 IMU

The IMU we use is MMA8451 Triple-Axis Accelerometer 14-bit ADC. The reason we use this mode is because is has three axis sensing and offers 10-bit precision output and it can process data at a rate two times faster than nature frequency(from math model).This sensor communicates over I2C so we can share it with many other sensors on the same two I2C pins. There's an address selection pin that we can have accelerometers share an I2C bus. We have the schematic circuit for IMU is following:

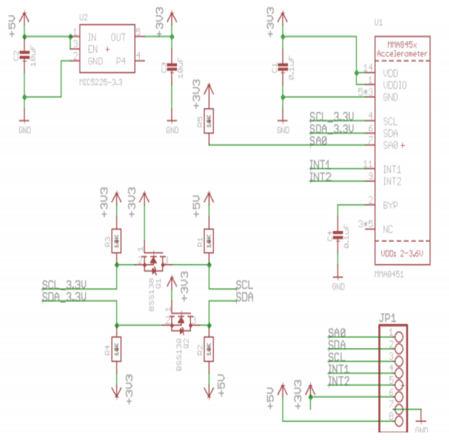


Figure 2.5: IMU Schematics

Here is how we connect pins to out MCU: Connect Vin to the power supply, we use 5V. Use the same voltage that the microcontroller logic is based off of. Connect GND to common power/data ground, connect the SCL pin to the I2C clock SCL pin on our Arduino. On an Atmega 328 based Arduino, this is A5. Connect the SDA pin to the I2C data SDA pin on your Arduino. On an UNO 328 based Arduino, this is also known as A4.

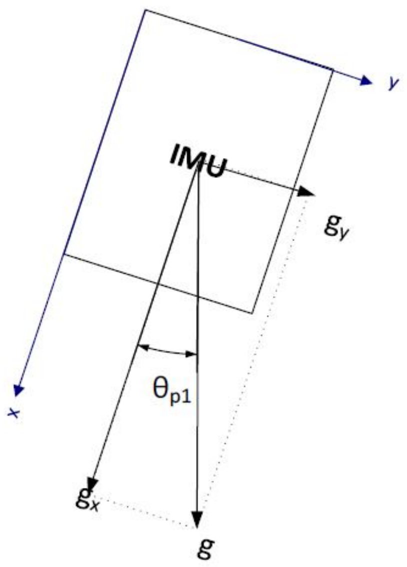


Figure 2.6: IMU angle

We calculated the angle from raw output of IMU by the equation:

$$\theta = sin^{-1}(\theta_p/G)$$

G is the gravity constant. This equation is used to calculate one direction angle, we need two angles which are x and y directions.

2.3.4 Median Filter

Although our calculated angles from IMU inputs are both accurate and stable at stationary positions, the motion of the monopod may still add noise into the system. Since the motion of the monopod generates accelerations along all axis of IMU chip, and our angles are calculated by the accelerations detected by IMU, our angles as a result, will be disturbed by the motion of the monopod. Such disturbance can generate significant amount of noise. As a result, a filter is needed to filter out the noise created by such motion. We chose median filter to do such a job because such filter is quite effective filtering out outlier data points. The plot below shows a simple experiment on the effect of median filter. During this experiment, we manually oscillated the monopod in a +/-5 degree motion range. The blue curve is the IMU angle before filtering, and the red curve is the IMU angle after filtering. We notice that the high frequency oscillating motion generates significant amount of noise to the IMU angle, for the angle goes from approximately -20 degrees to 20 degrees before filtering process. However, the range of the angle goes back to a reasonable +/-5 degree range after the filtering process, which means that our median filter does a pretty good job eliminating the outliers. From the appendix B arduino code, it shows the way how we implemented it.

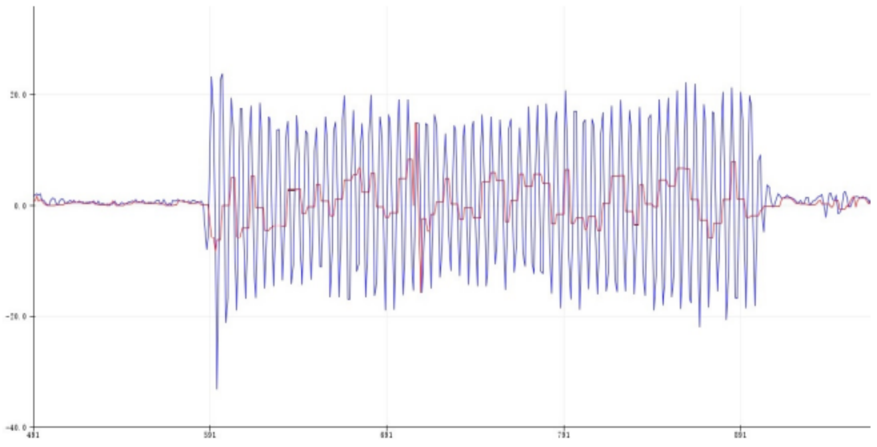


Figure 2.7: Median filter

In the following plot, we again compared the result of the data before and after the median filter for a smooth trajectory. This time we noticed that the filterd data have similar trajectory as the unfiltered data, which means that when the trajectory is smooth, that is to say there is little noise created by the motion, our filter actually let the data pass through.

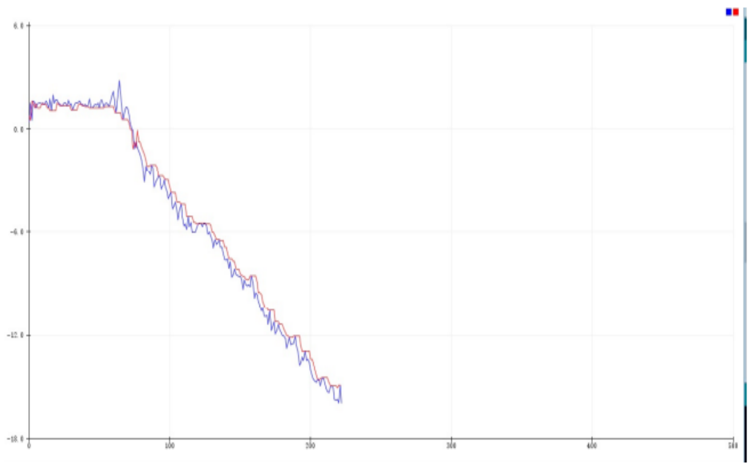


Figure 2.8: Median filter

2.3.5 Brushless motor & Propeller system

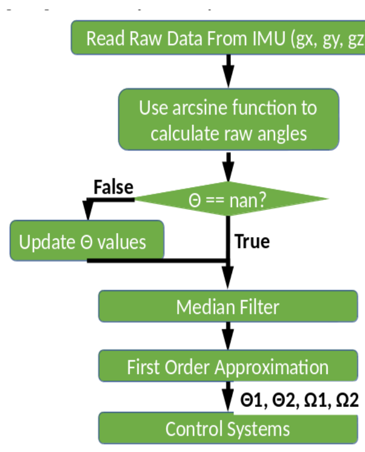
These motors function as actuators of our system. They are attached to propellers that generate thrust when spinning to stabilize our monopod. The maximum rpm is 11500 rpm. It takes inputs of three phase PWM signals. The reason we use brushless motors over brushed motors is that it can provide more torque under the same power supply. It operates under voltage range from 7V to 11V. The radius for our propeller is 5.5cm. The motor and propeller system is mechanical device that can generate torque to offset the acceleration caused by monopod gravity.

2.3.6 Control system

We programmed our control system in our microcontroller and implemented it with the help of PD control. The reason we used PD(Proportional-Derivative) control is because PD controller is enough for fast response controller since we don't need the steady state error in our system to be 0. The proportional part in our controller provides an instantaneous response to the control error. This is useful for improving the response of a stable system but cannot control an unstable system by itself. The derivative part provides a fast response, as opposed to the integral action, but cannot accommodate constant errors. Furthermore, we tuned our k_p, k_d by experiment. In this project, the ideal case we expect is that: rising time should less 20.8ms, steady state error should be less than 10% of input signal and overshoot should be less than 5% of input signal. The equation for PD control is following:

$$Output = k_p * (\Theta_d - \Theta(t)) + k_d * (\Omega_d - \Omega(t))$$

2.3.7 Flow chart



Read Raw Data From IMU (gx, gy, gz)
Use arcsine function to calculate raw angles
Θ == nan?
False
True
Update Θ values
Median Filter
First Order Approximation
Θ1, Θ2, Ω1, Ω2
Control Systems

Figure 2.9: IMU to control system

This is the bridge from IMU raw input to control system. As shown in the algorithm flow chart above, we first read raw acceleration input from IMU, then used arcsine function to calculate the two angles for each direction. We checked for Nan condition and then passed the two angles into a median filter to eliminate the noise produced by the accelerations of the monopod during the motion. For the derivatives terms, we took filtered angle values and did a first order approximation on each angle. We then pass the approximated Ω values into a mean filter. Finally we pass Theta and Ω values into the control system.

Chapter 3

Design verification

3.1 Microcontroller

In order to verify the functionality of microcontroller, we need to make sure that the following pins output expected signals: 5V, ground, 4 PWM signals, A4 and A5. We set up our circuit, connecting output pins to oscilloscope, we have the following outputs in figure 3.1.

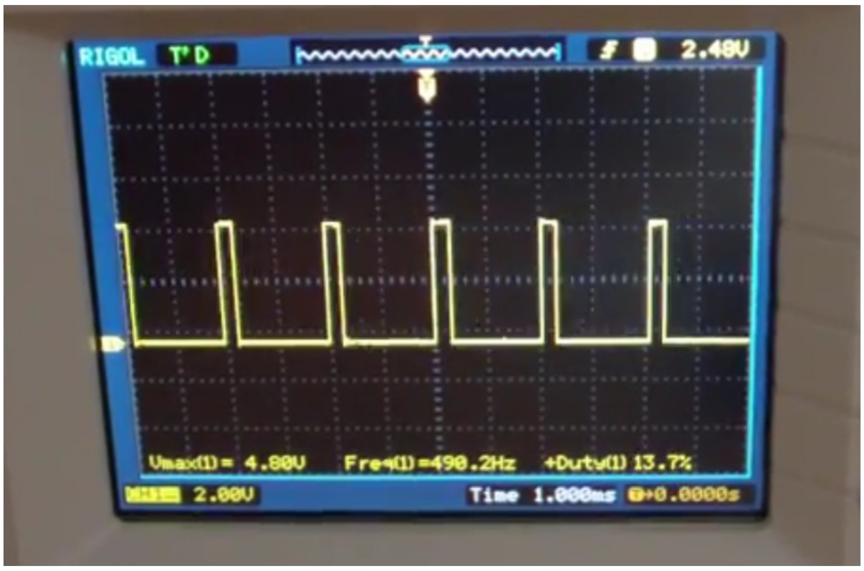


Figure 3.1: Data analysis

While testing, we set our PWM signals to be analogewrite(30) which is about quarter of duty cycle, the rising edge is the about quarter of the unit block, this means our four PWM ports are working properly. We use digital multimeter to test the output of the pin. The output is also 5V, this means the pin is working properly.

3.2 IMU

To test functionalty of IMU, we connected our IMU to our microcontroller and we printed the output from IMU in the series monitor in arduino software. We have our data analysis plot in figure 3.2.

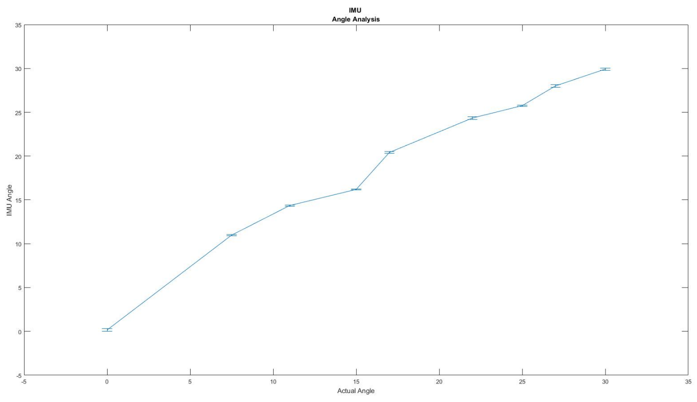


Figure 3.2: Data analysis

While testing, we obtained 10 samples for each angle, the bar on each angle is standard deviation. We can see on the graph that the standard deviation at each angle is very small, this means each sample is close to the average value and average value is close to real value. Therefore, this IMU can accurately detect angle.

3.3 Control system

Unfortunately, we were not able to verify the rise time and steady state error requirements for our control system. The main reason is that the rise time of the propeller-monopod system is mainly due to the hardware delay: it takes a long time for a motor to spin to the speed requested by PWM signal. So as a result, the rise time for the control system itself became less of an issue compared to the long rise time created by the hardware. Also for the steady state error, since our propeller-motor system failed to produce enough force to compensate the gravity of the monopod, so we were never able to actually reach the steady state of the system.

3.4 Brushless motor & Propeller system

Our brushless motor and propeller system should have provide enough thrust to support the falling gravity of monopod. Therefore, we tested the force generated by the system with the help of a scale and a bench as following figure.

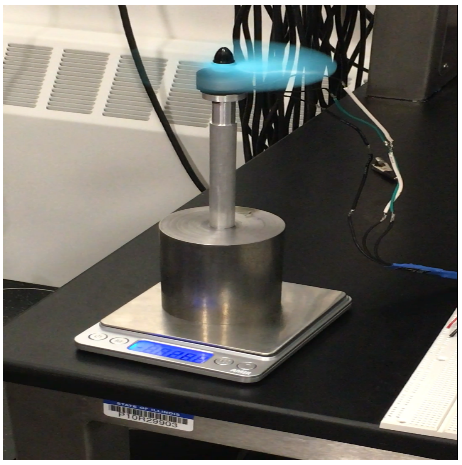


Figure 3.3: Testing motor

After testing the whole system, we had the data about PWM signal output and lifting force. After analyzing these data, we have the following diagram:

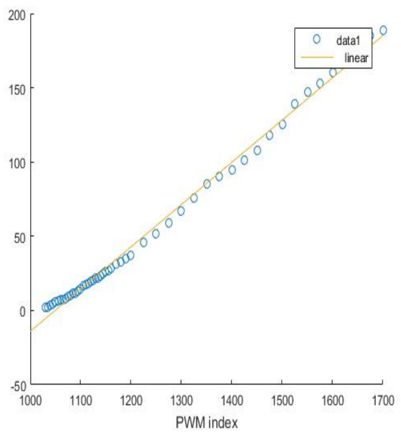


Figure 3.4: Testing motor

The lifting force and PWM signal has a linear relationship. This ensures that we can approximate lifting force by adjusting the PWM signal. This is important to our control system since we only need to modify PWM signal to change the spinning speed.

Chapter 4

Cost

4.1 Cost

4.1.1 Labor

Employee	Hourly rate	Total hours	Total cost
Xin Chen	\$50	150	7500
Diyu Yang	\$50	150	7500
Jianan Gao	\$50	150	7500

4.1.2 Equipments

Device	quantity	Model	Total cost
IMU	1	SparkFun 9 DoF Razor IMU M0	\$50
Microcontroller	1	Netmega	\$25
Motor	2	uxcell motor	\$2
Wheel	2	Carbon Fiber Propeller for Mini Electric Planes, 32mm	\$8
Wires	40	supply center	\$40

total: 22585

Chapter 5

Conculsion

5.1 Accomplishments

- Correct acceleration reading from IMU.
- Correct angle interpretation of IMU output in microcontroller.
- Median filter to filter out the noise of acceleration output.
- Developed working circuit for microcontroller unit.

5.2 Difficulties

- **Power supply**
  We did not manage to find any battery that is able to meet our requirement of having a peak current of 3A. We do not have an viable solution except that trying harder to find a lithium battery that meets our requirement.
- **Delay in motor**
  The delay of sending PWM signal to motor and motor spinning to the desired speed is the reason why we did not manage to get the monopod stand by itself. Potential solution is to design other control method such as feed forward.
- **Thrust of motor**
  The thrust of motor is not big enough to pull the monopod to its equilibrium point when the monopod is 30 degree from vertical line even though the motors are spinning at its maximum speed. One of the potential solution is to have more powerful motors.

5.3 Safety &Ethics

1.Our monopod use 4 propellers to generate torque. So one of the potential danger is that the sharp edges of propellers can damage objects or injure a human if they are close to the propellers. To be consistent with IEEE Code of Ethics #4[2]: "To accept responsibility in making decisions consistent with the safety, health, and welfare of the public." [1] We eliminate this danger by creating a cover for each of the propeller that protect users get in touch with these propellers.

2.We take great care when soldering components on PCB and soldering wires. Fume extractor is used all time to ensure clean air. We also wear safety googles every time we solder any components. We make sure nothing flammable is close to our soldering iron when we solder any components.

3.To be consistent with IEEE Code of Ethics #4: "IEEE Code of Ethics #4: "To accept responsibility in making decisions consistent with the safety, health, and welfare of the public." We are honest about our design specification throughout the design process.

4.To be consistent with IEEE Code of Ethics #9: "to avoid injuring others, their property, reputation, or employment by false or malicious action." We set the rule that in our development process; any personnel except the person holding the monopod should keep away from the monopod at least 5 feet. The person holding the monopod should take precaution.

5.4 Future work

- Add additional feature. Hold camera on top of the monopod.
- Make the height changeable so that there are more shooting angles.
- Solve the motor system delay problem by using more responsive motor or using reaction wheel instead of propellers. Make the monopod stand by itself.

5.5 Reference

[1] https://www.youtube.com/watch?v=woCdjbsjbPgt=13s

[2]"IEEE Code of Ethics." IEEE - IEEE Code of Ethics. N.p., n.d. Web. 08 Feb. 2017.

Appendix A

Requirement and verification

Requirements	Verifications	
Micro-controller • Input: signals from IMU outputs that tells MC the acceleration. • Output: signals to two motors that can control spin speed. • Requirements1: it must be able to communicate with IMU and motors simultaneously at speed greater than 4mHz. • Requirements2: it must be able to work under 5V voltage and calculate how fast the motor will spin to generate torque to offset the acceleration caused by monopod itself.	• Connect output to computer monitor to check if the micro-controller has correct output when monopod has some acceleration. • Provide input voltage for the microcontroller and test if it can run under the specific voltage. • Check code for calculating the torque for each motor to ensure the functionality for wheels.	30
IMU • Input: power supply with 5V voltage • Output: angular velocity, acceleration. • Requirement1: it should detect correct angular velocity and acceleration with the 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer inside the circuit. • Requirement2: it should report all the data collected from experiments.	• Check the circuit inside the manual and eyebow the correctness of the circuit. • Connect the output to computer monitor and check if the output of angular velocity, acceleration make sense to us. • Provide 5V voltage and test the functionality of the chip.	30

Motor 1 & 2 • Input: power supply and signal from microcontroller. • Output: motor spinning. • Requirement: it will spin in the direction we assign.	• Provide 5V power supply and test if the motor can spin in speed of 10400RPM • Reverse the power source and test if it can spin another direction.	10
Wheel 1 & 2 • Since they're mechanical devices, we need to make sure it can work fluently.	• Test the friction inside wheels and find the maximum torque it can generate.	10
Power supply • This power supply is from lab function generator.	• Test if the function generator can work.	10

Appendix B

Arduino code

```
    #include <Wire.h>
#include<math.h>
#include <Adafruit_MMA8451.h>
#include <Adafruit_Sensor.h>
#include <Servo.h>
#include <Filters.h>


Servo ESC10, ESC11, ESC9, ESC6; //Create as much as Servoobject you want. You can co
Adafruit_MMA8451 mma = Adafruit_MMA8451();

void setup() {
  // put your setup code here, to run once:
  ESC10.attach(10);     // attached to pin 10
  ESC11.attach(11);     // attached to pin 11
  ESC9.attach(9);
  ESC6.attach(6);
  Serial.begin(9600);
  if (! mma.begin()) {
    Serial.println("Couldnt start");
    while (1);
  }
  mma.setRange(MMA8451_RANGE_2_G);
  Serial.print("Range = "); Serial.print(2 << mma.getRange());
  Serial.println("G");
}
int initial = 1;

float theta1motor = 0; //Current angle, will be calculated via IMU measurement
float theta2motor = 0; //Current angle, will be calculated via IMU measurement

float Theta1_old = 0;
float Omega1_old1 = 0;
float Omega1_old2 = 0;
float Omega1 = 0; //Obtained by first order approximation

float Theta2_old = 0;
float Omega2_old1 = 0;
float Omega2_old2 = 0;
float Omega2 = 0;


float Ik1 = 0;  //integral part integral(e(t)dt) from 0 to current
float delt_IK1; //e(t)dt part



// Destination variables. These should always be 0 according to our purpose
float theta1d = 0;
float d_theta1d = 0;

float theta2d = 0;
float d_theta2d = 0;

float G = 9.81;
float delt_T = 0;       //time between two samples. How do we get this val??

int tau1 = 0;
int tau2 = 0;   //output torque
int tau3 = 0;
int tau4 = 0;
float tempval = 0;
int value1 = 0; //PWM signal
int value2 = 0;
int value3 = 0;
int value4 = 0;
unsigned long CurrentTime = 0;
unsigned long StartTime = 0;
float theta1rad = 0;
float theta2rad = 0;

//PID params
float Kpy = 20;
float Kdy = 3;

float Kpx = 20;
float Kdx = 3;

float Kp1 = Kpy;
float Kd1 = Kdy;
float Ki1 = 0;

float Kp2 = -Kpy;
float Kd2 = -Kdy;
float Ki2 = 0;

float Kp3 = Kpx;
float Kd3 = Kdx;
float Ki3 = 0;

float Kp4 = -Kpx;
float Kd4 = -Kdx;
float Ki4 = 0;

float theta1[4] = {0, 0, 0, 0};
float theta2[4] = {0, 0, 0, 0};
int compensation10 = 1160;
int compensation11 = 1030;
int compensation9 = 1516;
int compensation6 = 1595;
int PWMmax = 500;
int PWMmin = 0;

float freq = 15;
float gy, gx;
void loop() {
  //First send an initial PWM signal to motor
  if (initial)
  {
    delay (1500);
    ESC10.writeMicroseconds(1000);
    ESC11.writeMicroseconds(1000);
    ESC9.writeMicroseconds(1000);
    ESC6.writeMicroseconds(1000);
    initial = 0;
    delay(1000);
  }

    // First let's get output from IMU and calculate motor angles
    StartTime = CurrentTime;     //starttime is previous time
    CurrentTime = millis();
    delt_T = (CurrentTime - StartTime) * 0.001;
    //store the current time right before reading mma values
    mma.read();
    /* Get a new sensor event */
    sensors_event_t event;
    mma.getEvent(&event);
    //calculating angles
    // First let's LPF the acceleration
    gx = event.acceleration.x;
    gy = event.acceleration.y;
    tempval = asin(gy/G);
    if (!isnan(tempval))
    {
      theta1rad = tempval;
      theta1motor = theta1rad*180/M_PI;
    }
    tempval = asin(gx/G);
    if (!isnan(tempval))
    {
      theta2rad = tempval;
      theta2motor = theta2rad*180/M_PI;
    }
    Serial.print("Theta before LPF  = ");Serial.print("\t");Serial.print(theta1motor)
    //Apply a LPF filter directly to acceleration
    /*gy = (2*delt_T*freq+1)*gy + (2*delt_T*freq-1)*gy_old;
    gy_old = gy;
    tempval = asin(gy/G);
    if (!isnan(tempval))
    {
      theta1rad = tempval;
      theta1motor = theta1rad*180/M_PI;
    }
  */

    // First push thetamotor into the queue
    push(theta1, theta1motor);
    if (checkzero(theta1))
    // if no zeros in array, then apply median filter and update thetamotor value
    {
      theta1motor = sort(theta1);
    }
    push(theta2, theta2motor);
    if (checkzero(theta2))
    // if no zeros in array, then apply median filter and update thetamotor value
    {
      theta2motor = sort(theta2);
    }
    Serial.print("Theta after LPF = "); Serial.print(theta1motor); Serial.print("\t")
          //tau1 = 0;   //reset output torque to be 0

        // first order approximation for Omega
        Omega1 = (theta1motor - Theta1_old)/delt_T;
        Omega1 = (Omega1 + Omega1_old1 + Omega1_old2)/3.0;
        Theta1_old = theta1motor;
        Omega1_old2 = Omega1_old1;
        Omega1_old1 = Omega1;

    Omega2 = (theta2motor - Theta2_old)/delt_T;
    Omega2 = (Omega2 + Omega2_old1 + Omega2_old2)/3.0;
    Theta2_old = theta2motor;
```

```
    Omega2_old2 = Omega2_old1;
    Omega2_old1 = Omega2;


        //putting all pieces together
        tau1 = Kp1*(theta1d-theta1motor)+Kd1*(d_theta1d-Omega1);
    tau2 = Kp2*(theta1d-theta1motor)+Kd2*(d_theta1d-Omega1);
    tau3 = Kp3*(theta2d-theta2motor)+Kd3*(d_theta2d-Omega2);
    tau4 = Kp4*(theta2d-theta2motor)+Kd4*(d_theta2d-Omega2);
      //tau1 = tau1 + 5;
        // Boundary Check. if the torque is too big then saturate the integral
  /*
        if (fabs(tau1) < PWMmax)
                Ik1 = Ik1+delt_IK1;// normal calculation. No need to saturate
        else
                tau1 = tau1-Ki1*delt_IK1;
   */
  //Serial.print("Omega1: \t"); Serial.print(Omega1); Serial.println("\t");
  // raw PWM values without boundary check
  value1 = tau1 + compensation10;
  value2 = tau2 + compensation11;
  value3 = tau3 + compensation6;       //output to pin 6
  value4 = tau4 + compensation9;       // output to pin 9
  //Serial.print("Calculated PWM2 without saturation conditions: ");Serial.print(valu
  //run motor 1
  if (tau1 > PWMmax)
  {
    value1 = PWMmax+compensation10;
   //Serial.println("PWM value out of operatable range");
  }

  else if (tau1 < PWMmin)
  {
    value1 = 1000;
    //Serial.println("Motor 1 Pauses");
  }
  ESC10.writeMicroseconds(value1);

// run motor 2
  if (tau2 > PWMmax)
  {
    value2 = PWMmax+compensation11;
    //Serial.println("PWM value out of operatable range");
  }
  else if (tau2 < PWMmin)
  {
    //Serial.print("Motor 2 Pauses, calculated PWM: ");Serial.print(tau2+compensation
    value2 = 1000;
  }
```

```
    ESC11.writeMicroseconds(value2);

   // pin 6,value 3
   if (tau3 > PWMmax)
   {
     value3 = PWMmax+compensation6;
     //Serial.println("PWM value out of operatable range");
   }
   else if (tau3 < PWMmin)
   {
     //Serial.print("Motor 2 Pauses, calculated PWM: ");Serial.print(tau2+compensation
     value3 = 1000;
   }
    ESC6.writeMicroseconds(value3);

    //pin 9, value 4
       if (tau4 > PWMmax)
   {
     value4 = PWMmax+compensation9;
     //Serial.println("PWM value out of operatable range");
   }
   else if (tau4 < PWMmin)
   {
     //Serial.print("Motor 2 Pauses, calculated PWM: ");Serial.print(tau2+compensation
     value4 = 1000;
   }

    ESC9.writeMicroseconds(value4);
   Serial.print("PWM1: "); Serial.print(value1); Serial.print("\t"); Serial.print("PWM
   Serial.print("PWM3: ");Serial.print(value2); Serial.print("\t");Serial.print("PWM4:
Serial.println();
   Serial.println();

}
```