

**NEW YORK UNIVERSITY – Polytechnic School of Engineering**

**Department of Mechanical and Aerospace Engineering**

---



**ME 7863 – Advanced Mechatronics**

**Spring 2015**

**Integrated Project**

**AR Drone**

**Phase 3**

**Charging Platform Development**

**HENRY M CLEVER**

**FENG WU**

**ASHWIN R**

## CONTENTS:

SL.NO.	TOPIC	PAGE NO.
1.	Abstract	3
2.	Introduction	3
3.	Components	4
3.1.	Raspberry Pi	4
3.2.	IR Emitter	6
3.3.	Nintendo-Wii camera	6
3.4.	Arduino Micro	7
3.5.	XBee Explorer Regulated	7
3.6.	SparkFun XBee Shield	8
3.7.	Charging Platform	9
4.	Bill of Materials	10
5.	Working Details	11
6.	Results and discussion	13
7.	Future Enhancement	14
8.	Programming Code: Arduino	15
9.	Programming Code: Raspberry Pi	17
10.	Prototype	24
11.	References	27

## 1. ABSTRACT:

The AR Drone project was successfully built to an end-goal of achieving an automated landing and battery charging station for the AR Drone. Through this integrated project, we will enable individuals with severe disabilities to operate the AR Drone with more autonomy. We designed a charging station and automatic landing sequence for the AR Parrot Drone 2.0 by integrating an lightweight Arduino Micro with a Wii remote camera to detect three LEDs in a 30-60-90 triangular configuration on the landing platform. This enables distance and position feedback through trigonometric calculations. The data acquired from the camera is sent via XBee to Raspberry Pi, which processes the control signal based on LED orientation, and sends commands to AR drone directly. The drone can land automatically on the charging platform and charge in the station. Furthermore, a user-friendly GUI built in Python code on the Raspberry Pi enables a quadriplegic person to actuate the quadcopter as a surveillance tool using cursor movement and clicking.

## 2. INTRODUCTION:

Recall that in recent years, the AR Parrot Drone has been used to allow bedridden people and those with severe disabilities to virtually travel anywhere in the world. This final phase of the project uses the design development of the first two project phases and concludes the project with a more consolidated and streamlined system and the addition of battery charging via landing pad.

The Raspberry Pi enable automatic control to take over once the AR Drone with within range of the landing pad. Using a series of infrared LEDs and the camera from a Nintendo Wii remote, we can track a series of LEDs on a landing pad to determine 3-dimensional distance and rotation of the AR Drone from the landing pad. An Arduino micro onboard the AR Drone interfaces directly with the camera and an XBee to send the feedback signal to Raspberry Pi.

In normal operation, the Raspberry Pi takes commands from the GUI. However, the unique ability to process parallel code using threading in python is used to engage the closed feedback loop automatic control sequence when the Wii camera on the AR Drone detects the landing pad. If this is sensed, the Raspberry Pi uses a modified proportional control algorithm (bang-bang control with a dead spot) to move the AR Drone closer to the landing pad, and attempts to land it exactly on a set of permanent magnets that attract the feet of the AR Drone.

Although all of our control methods work and control the AR Drone automatically based on our algorithm, an improvement of the control robustness is still necessary for consistency of landing. We have successfully landed the AR Drone on the landing pad, but it requires delicate commands from the GUI to maneuver properly. For this landing pad to be useful for people with disabilities who have more difficulty

controlling AR Drone, it will be essential to identify possible non linearities in the control system and tune the existing parameters.

### 3. COMPONENTS:

**3.1. Raspberry Pi:** We built a GUI in The Raspberry Pi to control the drone. The Raspberry Pi is receiving data from camera which mounted on the Arduino micro and the drone. Most importantly, the Raspberry Pi eliminates the previous need of a computer operating system, a Propeller, and a third XBee. This consolidation makes the setup easier and the system more reliable because there is less hardware. See the comparison of the two control flow diagrams below.

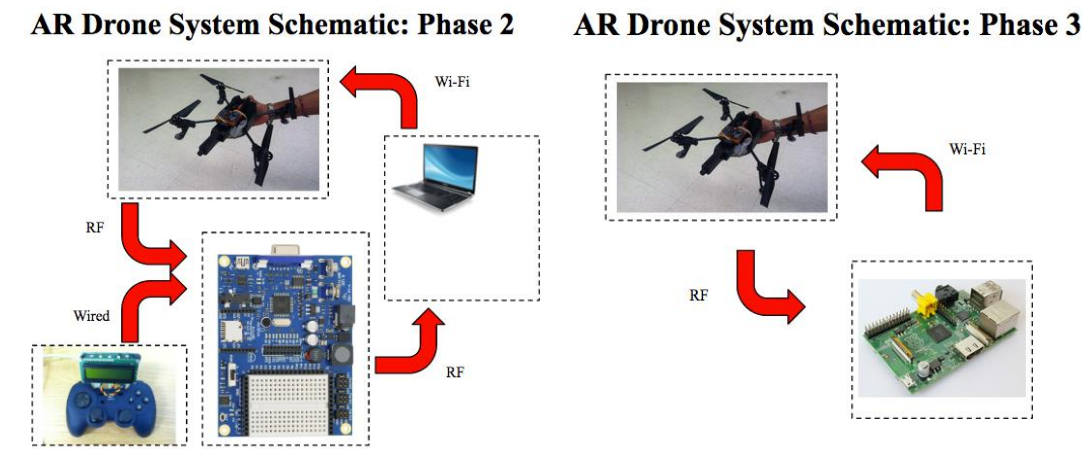


Figure 3.1.1: Control Comparison

The GUI built in Python code on the Raspberry Pi allows manual control of the drone and performs the same task as did the joystick controller in the first two phases. It is more reliable and much easier to use. Convenient ‘mouse over’ highlights buttons when the mouse is over them and further highlights them when depressed. Holding a button flies the drone continuously in a given direction. The speed applies to all of the movement commands.

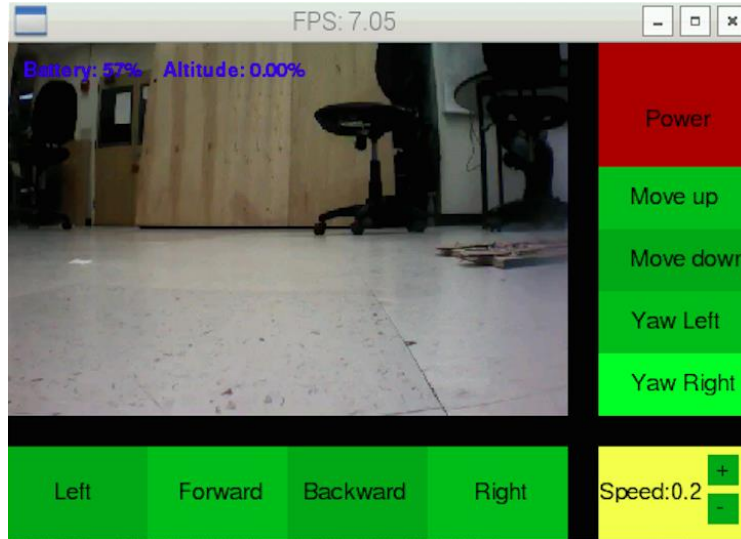


Fig. 3.1.2 Raspberry Pi GUI

With the camera data, the Raspberry Pi controls the drone to land automatically. The Raspberry Pi is a single-board computer developed in the UK by the Raspberry Pi Foundation with the intention of stimulating the teaching of basic computer science in schools. The design based on a Broadcom BCM2835 system on a chip, which includes an ARM1176JZF-S 700MHz processor, VideoCore IV GPU, and 512 megabytes of RAM. The design does not include a built-in hard disk or solid-state drive, instead relying on an SD card for booting and long-term storage.



Fig. 3.1.3 Raspberry Pi

**3.2. IR Emitter:** A high-performance infrared emitter whose peak wavelength is 940nm. These high-output infrared LEDs are a perfect light source for various kinds of security systems that need higher radiation. They also can be used in products ranging from TV remotes and auto-focus cameras to electronic gates. Our 30-60-90 triangular configuration of LEDs is ideal for spatial mapping, including distances, position, and rotation.

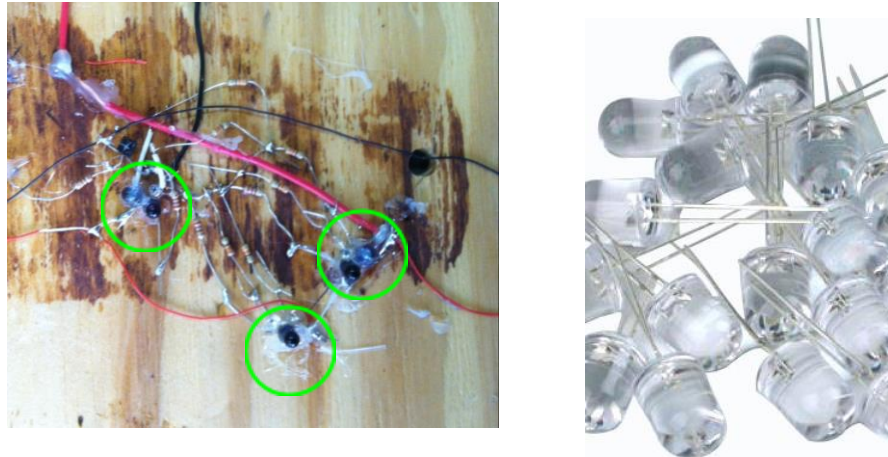


Fig. 3.2. IR LEDs

**3.3. Nintendo-Wii camera:** We physically extracted the IR camera from the Wii remote and interface it with an arduino micro. The Wii IR camera has an integrated processor which outputs the X and Y positions and size of the 4 brightest IR points that it detects.

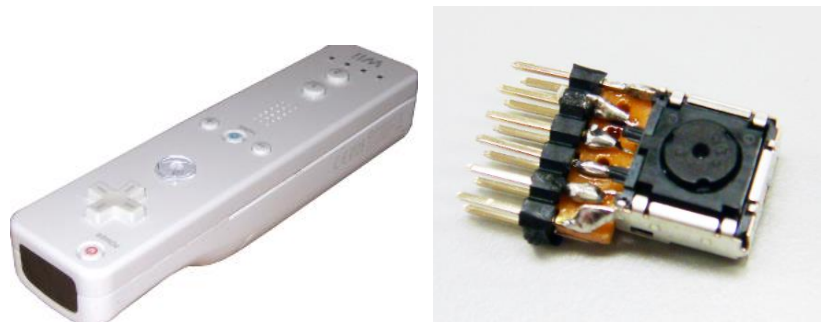


Fig. 3.3. Nintendo Wii and its camera

**3.4. Arduino Micro:** The Wii remote camera is processed by Arduino Micro. Arduino micro is one of the main controllers in the system. The Arduino Micro is a microcontroller board based on the ATmega32u4, developed in conjunction with Adafruit. It has 20 digital input/output pins (of which 7 can be used as PWM outputs and 12 as analog inputs), a 16 MHz crystal oscillator, a micro USB connection, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a micro USB cable to get started. It has a form factor that enables it to be easily placed on a breadboard. It processes the signals from camera and sends the messages to the Raspberry Pi through XBee pairs.



Fig. 3.4. ARDUINO Micro

**3.5. XBee Explorer Regulated:** This little device is used for wireless communication between Raspberry pi, Arduino micro. This SparkFun XBee Explorer Regulated makes the XBee module *so* much nicer to use, by managing the 3.3V power regulation, signal conditioning, and adds LED activity indicators (for power, RSSI and DIN/DOUT). It also level-shifts 5V signals to match the 3.3V the XBee expects.



Fig. 3.5. XBee Explorer Regulator

**3.6. SparkFun XBee Shield:** The SparkFun XBee Shield is mounted directly onto the Arduino micro and adds XBee radios an awesome way to our project. The shield form-factor mates directly with any dev board that has an Arduino standard footprint and equips it with wireless communication capabilities using the popular XBee module. This unit works with all XBee modules including the Series 1 and 2, standard and Pro versions.

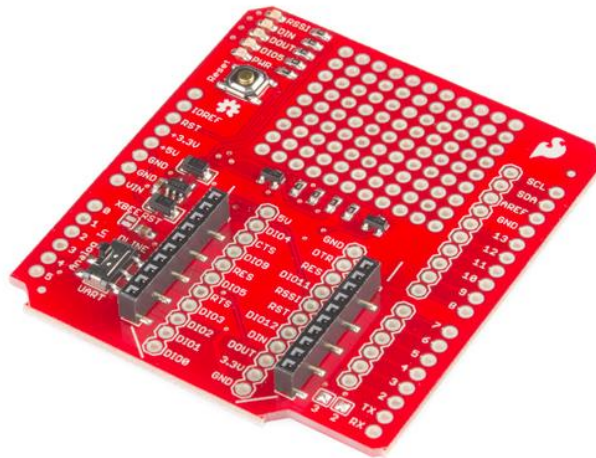


Fig. 3.6.1 SparkFun XBee Shield

Finally, this hardware was combined for placement on the AR Drone in Phase 2. It was designed to fit compactly on the AR Drone and as light as possible to avoid weight strain and wind turbulence.

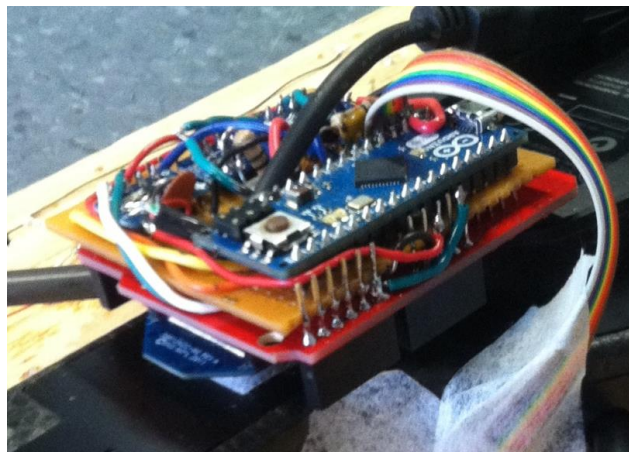


Fig. 3.6.2 SparkFun XBee Shield + Arduino Micro + necessary hardware



**3.7. Charging Platform:** We constructed a charging station with four landing pads contacting to the feet of the drone. The landing pad has copper and wires connecting to the charger. A dummy fixture was built to be plugged into the wall charger to take the place of battery contacts. Wires onboard the drone were soldered to the battery contacts that lead to this dummy fixture through the feet of the drone.

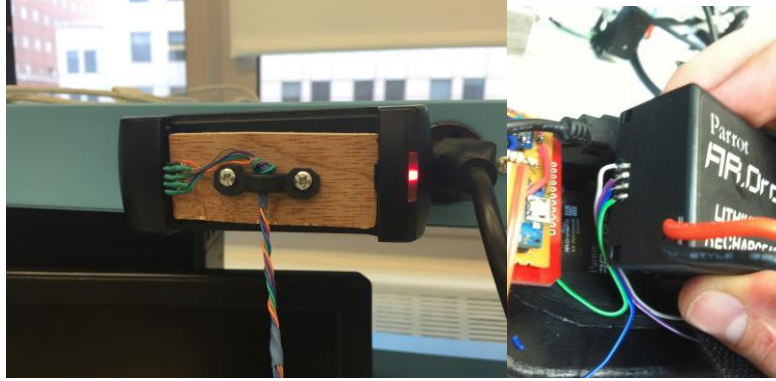


Figure 3.7.1: Charging dummy fixture; soldered wires onto the battery aboard the drone

The battery in the drone has charging wires looping to the feet. To hold the copper wire surfaces of the landing pad and the copper soldering wick surfaces of the feet together properly, solder was melted evenly over them to increase rigidity.

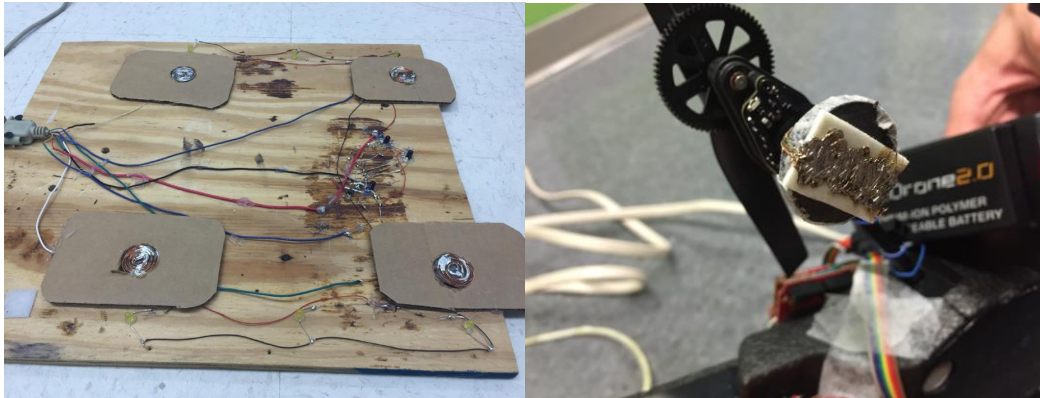


Fig. 3.7.2 Charging station and drone feet

#### 4. BILL OF MATERIALS:

The following is the list of the materials used in the project along with their price. The total is the cost of prototyping for the raspberry-controlled and auto landing drone. The greater part of the total cost was electronic components including Raspberry Pi, Arduino micro, Nintendo-Wii and XBee kit. The most expensive component is Parrot AR drone, which is free--borrowed from a team member.

Table: Bill of Materials.

Sl. No.	Component	Price per piece	Quantity	Total Price
1.	Raspberry Pi	\$31.99	1	\$31.99
2.	IR Emitter	\$2.00	6	\$12.00
3.	Nintendo wii camera	\$25.99	1	\$25.99
4.	Arduino micro	\$25.57	1	\$25.57
5.	XBee Explorer Regulated	\$17.95	1	\$17.95
6.	SparkFun XBee Shield	\$14.95	1	\$14.95
7.	Wifi dongle	\$7.99	1	\$7.99
7.	DB9 RS232 Serial Cable	\$3.49	1	\$3.49
8.	USB cable	\$4.67	1	\$4.67
9.	Yellow LED	\$0.99	6	\$5.94
10.	Parrot AR drone Quadricopter	\$0.00	1	\$0.00
	<b>TOTAL PRICE</b>			<b>\$150.54</b>

## 5. WORKING DETAILS:



### Technical Specifications Sensors:

- 3 axis gyroscope 2000°/second precision
- 3 axis accelerometer +-50mg precision
- 3 axis magnetometer 6° precision
- Pressure sensor +/- 10 Pa precision
- Ultrasound sensors for ground altitude measurement
- 60 FPS vertical QVGA camera for ground speed measurement

AR drone is a Wifi controllable drone, that currently has Iphone, Android apps to control the drone. We incorporate a traditional joystick with potentiometers and buttons to configure the device as a joystick controller for the drone.

We have incorporated potentiometers connected to analog input pins to precisely measure the joystick movements through arduino and scale them as appropriate inputs to be fed into the processing and to the drone.

### Working Explanation:

#### *Quadcopter Landing pad Design:*

The landing pad is fitted with 6 indicator Leds and 9 IR Leds (3 LEDs per cluster \* 3 clusters). The indicator Leds turn on when the Wii camera fitted in quadcopter detects the IR

leds(closeby to the landing pad). The IR leds are used to position and orient the quadcopter with reference to the landing pad. The landing pads are fitted with permanent magnets to attract the quadcopter and position it appropriately to implement the charging station.

#### *IR-Led tracking:*

The Wii camera, connected to an Arduino Micro fitted on the quadcopter. The communication between the Wii camera and the Arduino Micro are established by I2C communication protocol. The Wii camera is coded to track three coordinate points on the landing pad. The tracked coordinates, in wirelessly sent to raspberry pi through Xbee serial communication. The IR Led tracking is performed by clocking the Wii camera at 24MHz. The Arduino and Wii camera communicate at a sampling rate up to 400kHz. The Arduino micro sends data to raspberry pi at 9600 baud rate.

The raspberry pi python script computes the coordinate orientation and determines the longest side of the triangle formed by the IR Leds. The center point, the distance between the other two points and slope of the perpendicular bisector from first point.

#### *Quadcopter Control:*

The quadcopter is controlled using the GUI designed in python. The GUI is optimized in size to display the drone videos and the buttons, with options to turn right, left, back, forth, up, down, and rotation about the vertical axis (yaw).

#### *Autonomous Quadcopter landing:*

We have designed a landing pad with 3 IR led points tracked by Wii IR camera. The IR camera connected to Arduino Micro fitted on the Quadcopter sends the data of 3 coordinates on the landing pad. The landing pad, connected to the raspberry pi pin 25 lights once the IR-camera detects all the three IR-leds. Based on the current location of the quadcopter, the raspberry pi python code for automatic landing steers the quadcopter to align it with the magnets located on the landing pad through a simple bang-bang control algorithm.

#### *Mathematical Modeling:*

The Wii camera detects the IR leds on the landing pad. The IR led points on the landing pad are mathematically modelled to compute the desired back-front, left-right and down motions of the drone. The position of first point is referenced to compute the direction to be moved to orient the quadcopter right below the magnets. The range of the IR camera is 1000 along left-right direction and 750 along front-back direction. The mid-point is computed as 500 and 360. The ideal landing position of the first point is 500,360. The quadcopter is programmed to fly according to the location of the first point with respect to the reference point. The height of the quadcopter is programmed with respect to the distance between the other two points. The orientation is linked to the slope of the perpendicular bisector from the first point to the line between other two points.

## 6. Results and discussion:

We were able to implement the Raspberry pi to control the AR Drone from GUI that can be accessed easily by a quadriplegic person in a similar fashion that the phase 2 was completed with propeller. However, we removed the manual joystick and implemented all functions into the GUI. With Propeller, we utilized parallel processing loops to integrate automatic control takeover of the landing pad, and Raspberry Pi completes this method similarly with a threading method. The Raspberry Pi processes the IR LED signal and commands the AR Drone to move automatically via modified proportional (bang-bang) control. This control system still exhibits the same issues that the Propeller control did in the previous project. It lowers the AR Drone, moves it back/forth/right/left automatically until it reaches the desired place to land. Similarly to Propeller, this automatic control system sometimes fails to land the AR Drone properly because the bang-bang control algorithm is insufficient for reliable use.

Previously, the AR Drone drifted and had trouble hovering for a variety of reasons. First, we have crashed it multiple times and damaged the blades. Also, the circuit board holding the XBee, Arduino Micro, Wii camera, and circuitry is off center and causes turbulence in flight and uneven weight. Before our crashes and modifications, the AR Drone hovered without any noticeable movement in any direction. This hovering negatively affects the control system of the automatic landing pad and can cause the AR Drone to land incorrectly.

Our current Raspberry Pi system is a far better platform than the previous phase for testing the control system and improving it. Through the elimination of the joystick, Propeller, extra XBee, and computer OS, we have designed a much simpler system that passes control feedback through fewer links, resulting in a faster response time. Simpler systems also tend to be more reliable because there are fewer parts that could break during operation. Although the Raspberry Pi + onboard Arduino Micro automatic landing system is not ready to sell in the Best Buy nearest you, it is an excellent platform for an advanced controls project.

The battery charger we implemented was successful because of the high conductivity and robustness of materials used in contact areas. Soldering wick is like a copper shoe lace or even a miniature Chinese finger trap--it can be stretched to thin or pushed together to widen, and hold its shape due to the malleability of the copper. Once it was bent around the feet of the AR Drone under the magnets, we melted solder over it so it would hold its shape. Similarly to the soldering wick, a long strand of copper wire (2-3 ft) was bent in a circular fashion and placed on the magnets of the landing pad.

Because of the large bandwidth required to run the AR Drone camera on wifi, the Raspberry Pi has difficulty processing the data because it has less processing power than a conventional computer. During testing, we ran into problems when the system 'crashed' and

the AR Drone wifi communication stopped at mid-flight. This could be improved by decreasing the video feed quality of the camera as it sends data via wifi. However, it requires significant modification of the C libraries used to run the AR Drone in Raspberry Pi.

Overall, the project was successful and the Raspberry Pi provides an excellent simplified solution to the previous designs in Arduino and Raspberry Pi. With more development of the C libraries and control system, we are confident the system would work reliably.

## **7. Future Enhancement:**

First, a new properly working and undamaged AR Drone is necessary for optimally testing our control parameters. Removing the multiple directional drift will enable our AR Drone fly more precisely and therefore facilitate automatic landing control.

Similar to the recommendation of phase 2, the circuitry for the camera should be moved to the center of mass of the standalone Drone. Although the circuit boards are compact in the current configuration, a better system would feature a customised board to consolidate the surface mount chips into a board that would be lighter and easier to assemble.

Due to processing constraints of Raspberry Pi we need to modify the C libraries that link our Python code to the AR Drone wifi signal to decrease the quality of camera data. Once this is decreased the problem of system crashing should no longer occur. This would make control system testing much less frustrating.

Lastly, also similar to the recommendation of phase 2, it is necessary to use the gyro sensor or accelerometer inside of the AR Drone circuitry as an additional input in the control system loop. However, it may be difficult to recover these sensor readings using the WiFi and Processing code that connects to the AR Drone. The library for AR Drone control is quite sophisticated and changing merely a few lines of code to obtain a sensor reading may cause problems elsewhere, such as timing delays in WiFi connection. A sensor could be added to the Arduino Micro circuitry onboard the AR Drone so the landing pad 'add-on' procedure is less invasive and easier implement. This would complete the control scheme for the MIMO system and give 4 inputs to match our 4 outputs.

## 8. PROGRAMMING CODE: ARDUINO

```
// Wii Remote IR sensor
//written by Henry M. Clever, Ashwin R, Feng Wu
//Date: April 13, 2015
#include <Wire.h>           //For I2C
#include <SoftwareSerial.h> //SoftwareSerial for Xbee
// Xbee's DOUT (TX) is connected to pin 2 (Arduino's Software RX)
// Xbee's DIN (RX) is connected to pin 3 (Arduino's Software TX)
SoftwareSerial XBee(8,9); //Xbee communication with the Raspberry PI
int IRsensorAddress = 0xB0; //The address of Wii camera
int slaveAddress; //The address of the slave
//int ledPin = 13;
boolean ledState = false;
byte data_buf[16];
int i;

int Ix1,Iy1,Ix2,Iy2; //The x,y coordinates of the point 1 and 2
int Ix3,Iy3,Ix4,Iy4; //The x,y coordinates of point 3 and 4
int s; //

void Write_2bytes(byte d1, byte d2)
{
    Wire.beginTransmission(slaveAddress);
    // Serial.println("AA");
    Wire.write(d1); Wire.write(d2);
    // Serial.println("AB");
    Wire.endTransmission();
    // Serial.println("AC");
}

void setup()
{
    slaveAddress = IRsensorAddress >> 1; // This results in 0x21 as the address
to pass to TWI or i2c
    // Serial.begin(9600); // For serial printing verification
    XBee.begin(9600); //Xbee serial printing for Xbee serial
    delay(10);
    pinMode(ledPin, OUTPUT); // Set the LED pin as output
    Wire.begin(); //beginning I2C communication
    delay(10);
    // IR sensor initialize
    Write_2bytes(0x30,0x01); delay(10);
    // Serial.print("A");
    delay(5);
    Write_2bytes(0x30,0x08); delay(10);
    Write_2bytes(0x06,0x90); delay(10);
    Write_2bytes(0x08,0xC0); delay(10);
    Write_2bytes(0x1A,0x40); delay(10);
    Write_2bytes(0x33,0x33); delay(10);
    delay(100);
}

void loop()
{
    ledState = !ledState;
    if (ledState) { digitalWrite(ledPin,HIGH); } else { digitalWrite(ledPin,LOW); }
    // Serial.print("F"); // for testing I2C
}
```

```

        //IR sensor read
        Wire.beginTransaction(slaveAddress);
    //  Serial.print("X");
        Wire.write(0x36);
        Wire.endTransmission();
    //  Serial.print("B");
        Wire.requestFrom(slaveAddress, 16);          // Request the 16 byte heading (MSB
comes first)
        for (i=0;i<16;i++) {data_buf[i]=0;} // clears the buffer
        i=0;
    //  Serial.print("C");
        while(Wire.available() && i < 16) { //read 16 bytes
            data_buf[i] = Wire.read();
            i++;
        //      Serial.println(data_buf[i]);
        }
    // Read the sensor co-ordinates from the buffer
        Ix1 = data_buf[1];
        Iy1 = data_buf[2];
        s   = data_buf[3];
        Ix1 += (s & 0x30) <<4;
        Iy1 += (s & 0xC0) <<2;

        Ix2 = data_buf[4];
        Iy2 = data_buf[5];
        s   = data_buf[6];
        Ix2 += (s & 0x30) <<4;
        Iy2 += (s & 0xC0) <<2;

        Ix3 = data_buf[7];
        Iy3 = data_buf[8];
        s   = data_buf[9];
        Ix3 += (s & 0x30) <<4;
        Iy3 += (s & 0xC0) <<2;

        Ix4 = data_buf[10];
        Iy4 = data_buf[11];
        s   = data_buf[12];
        Ix4 += (s & 0x30) <<4;
        Iy4 += (s & 0xC0) <<2;

    //Convert the co-ordinates into strings

    char ix1[5];String Ix1_str;Ix1_str=String(int(Ix1));Ix1_str.toCharArray(ix1,5);
    char iy1[5];String Iy1_str;Iy1_str=String(int(Iy1));Iy1_str.toCharArray(iy1,5);
    char ix2[5];String Ix2_str;Ix2_str=String(int(Ix2));Ix2_str.toCharArray(ix2,5);
    char iy2[5];String Iy2_str;Iy2_str=String(int(Iy2));Iy2_str.toCharArray(iy2,5);
    char ix3[5];String Ix3_str;Ix3_str=String(int(Ix3));Ix3_str.toCharArray(ix3,5);
    char iy3[5];String Iy3_str;Iy3_str=String(int(Iy3));Iy3_str.toCharArray(iy3,5);
    char ix4[5];String Ix4_str;Ix4_str=String(int(Ix4));Ix4_str.toCharArray(ix4,5);
    char iy4[5];String Iy4_str;Iy4_str=String(int(Iy4));Iy4_str.toCharArray(iy4,5);
    //Serial sent to Xbee
    XBee.write(ix1);
    XBee.write('\t');
    XBee.write(iy1);
    XBee.write('\t');
    XBee.write(ix2);

```



```

XBee.write('\t');
XBee.write(iy2);
XBee.write('\t');
XBee.write(ix3);
XBee.write('\t');
XBee.write(iy3);
XBee.write('\t');
XBee.write(ix4);
XBee.write('\t');
XBee.write(iy4);
XBee.write('\n');
delay(50);
}

```

## 9. PROGRAMMING CODE: RASPBERRY PI (PYTHON):

```
#!/usr/bin/env python
```

```
"""Demo app for the AR.Drone.
```

```
This simple application allows to control the drone and see the drone's video stream.
```

```
"""
```

```

import pygame
import pygame.surfarray
import pygame.transform
import libardrone
from threading import Thread
import time
import serial
import RPi.GPIO as GPIO
import math
pygame.init()
W, H = 480, 320
screen = pygame.display.set_mode((W, H)) #initial screen object
clock = pygame.time.Clock()
running = True
drone = libardrone.ARDrone(True)
drone.reset()
flag = False
GPIO.setmode(GPIO.BCM) #setting GPIO
GPIO.setwarnings(False)
GPIO.setup(23,GPIO.OUT)
mouse=pygame.mouse.get_pos() #get mouse position

def button(x,y,w,h,c1,c2): #setting the function of button
    mouse1=pygame.mouse.get_pos()
    if x+w>mouse1[0]>x and y+h>mouse1[1]>y:
        pygame.draw.rect(screen, c1, (x,y,w,h))
    else:
        pygame.draw.rect(screen, c2, (x,y,w,h))

def manualControl():

```

```

myfont = pygame.font.SysFont("Arial",15)
global running
global flag
black=(0,0,0)
white=(255,255,255)
green=(0,160,0)
green2=(0,180,0)
green_bright=(0,255,0)
red=(160,0,0)
red_bright=(255,0,0)
blue = (0,0,255)
yellow = (255,255,80)
yellow_bright=(255,255,0)

while running:
    speed_display=drone.speed
    s="Speed:"+str(speed_display)
    mouse=pygame.mouse.get_pos()
    #right button
    button(0,260,90,60,green_bright,green)
    screen.blit(myfont.render("Right",1,black),(30,280))
    #forward button
    button(90,260,90,60,green_bright,green2)
    screen.blit(myfont.render("Forward",1,black),(110,280))
    #backward button
    button(180,260,90,60,green_bright,green)
    screen.blit(myfont.render("Backward",1,black),(190,280))
    #left button
    button(270,260,90,60,green_bright,green2)
    screen.blit(myfont.render("Left",1,black),(300,280))
    #Start button
    button(380,0,100,80,red_bright,red)
    screen.blit(myfont.render("Power",1,black),(410,40))
    #move up button
    button(380,80,100,40,green_bright,green)
    screen.blit(myfont.render("Move up",1,black),(400,90))
    #move down button
    button(380,120,100,40,green_bright,green2)
    screen.blit(myfont.render("Move down",1,black),(400,130))
    #Yaw left button
    button(380,160,100,40,green_bright,green)
    screen.blit(myfont.render("Yaw Left",1,black),(400,170))
    #Yaw right button
    button(380,200,100,40,green_bright,green2)
    screen.blit(myfont.render("Yaw Right",1,black),(400,210))
    #speed display background
    pygame.draw.rect(screen, yellow, (380,260,100,60))
    #speed control button +
    button(450,265,20,20,green_bright,green)
    screen.blit(myfont.render("+",1,black),(455,265))
    #speed control button -
    button(450,290,20,20,green_bright,green)
    screen.blit(myfont.render("-",1,black),(455,290))
    #speed display
    screen.blit(myfont.render(str(s),1,black),(380,280))

```

```

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False
    elif event.type == pygame.KEYUP or event.type == pygame.MOUSEBUTTONDOWN:
        drone.hover()
        pygame.draw.rect(screen,black,(0,240,360,20))
        pygame.draw.rect(screen,black,(360,0,20,240))
    elif event.type == pygame.MOUSEBUTTONDOWN:
        if event.button == 1:
            #if pygame.mouse.get_pressed():
            #left
            if 90>mouse[0]>0 and 320>mouse[1]>260:
                #print 'Right'
                drone.move_right()
                pygame.draw.rect(screen, yellow_bright, (0,240,90,20))
            #forward
            elif 180>mouse[0]>90 and 320>mouse[1]>260:
                #print 'Forward'
                drone.move_forward()
                pygame.draw.rect(screen, yellow_bright, (90,240,90,20))
            #backward
            elif 270>mouse[0]>180 and 320>mouse[1]>260:
                #print 'backward'
                drone.move_backward()
                pygame.draw.rect(screen, yellow_bright, (180,240,90,20))
            #right
            elif 360>mouse[0]>270 and 320>mouse[1]>260:
                #print 'left'
                drone.move_left()
                pygame.draw.rect(screen, yellow_bright, (270,240,90,20))
            #take off/land
            elif 480>mouse[0]>380 and 80>mouse[1]>0:
                if flag == False:
                    print 'take off'
                    drone.takeoff()
                    flag = True
                elif flag == True:
                    print 'land'
                    drone.land()
                    flag = False
            #move up
            elif 480>mouse[0]>380 and 120>mouse[1]>80:
                #print 'move up'
                drone.move_up()
                pygame.draw.rect(screen, yellow_bright, (360,80,20,40))
            #move down
            elif 480>mouse[0]>380 and 160>mouse[1]>120:
                #print 'move down'
                drone.move_down()
                pygame.draw.rect(screen, yellow_bright, (360,120,20,40))
            #yaw left
            elif 480>mouse[0]>380 and 200>mouse[1]>160:
                #print 'yaw left'
                drone.turn_left()
                pygame.draw.rect(screen, yellow_bright, (360,160,20,40))
            #yaw right
            elif 480>mouse[0]>380 and 240>mouse[1]>200:

```

```

        #print 'yaw right'
        drone.turn_right()
        pygame.draw.rect(screen, yellow_bright, (360,200,20,40))
#speed up
elif 470>mouse[0]>450 and 285>mouse[1]>265:
    if drone.speed <= 0.9:
        drone.speed+=0.1
#speed down
elif 470>mouse[0]>450 and 310>mouse[1]>290:
    if drone.speed >=0.2:
        drone.speed-=0.1
elif event.type == pygame.KEYDOWN:
    if event.key == pygame.K_ESCAPE:
        drone.reset()
        running = False
# takeoff / land
elif event.key == pygame.K_RETURN:
    print("return")
    drone.takeoff()
elif event.key == pygame.K_SPACE:
    print("space")
    drone.land()
# emergency
elif event.key == pygame.K_BACKSPACE:
    drone.reset()
# forward / backward
elif event.key == pygame.K_w:
    drone.move_forward()
elif event.key == pygame.K_s:
    drone.move_backward()
# left / right
elif event.key == pygame.K_a:
    drone.move_left()
elif event.key == pygame.K_d:
    drone.move_right()
# up / down
elif event.key == pygame.K_UP:
    drone.move_up()
elif event.key == pygame.K_DOWN:
    drone.move_down()
# turn left / turn right
elif event.key == pygame.K_LEFT:
    drone.turn_left()
elif event.key == pygame.K_RIGHT:
    drone.turn_right()
# speed
elif event.key == pygame.K_1:
    drone.speed = 0.1
elif event.key == pygame.K_2:
    drone.speed = 0.2
elif event.key == pygame.K_3:
    drone.speed = 0.3
elif event.key == pygame.K_4:
    drone.speed = 0.4
elif event.key == pygame.K_5:
    drone.speed = 0.5
elif event.key == pygame.K_6:

```

```

        drone.speed = 0.6
    elif event.key == pygame.K_7:
        drone.speed = 0.7
    elif event.key == pygame.K_8:
        drone.speed = 0.8
    elif event.key == pygame.K_9:
        drone.speed = 0.9
    elif event.key == pygame.K_0:
        drone.speed = 1.0
    pygame.display.update()
    time.sleep(0.05)
print("Shutting down...")
drone.halt()
print("Ok.")

def display():
    while running:
        try:
            pixelarray = drone.get_image()
            if pixelarray != None:
                surface = pygame.surfarray.make_surface(pixelarray)
                rotsurface = pygame.transform.rotate(surface, 270)
                resurface = pygame.transform.scale(rotsurface, (360,240))
                screen.blit(resurface, (0, 0))
            # battery status
            hud_color = (255, 0, 0) if drone.navdata.get('drone_state',
dict()).get('emergency_mask', 1) else (10, 10, 255)
            bat = drone.navdata.get(0, dict()).get('battery', 0)
            f = pygame.font.Font(None, 20)
            hud = f.render('Battery: %i%%' % bat, True, hud_color)
            screen.blit(hud, (10, 10))
            alt = drone.navdata.get(0, dict()).get('altitude', 0)
            g = pygame.font.Font(None, 20)
            fed = g.render('Altitude: %.2f%%' % alt, True, hud_color)
            screen.blit(fed, (100,10))
        except:
            pass
        pygame.display.update()
        pygame.display.flip()
        clock.tick(50)
        pygame.display.set_caption("FPS: %.2f" % clock.get_fps())

def automaticControl():
    ser = serial.Serial('/dev/ttyUSB0', 9600)
    global flag
    while True:
        if (ser.inWaiting!=0):
            incoming = ser.readline().strip('\x00')
            data=incoming.split('\t')
            if len(data)==8:
                X1=int(data[0],base=10)
                Y1=int(data[1],base=10)
                X2=int(data[2],base=10)
                Y2=int(data[3],base=10)
                X3=int(data[4],base=10)

```

```

        Y3=int(data[5],base=10)
        #this part takes the x and y coords and finds the distances between them for
auto control
        d3= ((X2-X1)*(X2-X1))+((Y2-Y1)*(Y2-Y1))
        d1= ((X3-X2)**2+(Y3-Y2)**2)
        d2= ((X1-X3)**2+(Y1-Y3)**2)
        if(d3 >= d2 and d3>=d1):
            x1a=X3
            y1a=Y3
            if(d2>=d1):
                x2a=X1
                y2a=Y1
                x3a=X2
                y3a=Y2
            else:
                x2a=X2
                y2a=Y2
                x3a=X1
                y3a=Y1
        elif(d2>=d3 and d2>=d1):
            x1a=X2
            y1a=Y2
            if(d3>d1):
                x2a=X1
                y2a=Y1
                x3a=X3
                y3a=Y3
            else:
                x2a=X3
                y2a=Y3
                x3a=X1
                y3a=Y1
        elif(d1>=d3 and d1>=d2):
            x1a=X1
            y1a=Y1
            if(d3>=d2):
                x2a=X3
                y2a=Y3
                x3a=X2
                y3a=Y2
            else:
                x2a=X2
                y2a=Y2
                x3a=X3
                y3a=Y3
        d3a=(((x2a-x1a)**2)+((y2a-y1a)**2))
        d1a=(((x3a-x2a)**2)+((y3a-y2a)**2))
        d2a=(((x1a-x3a)**2)+((y1a-y3a)**2))
        # print d1a, d2a, d3a
        GPIO.output(23,GPIO.LOW) #turn off yellow LEDS just before it detects
sensing of the landing pad
        if(x1a<1023 and x2a<1023 and x3a<1023): # if all leds are all detected
            GPIO.output(23,GPIO.HIGH)
            time.sleep(0.01) #pause for 10 ms
            drone.speed = 0.3
            #Automatic Landing control

```

```

if(x1a>400 and x1a<600 and y1a>260 and y1a<460):
    print("Center point found")
    #red point 1 is close to center
    #x1a found
    if(d1a>=(220*220)):          #height control, land if close
        print("LAND")
        drone.land()            #land
        flag=False
    elif (d1a<(220*220)):       #d3a = 368   d1a=400
        print("Lowering")
        drone.move_down()      #lower drone
elif((x1a-500)>100):
    drone.move_left()          #go left
    print("Move left")
    if((360-y1a)>100):
        drone.move_forward()   #go forward
        print ("Move_forward")
    elif((y1a-360)>100):
        drone.move_backward()   #go back
        print("Move Backward")
elif((500-x1a)>100):
    drone.move_right()         #go right
    print("Move right")
    if((360-y1a)>100):
        drone.move_forward()   #go forward
        print ("Move_forward")
    elif((y1a-360)>100):
        drone.move_backward()   #go back
        print ("Move_backward")
elif((360-y1a)>100):
    drone.move_forward()       #go forward
    print ("Move_forward")
elif((y1a-360)>100):
    drone.move_backward()      #go back
    print ("Move_backward")
drone.hover()

t1 = Thread(target = manualControl)
t2 = Thread(target = automaticControl)
t3 = Thread(target = display)
if __name__ == '__main__':
    t1.start()
    t2.start()
    t3.start()

```

## 10. PROTOTYPE

The pictures of the prototype are shown below.

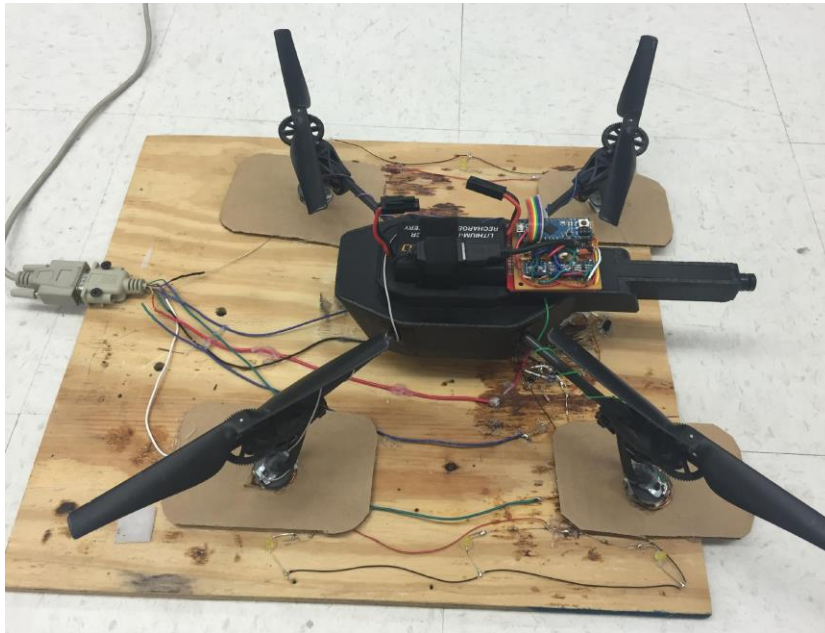


Fig. 9.1. Prototype of Raspberry Pi-controlled AR Drone.



Fig. 9.2. Modified Parrot AR drone with Arduino micro



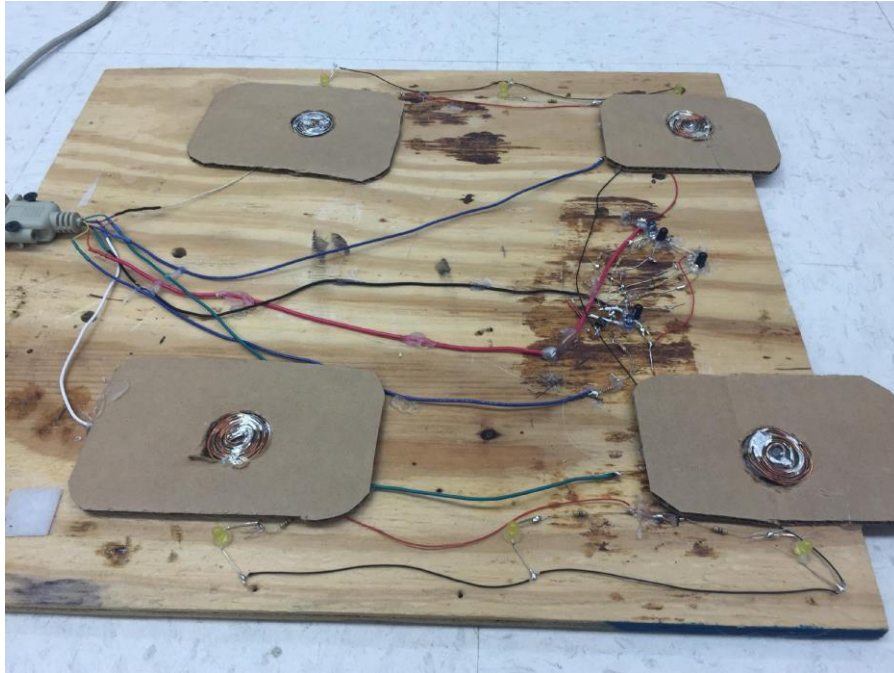


Fig. 9.3. Charging Platform

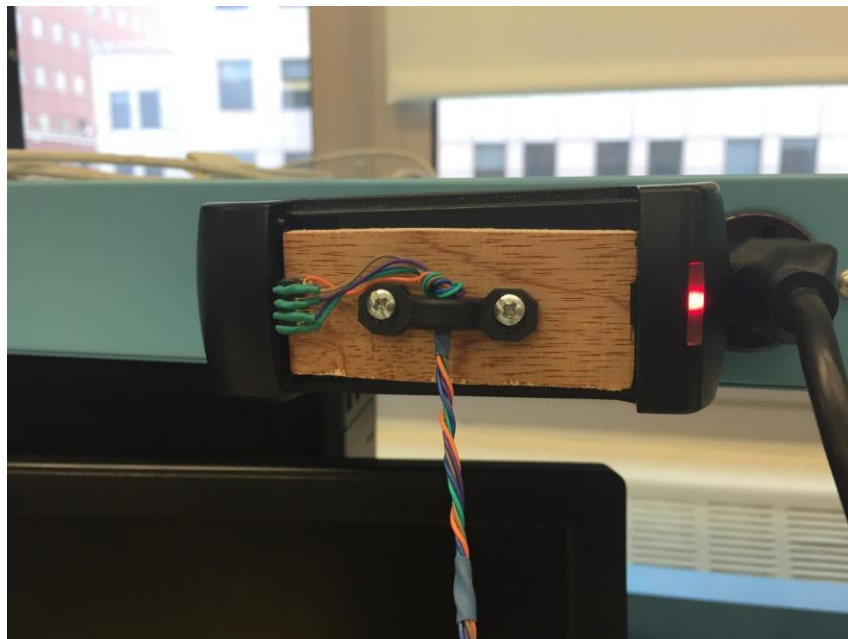


Fig. 9.4. Closer Look of Modified ChargerProject

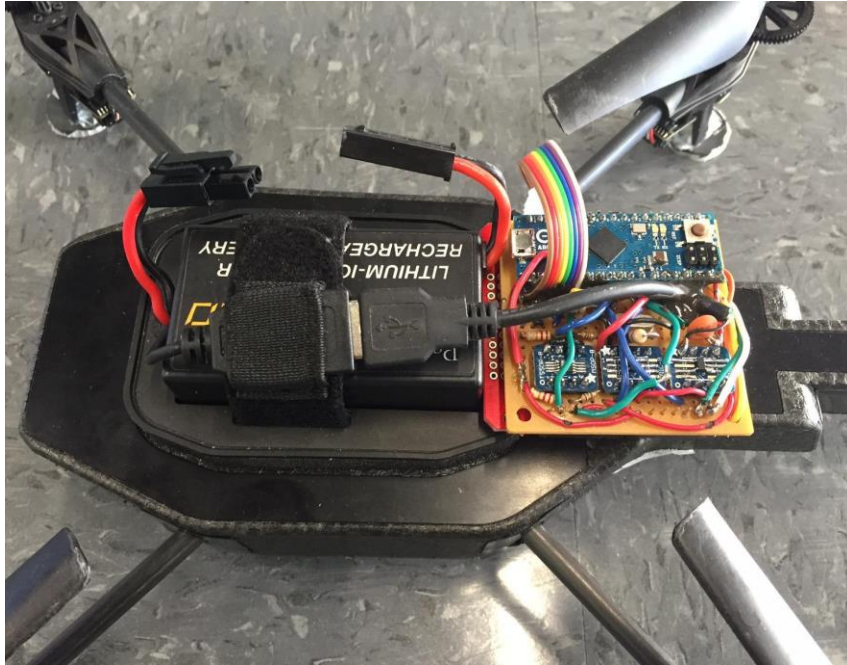


Fig. 9.5. Closer Look of Modified Parrot AR drone



Fig. 9.6. Closer Look of Mounted Camera

## **11. REFERENCES**

1. ME-7863 Advanced Mechatronics notes by Prof. Vikram Kapila.
2. ME-5643 Mechatronics notes by Prof. Vikram Kapila.
3. Steven F. Barrett, Arduino Microcontroller Processing for Everyone, Morgan & Claypool 2012.
4. Getting Started with Raspberry Pi, Matt Richardson & Shawn Wallace
5. Programming the Raspberry Pi. Getting Started with Python, Simon Monk