# AN INTELLIGENT DRONE SYSTEM TO AUTOMATE THE AVOIDANCE OF COLLISON USING AI AND COMPUTER VISION TECHNIQUES

Steven Zhang[1] and Yu Sun[2]

[1]Crean Lutheran High School, Irvine, CA 92618
[2]California State Polytechnic University, Pomona, CA, 91768

***ABSTRACT***

*People love to fly drones, but unfortunately many end up crashing or losing them. As the technology of flying drones improves, more people are getting involved. With the number of users increasing, people find that flying drones with sensors is safer because it can automatically avoid problems, but such drones are expensive. This paper describes an inexpensive UAV (unmanned aerial vehicle) system that eliminates the need for sensors and uses only the camera to avoid collisions. This program helps avoid drone crashes and losses. We used the Tello Education drone as our testing drone, which is only outfitted with a camera. Using the camera feed and transmitting that data to the program, the program will then give commands to the drone to avoid collisions.*

***KEYWORDS***

*Machine Learning, Electrical Engineering, Computer Vision, Drone*

## 1. INTRODUCTION

With the continuous development of science and technology, UAV (unmanned aerial vehicle) technology is also getting recognized by new users. [1, 2, 3, 4] With the increasing number of people participating in UAV, there are many companies specializing in manufacturing this technology. [5] But there's a problem. UAVs are small, flexible, and crucially, pilotless, which means they're vulnerable to damage or accidents during flight. [6] Using sensors, drones can automatically avoid collisions, but this feature comes with problems. First, it is expensive to build, and second, it is heavier. We designed a UAV system to effectively avoid these problems. Our UAV system uses a camera to scan the surrounding space, then the background processing system is used to calculate the most suitable solutions for directional movement for collision avoidance. This makes our UAV system affordable to build and reduces unnecessary weight.

Using existing technology, combined with our group design program, the camera scans the environment. Data is sent to the terminal to calculate and determine obstacles and the best directional path and automatic correction to avoid the obstacles. Our system doesn't need a new camera or censors, only the UAV's own camera and a program to avoid collisions and compute directional movement. Many of today's drones have automatic avoidance technology, but this usually requires special sensors. This means their costs are higher and weights are heavier. For any UAV, weight is an important consideration for flight range, and lighter models are generally more economical as well.

We searched for a drone platform that would allow us to code and have good flying control and visual feedback. After some research, we ended up choosing the Tello model as out project drone. Our choice and method of using a positive camera feed to catch and process images was inspired by the DJI Phantom 4 Pro drone. [8] Our drone can also make decisions to avoid objects. Our ETCollision drone has many useful features. First, there is no need to intervene to have the drone avoid obstacles. Second, there are no extra accessories to be installed on the drone itself. Therefore, the drone flight time won't be affected, since there is no extra weight. Third, in the future we hope we can add faster processing as a feature. For example, perhaps we can improve the drone's reaction time with a photo process. This will allow for a quicker response when the drone needs to avoid objects. Moreover, due to having a quicker response time, we can also improve the program to make it to do more complicated stunts and further reduce the possibility of crashes. Overall, we believe the ETCollision is a program that will reduce the possibility of crashing and has potential for improvements over time as well.

In two application scenarios, we demonstrated how the above combination of features increases the UAV's performance. First, we conducted a comprehensive case study on the evolution of the Tello drone, which allowed us to have a precise understanding of this model's movement and performance, especially when navigating the drone around objects quickly and smoothly without crashing. All of our data was calculated by the centimeter, and was double-checked by GPS to make sure that we had a drone up and running that would get the job done. [9, 10] What's more, we coded the drone with automatic flight control, which means if there was wind or other conditions that affected the drone, the drone will automatically adjust back into its original flight path. Therefore, the drone can operate under harsh conditions. For example, if a wind is blowing from north to south, the drone will automatically exert energy to make sure the wind does not carry it away from the desired flight path or into obstacles. Second, we analyzed the evolution of data from each time we allowed updates from the drone. We always consulted our data to make sure there were no bugs or errors at any given time. All in all, our ETCollision Tello drone has excellent performance and precise GPS data to ensure that it can securely get the job done at all times.

## 2. Challenges

In order to develop an inexpensive UAV (unmanned aerial vehicle) system that eliminates the need for sensors and uses only the camera to avoid collisions, a few challenges were identified as follows.

### 2.1. Challenge 1: Choosing a Platform and Drone to Use

Our first challenge was to decide on a platform to design our Collision project. Out of several platforms, we decided on Python, since it is easy to use compared to other program platforms. Therefore, after deciding on Python, will needed to find a drone that could let us use it to code it while still achieving excellent flight control. We chose Tello, since it offers advantages that other drones do not. For example, Tello has a GPS system and an excellent flight time of 15 minutes so we don't have to worry as much about batteries. What's more, Tello also has an HD camera that allows 30 fps feedback from the drone with very little delay.

### 2.2. Challenge 2: Setting Up Positive Video Feedback from the Drone

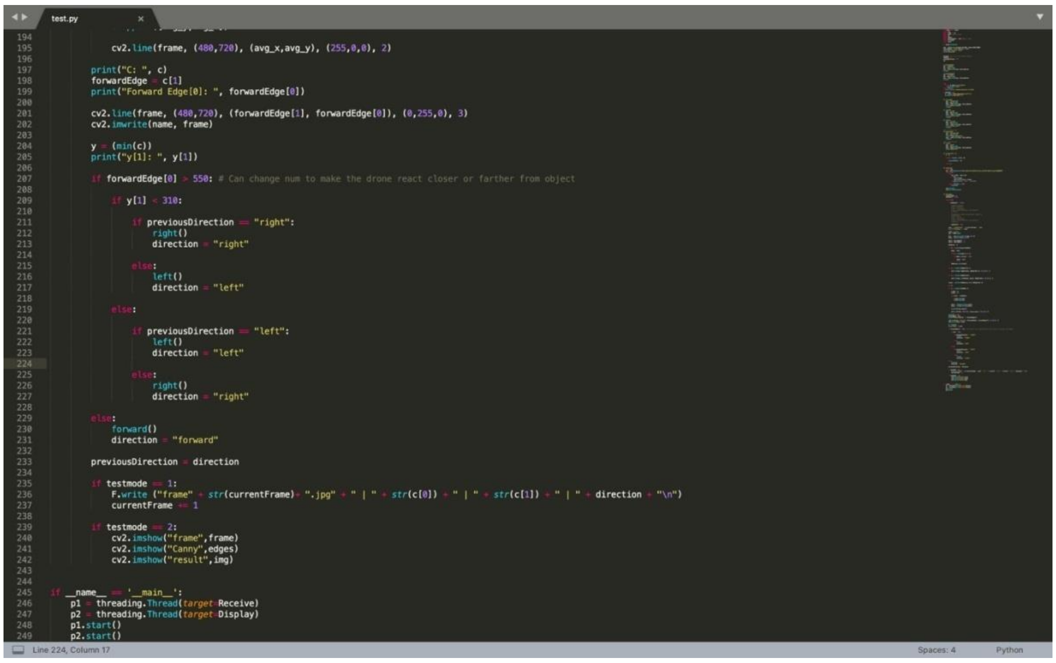
Our second challenge was to have a positive drone feed. Initially, our code was allowing the drone to detect objects, but the feedback form the drone had a 10-second delay. Therefore, the drone had a slow response time and an inefficient flight time. At this time, all the drone was doing was hovering and waiting for the video feedback to get processed. We solved this by using a positive video feedback form the drone, and allowing a window pop up on our computer to speed up the drone feedback and allow us to see the drone's flight path.

### 2.3. Challenge 2: Getting The Drone on the Market

The last challenge was buying a domain name, making a website, and entering a competition. It was difficult to choose a domain name that no one else had registered. We wanted to come up with a name that's easy to remember and find in web searches. We also had difficulty recording the screen, since we could not find software that allowed us to record our screen and our voice at the same time. However, we solved this problem by using different software. For the domain, we used "etcollision.com" so it would be easy to remember.

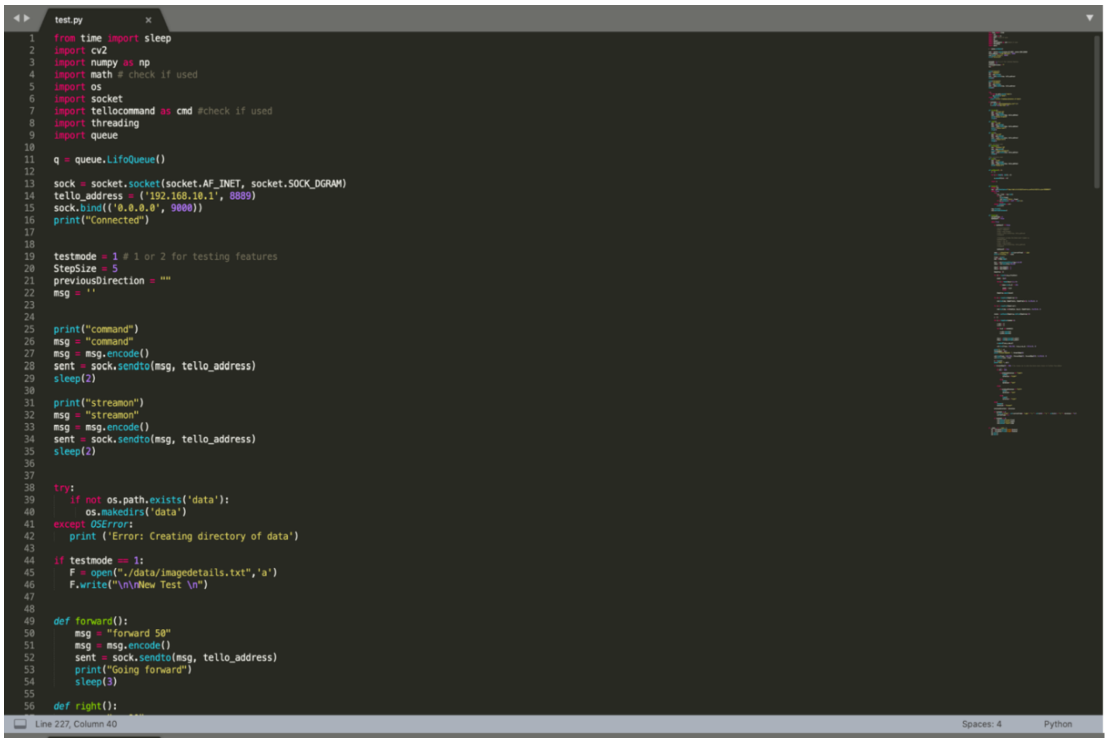
## 3. SOLUTION

Our ETCollision drone is a system that allows the drone to process video intake to avoid objects on its own. Using Python, we wrote code to process feedback form the drone. The code was scaled in centimeters, which allowed precise feedback from the drone. This special coding allowed us to see the drone's flight movement. The drone provided video feedback like a normal drone does, and it ran through a special code that allowed it to detect and process objects within its airspace. Therefore, the drone calculated varying flight courses to avoid objects. Moreover, while the drone was in the air, the command prop launched and opened a window that allowed us to see the drone's video feedback with very little delay. We could also give commands or press the emergency stop button when needed. All the drone's movements were tested to be done in a split second to make sure it would work without delay. In the future, we hope we can set up a larger data processor so the drone could remember where it has been, have faster reaction and maneuver times and navigate more smoothly and quickly.

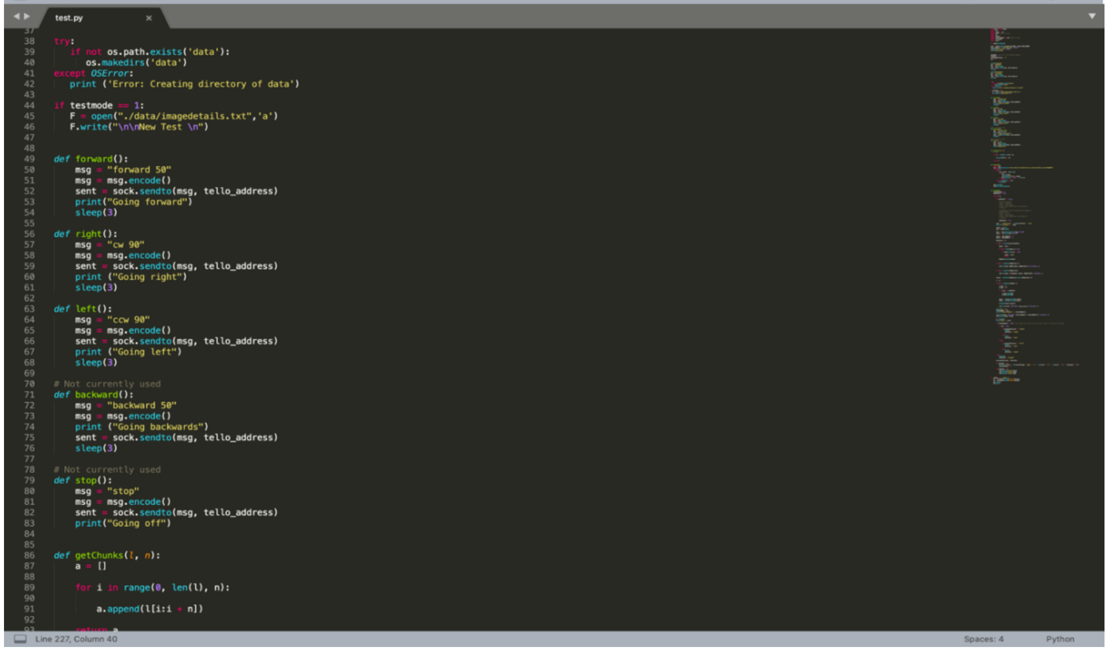


```
            cv2.line(frame, (480,720), (avg_x,avg_y), (255,0,0), 2)

        print("C: ", c)
        forwardEdge = c[1]
        print("Forward Edge[0]: ", forwardEdge[0])

        cv2.line(frame, (480,720), (forwardEdge[1], forwardEdge[0]), (0,255,0), 3)
        cv2.imwrite(name, frame)

        y = (min(c))
        print("y[1]: ", y[1])

        if forwardEdge[0] > 550: # Can change num to make the drone react closer or farther from object

            if y[1] < 310:

                if previousDirection == "right":
                    right()
                    direction = "right"

                else:
                    left()
                    direction = "left"

            else:

                if previousDirection == "left":
                    left()
                    direction = "left"

                else:
                    right()
                    direction = "right"

        else:
            forward()
            direction = "forward"

        previousDirection = direction

        if testmode == 1:
            F.write ("frame" + str(currentFrame)+ ".jpg" + " | " + str(c[0]) + " | " + str(c[1]) + " | " + direction + "\n")
            currentFrame += 1

        if testmode == 2:
            cv2.imshow("frame",frame)
            cv2.imshow("Canny",edges)
            cv2.imshow("result",img)


if __name__ == '__main__':
    p1 = threading.Thread(target=Receive)
    p2 = threading.Thread(target=Display)
    p1.start()
    p2.start()
```

Figure 1a. Code segments

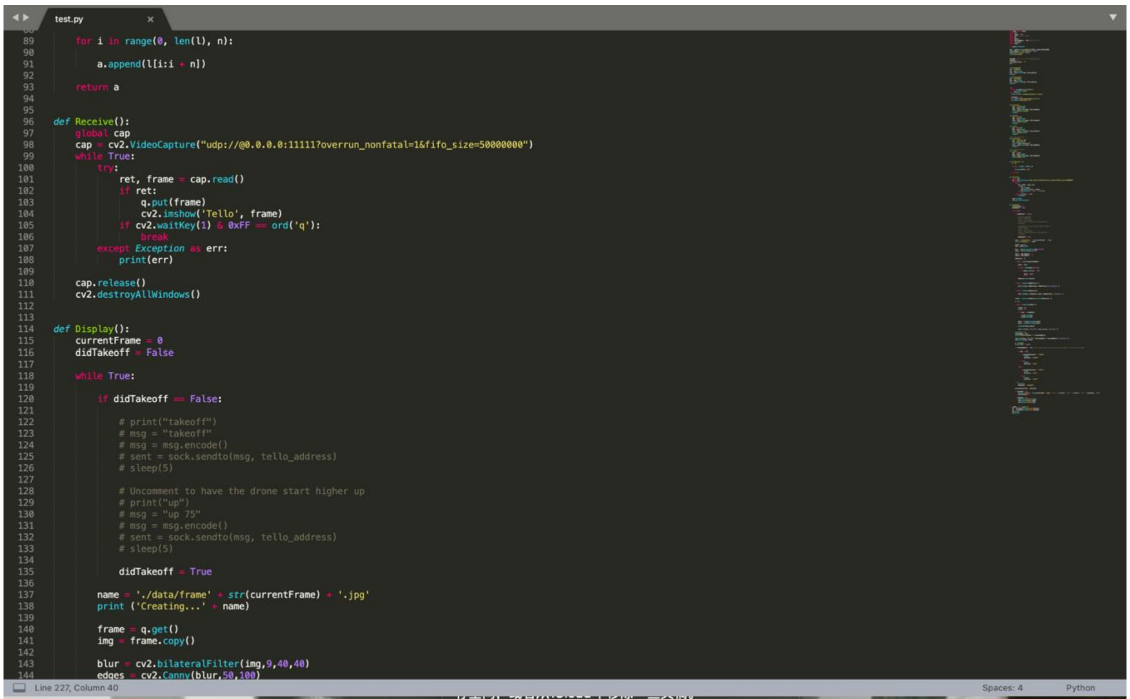


```python
from time import sleep
import cv2
import numpy as np
import math # check if used
import os
import socket
import tellocommand as cmd #check if used
import threading
import queue

q = queue.LifoQueue()

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
tello_address = ('192.168.10.1', 8889)
sock.bind(('0.0.0.0', 9000))
print("Connected")


testmode = 1 # 1 or 2 for testing features
StepSize = 5
previousDirection = ""
msg = ''


print("command")
msg = "command"
msg = msg.encode()
sent = sock.sendto(msg, tello_address)
sleep(2)

print("streamon")
msg = "streamon"
msg = msg.encode()
sent = sock.sendto(msg, tello_address)
sleep(2)


try:
    if not os.path.exists('data'):
        os.makedirs('data')
except OSError:
    print ('Error: Creating directory of data')

if testmode == 1:
    F = open("./data/imagedetails.txt",'a')
    F.write("\n\nNew Test \n")


def forward():
     msg = "forward 50"
     msg = msg.encode()
     sent = sock.sendto(msg, tello_address)
     print("Going forward")
     sleep(3)

def right():
```



```python
try:
    if not os.path.exists('data'):
        os.makedirs('data')
except OSError:
    print ('Error: Creating directory of data')

if testmode == 1:
    F = open("./data/imagedetails.txt",'a')
    F.write("\n\nNew Test \n")


def forward():
     msg = "forward 50"
     msg = msg.encode()
     sent = sock.sendto(msg, tello_address)
     print("Going forward")
     sleep(3)

def right():
     msg = "cw 90"
     msg = msg.encode()
     sent = sock.sendto(msg, tello_address)
     print ("Going right")
     sleep(3)

def left():
     msg = "ccw 90"
     msg = msg.encode()
     sent = sock.sendto(msg, tello_address)
     print ("Going left")
     sleep(3)

# Not currently used
def backward():
     msg = "backward 50"
     msg = msg.encode()
     print ("Going backwards")
     sent = sock.sendto(msg, tello_address)
     sleep(3)

# Not currently used
def stop():
     msg = "stop"
     msg = msg.encode()
     sent = sock.sendto(msg, tello_address)
     print("Going off")


def getChunks(l, n):
     a = []

     for i in range(0, len(l), n):

         a.append(l[i:i + n])

     return a
```

Figure 1b. Code segments, continued



```python
    for i in range(0, len(l), n):

        a.append(l[i:i + n])

    return a


def Receive():
    global cap
    cap = cv2.VideoCapture("udp://@0.0.0.0:11111?overrun_nonfatal=1&fifo_size=50000000")
    while True:
        try:
            ret, frame = cap.read()
            if ret:
                q.put(frame)
                cv2.imshow('Tello', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        except Exception as err:
            print(err)

    cap.release()
    cv2.destroyAllWindows()


def Display():
    currentFrame = 0
    didTakeoff = False

    while True:

        if didTakeoff == False:

            # print("takeoff")
            # msg = "takeoff"
            # msg = msg.encode()
            # sent = sock.sendto(msg, tello_address)
            # sleep(5)

            # Uncomment to have the drone start higher up
            # print("up")
            # msg = "up 75"
            # msg = msg.encode()
            # sent = sock.sendto(msg, tello_address)
            # sleep(5)

            didTakeoff = True

        name = './data/frame' + str(currentFrame) + '.jpg'
        print ('Creating...' + name)

        frame = q.get()
        img = frame.copy()

        blur = cv2.bilateralFilter(img,9,40,40)
        edges = cv2.Canny(blur,50,100)
```

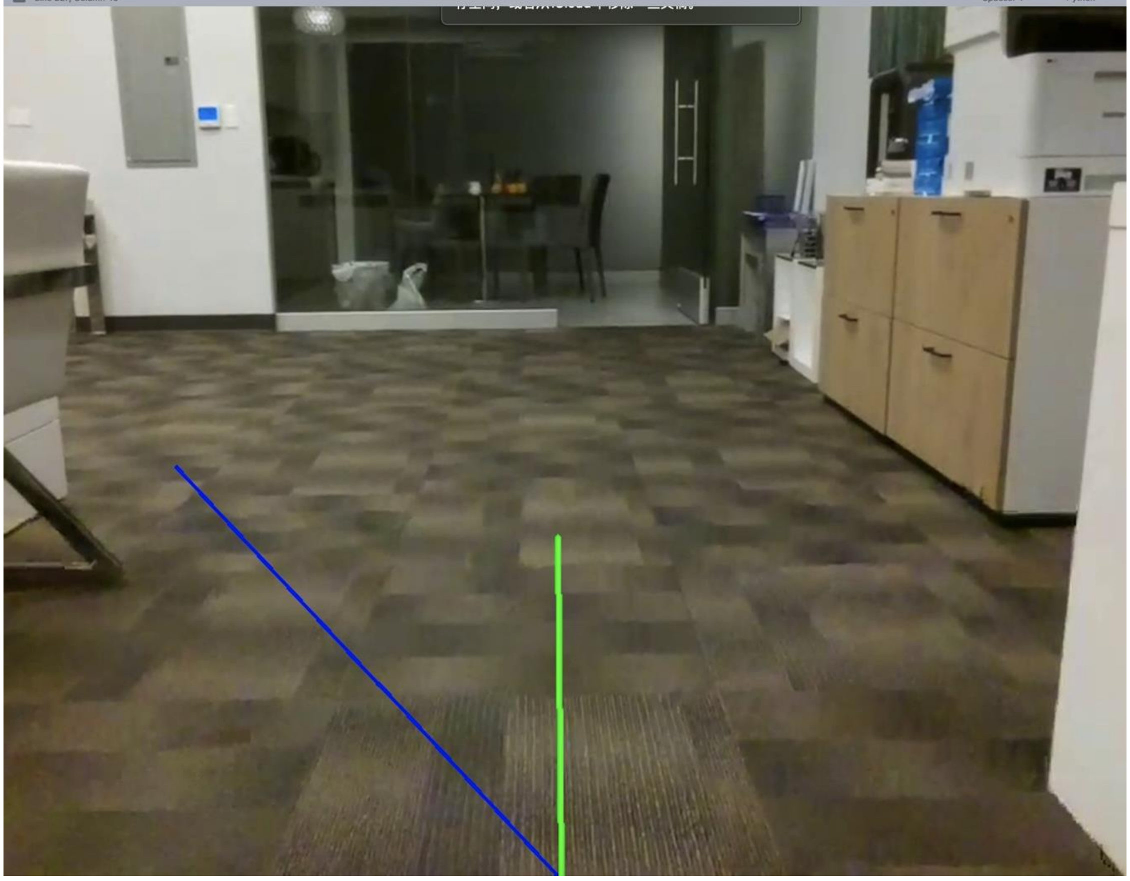


Figure 1c. Code segments and display image

1-9 import library (the video feedback from the drone will be imported to the library for later review)

25-45 load data (the data will load in a format that the system can process)

49-68 drone movement (After the program processes the movement it will start to control the drone to do the movement)

79-83 drone action (After the drone decides to make a move, all movements are ruled by the action code)

86 object detection (this is the process when the drone is processing the picture that is coming back from the video feedback)

114-242 display

245-end run all code

We decided to use Python for our platform code, since it is was the best coding platform for our drone. We did not add any additional system or hardware to do this, only add code to allow the program to process video feed from the drone for navigational purposes. See Figures 1a.-1c.

## 4. EXPERIMENT

At first, we did not have a positive video feed going to our computer to see what the drone was seeing. Therefore, we could not be sure if the drone was doing its job. We had to use a different command to allow the drone to process the video feed on its own to avoid objects. Moreover, we allowed the drone to make more precise movements without overreacting to objects. All in all, after lots of testing and coding, we were able to make the project work the way we designed it to.

We ran a percent test with our drone by directing it toward an object multiple times to calculate its passing percent and near misses. From this, we could get a percentage accuracy of its object detecting system. Moreover, we also changed the settings to make the drone more precise and flexible in certain situations, for example, varying weather conditions.

After we had conducted some experiments, we finally got an answer of 90 percent. Since our drone was relying on image processing, some of the objects could not be detected from the images alone. For example, some of the poles within our test area were not able to be picked up and processed by the drone’s video feedback. However, it was able to pick up the pole as a large shape. Moreover, it was unable to process clear glass or windows since the video feedback system can see through it. This caused the process system to think there was not an object present when at times there was glass present. Therefore, due to these situations, we only passed 90 percent of our experiment.

At first, we did not have a good video feed from our drone, which we had to fix. After doing that, we were able to move on to our second test, which was getting a percentage on avoiding objects. The problem we had with the drone was that it was not picking up some poles or clear glass. Therefore, we changed our settings on the drone to make it more precise. However, that also lowered the battery life down from the original 15 minutes of flight time. After the drone changed to high process mode, the flight time shrank to 10 minutes. In the future we hope to come up with new code to allow us to do the video feed process without using too much battery power.

## 5. Related Work

One related work is the self-recovery system used in DJI drones, which has a strong connection with the drone. The possibility of losing a connection is very small. However, they still developed a self-recovery mode to the drone to make the drone fly back to the original point of takeoff. Moreover, with this system, one can also set an altitude limit for its recovery flight. The drone will climb up to that height first, then fly back, which is a good system to make sure the drone doesn't hit anything on the way back. [11, 12]

Another related work is that DJI's drones all have average flight times of 30 minutes. Even in sport mode while at peak performance, their drones can still stay in the air for about at least 20 minutes. This is something we need to learn and study, since our drone had 15 minutes of flight time generally, and only 10 minutes of flight time while at perk performance. [13, 14]

There is also a low battery warning on DJI drones. However, since our drone doesn't have a controller, this information cannot be displayed. What we came up with to remedy this was to let the drone return by itself once its battery power hit a certain minimum. That way, we did not lose the drone or cause a crash. We also adjusted this feature further so we could also program the flight distance to make sure our drone could make it back every time. [15]

## 6. Conclusion and Future Work

We produced a special set of code to allow our drone to avoid objects in its path. To do this, we used video feedback from a camera that was already on the drone. We used Python as our primary coding platform, because it was the most compatible with our drone. Therefore, we could ensure the best performance of our code and drone at all times.

Our drone has few limitations. All a pilot might need would be to make sure to have a good battery and know how to start the drone. They must also know how to run the code and have quality video feedback showing on their computer screen. They also need to know how to land the drone and make an emergency landing in case the code has an error or the drone's video process program fails. Therefore, the limitations of our drone are few, since all these skills could be learned in under ten minutes.

One feature we hope to add is a self-recovery mode. This allows the drone to return to the place of takeoff in case of emergency. For example, if the drone loses connection with the computer, it can fly back by itself while the video feedback process system is still engaged to make sure it doesn't crash on the way back. There are lots of advantages for such a program. It could allow drones to fly out of sight and still make it back to the landing zone and can be used even if there are different signals jamming the drone's own signal. With such a self-recovery mode, our drone would not fall from a high attitude and be more likely to make it more home safely.

## References

[1] Lidynia, Chantal, Ralf Philipsen, and Martina Ziefle. "Droning on about drones—acceptance of and perceived barriers to drones in civil usage contexts." Advances in human factors in robots and unmanned systems. Springer, Cham, 2017. 317-329.

[2] Hendry, David. "“Drones Okay” Playground: Fun with Personal Drones." Designing Tech Policy.

[3] LaFay, Mark. Drones for dummies. John Wiley & Sons, 2015.

[4] Juniper, Adam. The Complete Guide to Drones: Whatever Your Budget. Wellfleet Press, 2016.

[5] Liu, Zhongli, et al. "Rise of mini-drones: Applications and issues." Proceedings of the 2015 Workshop on Privacy-Aware Mobile Computing. 2015.

[6] Vacek, Joseph J. "The next frontier in drone law: liability for cybersecurity negligence and data breaches for UAS operators." Campbell L. Rev. 39 (2017): 135.

[7] Wu, Wenhao. "React Native vs Flutter, Cross-platforms mobile application frameworks." (2018).

[8] Peppa, M. V., et al. "Photogrammetric assessment and comparison of DJI Phantom 4 pro and phantom 4 RTK small unmanned aircraft systems." ISPRS Geospatial Week 2019(2019).

[9] Gowda, Mahanth. "Bringing differential GPS to drones." Proceedings of the 3rd Workshop on Hot Topics in Wireless. 2016.

[10] Bo-tao, W. U., et al. "Testing and Analysis on differential GPS aerial drones technology." Journal of Yangtze River Scientific Research Institute 34.1 (2017): 142.

[11] Putch, A. N. D. Y. "Linear measurement accuracy of DJI drone platforms and photogrammetry." San Francisco: DroneDeploy(2017).

[12] Iqbal, Farkhund, et al. "Drone forensics: a case study on DJI phantom 4." 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA). IEEE, 2019.

[13] Salamh, Fahad E., Mohammad Meraj Mirza, and Umit Karabiyik. "UAV Forensic Analysis and Software Tools Assessment: DJI Phantom 4 and Matrice 210 as Case Studies." Electronics 10.6 (2021): 733.

[14] Xu, Fangqi, and Hideki Muneyoshi. "A Case Study of DJI, the Top Drone Maker in the World." Kindai Manag. Rev 5 (2017): 97-104.

[15] Yousef, Maryam, Farkhund Iqbal, and Mohammed Hussain. "Drone Forensics: A Detailed Analysis of Emerging DJI Models." 2020 11th International Conference on Information and Communication Systems (ICICS). IEEE, 2020.