# Autonomous Waypoint Generation System

EE551 Mitko Bozinov

# 1

# TASK DEVELOPED

Simulation Sequence of
an Autonomous Drone
Around an Object

My final project will focus on a simulation of an UAV (Unmanned Aerial Vehicle) waypoint generation system that will have said UAV start at position (1, 1, 0), fly around an object at position (6, 5, 5), while keeping a Z-position (height) of approximately 5 and return to initial position (1, 1, 1) where it will shut down and slowly descend down. A dataset will be created to track the UAV’s X, Y, Z and Yaw.

# 2

# PACKAGES USED

Packages Used in the
Development of this Project

# PACKAGES USED

- **Numpy** used to create waypoint matrices
  - [X,Y,Z,Yaw,Time]
- **‘Time’ package** imported to create timer
  - Used in tandem with desired velocity to automatically cycle through waypoints based on ‘Time’ variable
- **X,Y,Z,** and Yaw were then pulled from the desired matrix position to create a waypoint that would automatically change with time
- **Matplot** was used to verify flight trajectory

# 3

# FUNCTIONS DEVELOPED

Functions Created to
Develop Performance

# FUNCTIONS DEVELOPED

To begin the code, the user will put in the position of the object using the (xr, yr, zr) position notation within the code. Using this location and its initial position (x0, y0, z0), the UAV can calculate a target angle, theta, to use to find the adjusted x and y values, shown below in the three following equations. Using the numpy and math libraries, I was able to come up with the following:

**EQ**: yaw0 = numpy.arctan2((yr-y0), (xr-x0))
**EQ**: theta = numpy.arctan2((y0-yr), (x0-xr))
**EQ**: xAdj = xr + R*math.cos(theta)
**EQ**: yAdj = yr + R*math.sin(theta)

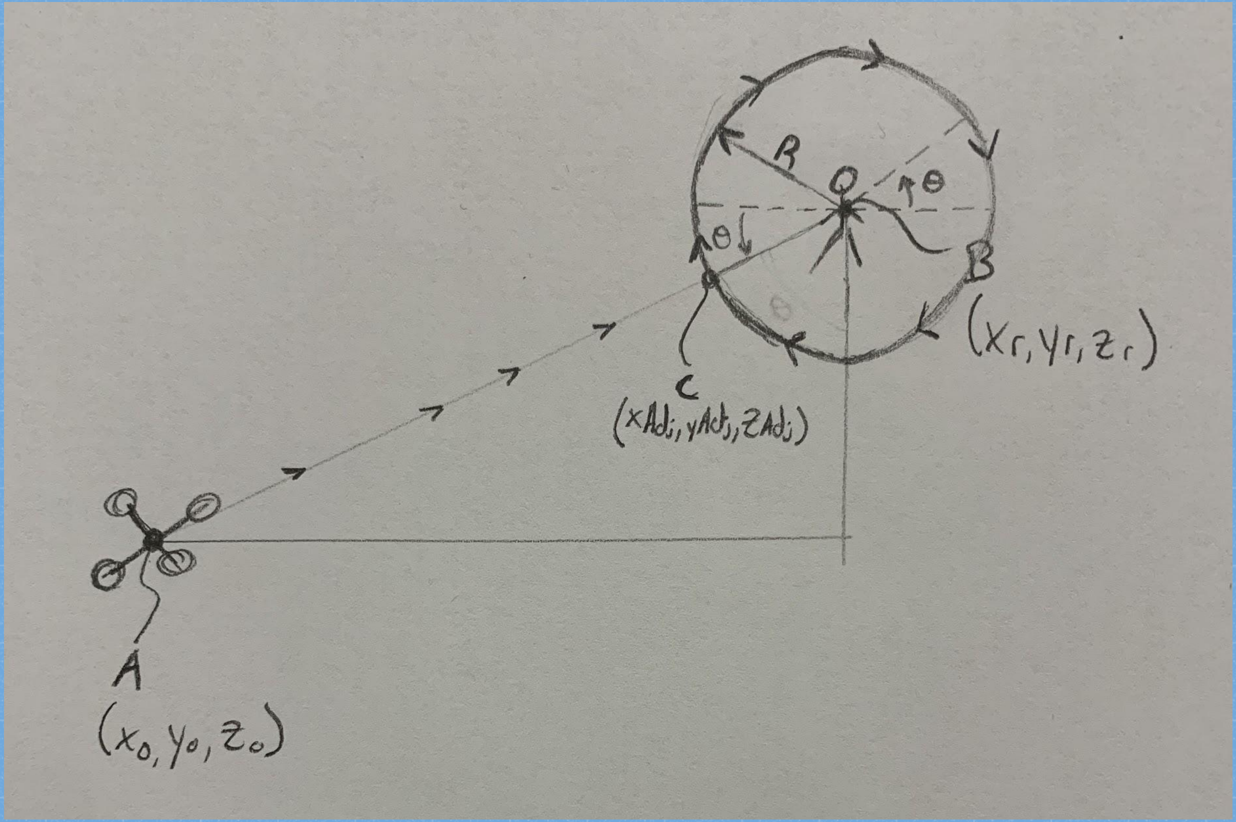
When the UAV reaches the adjusted values, it moves into the second phase of the path, circling the object. The value of omega is the angular velocity the UAV will have while circling the object. Then the UAV will return to its original position.

**EQ**: D1 = math.sqrt((xAdj - x0)**2 + (yAdj - y0)**2 + (zr - z0)**2)
**EQ**: Dc = 2*math.pi*R
**EQ**: D2 = math.sqrt((xAdj - x0)**2 + (yAdj - y0)**2+(zr - z0)**2)

The values x and y are the calculated x and y reference values that the UAV should take, R is the radius from the object, n is the desired number of times the UAV should circle the object, u is the current simulation time and t1 is the first phase’s time. After completing n number of circles, the UAV enters the third phase and returns back to its initial position.

```
# Heading
yaw0 = numpy.arctan2((yr-y0), (xr-x0))

# Radius of Target
R = 1

theta = numpy.arctan2((y0-yr), (x0-xr))

# Solving for adjusted x and y values
xAdj = xr + R*math.cos(theta)
yAdj = yr + R*math.sin(theta)

# from here i will be attempting to find the distances for the code
# first lets find define then D1 is there Dc is the since loop D2 is back
# L is the total distance of the travel

# Towards Target
D1 = math.sqrt((xAdj - x0)**2 + (yAdj - y0)**2 + (zr - z0)**2)

# Circular Path
Dc = 2*math.pi*R

# Back From Target
D2 = math.sqrt((xAdj - x0)**2 + (yAdj - y0)**2+(zr - z0)**2)

N = 2  # N is the number of rotations around the object
```

# FUNCTIONS DEVELOPED: Conceptual Sketch



# FUNCTIONS DEVELOPED: To Target

Once the points per section is completed the creation of the waypoint matrix can begin. This section utilizes a for loop to create each row of the matrix. The for loop counts from ii=1 to the sum of the points, *t1+t2+t3*. Within the for loop are three sections separated by if statements. If the count is less than *t1* then the formulas for a slope are utilized to create a straight line path from $X_0, Y_0$ to $X_{adj}, Y_{adj}$. Therefore, by using a constant value, when armed, the UAV will rise up quickly from the ground and to its target Z coordinate. The equations used are what create the waypoint matrix, referred to in the code as *waypoints*. The waypoint matrix constantly generated new rows with each iteration by calling out: *waypoints(ii,N)=E* where *N* = 1, 2, 3, or 4 to correspond with X, Y, Z, and Yaw respectively and *E* is the equation used to solve for that value of the matrix.

EQ: waypoints(ii,0) = (xAdj-x0)/t1*ii
EQ: waypoints(ii,1) = (yAdj-y0)/t1*ii
EQ: waypoints(ii,2) = zr
EQ: waypoints(ii,3) = yaw0

In the above mentioned equations a form of the slope formula (y = mx+b) is utilized. The value for m is the difference of the two points, $X_{Adj}$ and $X_0$, divided by the total number of points. The value for x is *ii* so at *ii+t1* the value would return as $X_{adj}$. Currently, $X_0$ and $Y_0$ are assumed to be equal to 0.

```
# for loop only has a few small changes
# indexing starts at 0 not 1
for ii in range(0, t1+t2+t3):
    # To the target
    if ii < t1:
        waypoints[ii, 0] = (xAdj-x0)/t1*ii+x0
        waypoints[ii, 1] = (yAdj-y0)/t1*ii+y0
        waypoints[ii, 2] = zr
        waypoints[ii, 3] = yaw0
```

# FUNCTIONS DEVELOPED: Around Target

Next, if the count was greater than *t1* and less than *t1+t2* then it was in the circular region. In this section the equation for a circle is parameterized in terms of X and Y position. The equations for these points can be seen in Equations below:

EQ: waypoints[ii,0] = xr + R*math.cos(theta + omega*(ii-t1))
EQ: waypoints[ii,1] = yr + R*math.sin(theta + omega*(ii-t1))
EQ: waypoints[ii,2] = (zt-zr)/(t2)*(ii-t1)+zr
EQ: waypoints[ii,3] = (N*2*math.pi)/t2*(ii-t1)+yaw0

In Equations above the value *ii-t1* is used so that the count is from 0 to *t2.* Currently, the Z axis position is the same as in the linear section but may be changed later to have the UAV linearly move up.

```
# Around the target
elif (t1 <= ii) and (ii < t1+t2):
    waypoints[ii, 0] = xr + R*math.cos(theta + omega*(ii-t1))
    waypoints[ii, 1] = yr + R*math.sin(theta + omega*(ii-t1))
    waypoints[ii, 2] = (zt-zr)/(t2)*(ii-t1)+zr
    waypoints[ii, 3] = (N*2*math.pi)/t2*(ii-t1)+yaw0
    #waypoints[ii,3] = waypoints[ii-1,3] + omega * timebtw
```

# FUNCTIONS DEVELOPED: Coming Back to Initial Position

The final section of the code is the return flight from the circular path back to the $X_0$, $Y_0$ position. This section follows a similar form to the first section but going from the $X_{adj}$, $Y_{adj}$ position back to $X_0$,$Y_0$. The equations can be seen below.

**EQ:** waypoints(ii,0) = (x0-xAdj)/(t3)*(ii-t1-t2)+xAdj
**EQ:** waypoints(ii,1) = (y0-yAdj)/(t3)*(ii-t1-t2)+yAd
**EQ:** waypoints[ii,2] = (zr-zt)/(t3)*(ii-t1-t2)+zt
**EQ:** waypoints(ii,3) = yaw0 + N*2*math.pi

After each of the aforementioned sections of code another critical variable, Yaw, is solved for.

The final column is time which is accounted for by using the equation listed below:

**EQ:** waypoints[ii,4] = timwbtw*ii

The total time needed for flight is 94 seconds.

```
    # Coming Back
    else:
        waypoints[ii, 0] = (x0-xAdj)/(t3)*(ii-t1-t2)+xAdj
        waypoints[ii, 1] = (y0-yAdj)/(t3)*(ii-t1-t2)+yAdj
        waypoints[ii, 2] = (zr-zt)/(t3)*(ii-t1-t2)+zt
        waypoints[ii, 3] = yaw0 + N*2*math.pi
    waypoints[ii, 4] = timebtw*ii
ii+1
```

4

# RUNNING THE PROGRAM

Steps Involved in Running Program
to Create Appropriate Plots

# INSTRUCTIONS TO RUN THE PROGRAM

- The UAV’s initial position can be set as follows on x0, y0, and z0.
- The desired position of the UAV’s target can be set as follows by variables xr, yr, zr, zt.
- Using this location and its initial position (x0, y0, z0), the quadcopter can calculate a target angle, theta, to use to find the adjusted x and y values.

```
# Drone Flight Path
# starting position
x0 = 1
y0 = 1
z0 = 0
# desired position and target
xr = 6
yr = 5
zr = 1  # to prevent skating
zt = 5  # top of object
```

# INSTRUCTIONS TO RUN THE PROGRAM

The program can print the target [X, Y, Z, Yaw, Time] in real time as the UAV is conducting it’s flight. The ‘*’ is used to graph the position on the plot

- **print(target)** will continuously print the UAV’s position throughout the 94 seconds.
- **plt.plot(x, y, '*')** # X-Y Position (m) Viewed From Above
- **plt.plot(z, '*')** # Z position (m) vs Array Position
- **plt.plot(yaw, '*')** # Yaw Angle (rad) vs Array Position
- **plt.show()** # Show the plot

Each **plt.plot()** can be uncommented to provide the users desired plot

GitHub Link: https://github.com/mbozinov/EE-551-Python-FinalProject

# 5

# CHALLENGES & SOLUTIONS

Challenges Faced and
Solution Found

# CHALLENGES

| | PROBLEMS | SOLUTIONS |
| --- | --- | --- |
| 1 | Other than a couple of courses taken in control systems, I wasn’t well versed in creating equations capable in keeping an object stable throughout a series of movements. The simulation used equations well known in control theory, however not well know by myself. | Equation development was made possible through the reference of existing flight control systems created in C++ and with the reference on Control Systems studies done by two IEEE papers. Yaw, Theta, Xadj and Yadj values were calculated by using equations provided by IEEE study. |
| 2 | Difficult to generate visual appeal with the current values the program was generating.<br>Matrix order was needed in order to make the correct equations needed within control theory. | Utilizing the Numpy library, I was able to reorganize my matrices in order to make the correct calculations with the utilized input equations that are needed to stabilize the UAV and calculate the target angle, theta, and yaw to use to find the adjusted x, y and z values. |

7

# PERFORMANCE EVALUATION

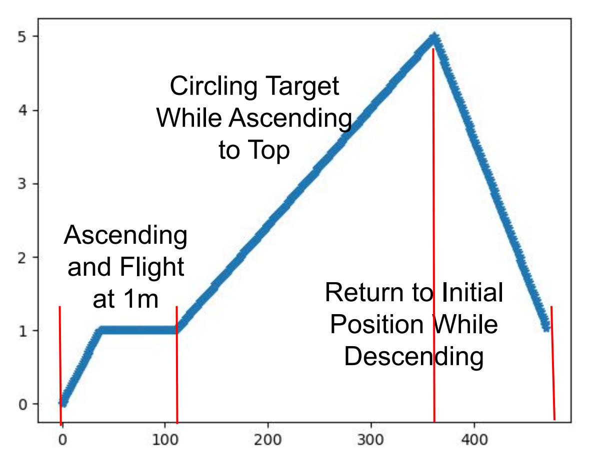
## The Results

# PERFORMANCE EVALUATION

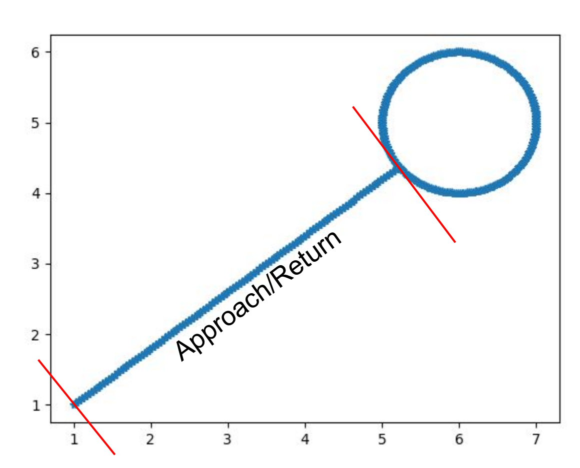
**Initial Position: (1m,1m), On the ground (Z = 0)**

**Object Position: (6m,5m), Height (5m)**

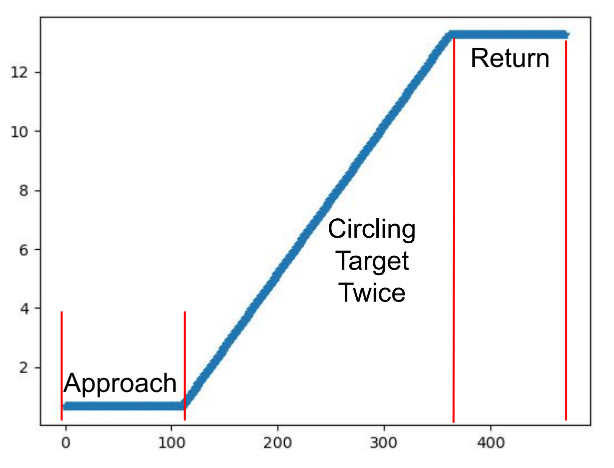
**Z position (m) vs Array Position**




**X-Y Position (m) Viewed From Above**




**Yaw Angle (rad) vs Array Position**



# Dataset Created using the Waypoint Generation

**Initial Position**

Column 1: X-position

Column 2: Y-position

Column 3: Z-position

Column 4: Yaw

Column 5: Time Passed (sec)

[ X Y Z Yaw Time ]

```
================== RESTART: C:\Users\Mitko\Desktop\drone.py ==
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.          1.          1.          0.67474094 0.         ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
[1.03835574 1.03068459 1.          0.67474094 0.2        ]
```

# Dataset Created using the Waypoint Generation

- Total time taken to complete fly-around is 94 seconds.

- Peak height reached at 72 seconds in

- Final Approximate position is (1, 1, 1)
    - Z-position will end at 1 because drone will shut down and slowly descend at that position

[ X Y Z Yaw Time ]

```
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
[ 1.03835574 1.03068459 1.03636364 13.24111156 94.        ]
```

8

# CONCLUSION

Final Thoughts,Insights
and Recommendations

# KEY FINDINGS


Total time taken to complete fly-around is 94 seconds.

Peak height reached at 5m at around 72 seconds.

UAV reaches the object, circles it and returns back to initial position.

# RECOMMENDATIONS

- With the increase in more control variable within the closed loop system, the wavepoint generation can be more accurate and more versatile by even introducing obstructing objects that the UAV will have to avoid.
- Data generation takes a very long time and would be better if it could be exported as a .csv file alongside the graphs
- Plot generation has to be exported as separate graphs in order get the correct scale for MatPlot. It would be better for next time that graph visuals can be plotted on the same plot in order to visually compare data at the same time.

# 9

# CODE

GitHub Link:
https://github.com/mbozinov/EE-551-Python-FinalProject

Full Code as Well as Attached GutHub Link

1)

```python
# pi is math.pi
# round is int(round(Value))
# need numpy to do matrices easier
# need matplotlib to make plots

# Need this for atan2
import math  # for most math functions
import numpy  # for arctan2 and matrices
import matplotlib.pyplot as plt  # for plotting
import time  # needed to count the time


# Drone Flight Path
# starting position
x0 = 1
y0 = 1
z0 = 0
# desired position and target
xr = 6
yr = 5
zr = 1  # to prevent skating
zt = 5  # top of object


# Heading
yaw0 = numpy.arctan2((yr-y0), (xr-x0))

# Radius of Target
R = 1

theta = numpy.arctan2((y0-yr), (x0-xr))

# Solving for adjusted x and y values
xAdj = xr + R*math.cos(theta)
yAdj = yr + R*math.sin(theta)

# from here i will be attempting to find the distances for the code
# first lets find define then D1 is there Dc is the since loop D2 is back
# L is the total distance of the travel

```

2)

```python

# Towards Target
D1 = math.sqrt((xAdj - x0)**2 + (yAdj - y0)**2 + (zr - z0)**2)

# Circular Path
Dc = 2*math.pi*R

# Back From Target
D2 = math.sqrt((xAdj - x0)**2 + (yAdj - y0)**2+(zr - z0)**2)

N = 2  # N is the number of rotations around the object

# Total Length
L = D1 + D2 + N*Dc

# now lets define the max speed, v, and the points per distance we want.
v = 0.25  # m/s
ppm = 20  # points/meter

# total points
Points = ppm*L  # points

# total time for the flight
totaltime = L/v  # seconds
# time to get to each point
timebtw = totaltime/Points

# Resolution
# to rise
t0 = 1
# To target
t1 = int(round(ppm*D1, 0))
# time to circle
t2 = int(round(ppm*Dc*N, 0))
# Return time
t3 = int(round(ppm*D2, 0))

```

3)

```python

# Circle calculation
omega = 2*math.pi/(t2/N)

# zeros matrix of 5 columns and t1+t2+t3 rows
waypoints = numpy.zeros((t1+t2+t3, 5))
# The new fifth column will be the time

# for loop only has a few small changes
# indexing starts at 0 not 1
for ii in range(0, t1+t2+t3):
    # To the target
    if ii < t1:
        waypoints[ii, 0] = (xAdj-x0)/t1*ii+x0
        waypoints[ii, 1] = (yAdj-y0)/t1*ii+y0
        waypoints[ii, 2] = zr
        waypoints[ii, 3] = yaw0
    # Around the target
    elif (t1 <= ii) and (ii < t1+t2):
        waypoints[ii, 0] = xr + R*math.cos(theta + omega*(ii-t1))
        waypoints[ii, 1] = yr + R*math.sin(theta + omega*(ii-t1))
        waypoints[ii, 2] = (zt-zr)/(t2)*(ii-t1)+zr
        waypoints[ii, 3] = (N*2*math.pi)/t2*(ii-t1)+yaw0
        # waypoints[ii,3] = waypoints[ii-1,3] + omega * timebtw
    # Coming Back
    else:
        waypoints[ii, 0] = (x0-xAdj)/(t3)*(ii-t1-t2)+xAdj
        waypoints[ii, 1] = (y0-yAdj)/(t3)*(ii-t1-t2)+yAdj
        waypoints[ii, 2] = (zr-zt)/(t3)*(ii-t1-t2)+zt
        waypoints[ii, 3] = yaw0 + N*2*math.pi
    waypoints[ii, 4] = timebtw*ii
ii+1
```

4)

```python

# Initialize the time and elapsed time
Start_Time = time.time()
Elapsed_Time = time.time() - Start_Time


def find_nearest(array, values):
    values = numpy.atleast_1d(values)
    indices = numpy.abs(numpy.subtract.outer(array, values)).argmin(0)
    out = array[indices]
    return out if len(out) > 1 else out[0]


# Finding the time, then going to the right line in the matrix
while Elapsed_Time < totaltime:
    Elapsed_Time = time.time() - Start_Time
    target_time = find_nearest(waypoints[:, 4], Elapsed_Time)
    position = int(round(target_time/timebtw, 0))
    target = waypoints[position, :]
    Elapsed_Time = time.time() - Start_Time
    print(target)

x = waypoints[:, 0]  # x values
y = waypoints[:, 1]  # y values
z = waypoints[:, 2]  # z values
yaw = waypoints[:, 3]  # yaw values

plt.plot(x, y, '*')  # X-Y Position (m) Viewed From Above
# plt.plot(z, '*')  # Z position (m) vs Array Position
# plt.plot(yaw, '*')  # Yaw Angle (rad) vs Array Position

plt.show()  # show the plot

```

# 10

# REFERENCES

Works Cited for the
Following Project

# REFERENCES


1. CrazyS is an extension of the ROS package RotorS, aimed to modeling, developing and integrating the Crazyflie 2.0
   a. https://github.com/gsilano/CrazyS

2. CrazyS: A Software-In-The-Loop Platform for the Crazyflie 2.0 Nano-Quadcopter
   a. https://ieeexplore.ieee.org/document/8442759

3. Unmanned Aerial Vehicle Path Following: A Survey and Analysis of Algorithms for Fixed-Wing Unmanned Aerial Vehicles
   a. https://ieeexplore.ieee.org/abstract/document/6712082