|  |  |  |  |
| --- | --- | --- | --- |
| Artifact ID:  CODE-001 | Artifact Title:  Code for Tracking Simulation | |  |
| Revision:  1.0 | Revision Date:  6 NOV 2019 | |
| Prepared by:  Autumn Twitchell | | Checked by:  Joe Hansen |
| Purpose:  This code is used to simulate the tracking concept of our in-flight vehicle. | | |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision: | Revised by: | Checked by: | Date: |
| 1.0 | Autumn Twitchell | Joe Hansen | 6 NOV 2019 |

# References

|  |  |  |
| --- | --- | --- |
| Artifact ID: | Revision: | Title: |
| N/A | N/A | N/A |

# These files are found in box under the 2019-2020-Capstone-15/Concept Development/Concept Development Packet/Concept\_Definitions/Tracking/Controls\_Simulations folder.

# ControlsMain.py

'''

controls main

- this code runs with the DrawSystem class and ControlsParameters.py

'''

**import** hwcounter **as** counter

**import** time

**from** DrawSystem **import** DrawSystem

**import** ControlsParameters **as** cp

**import** numpy **as** np

**from** matplotlib **import** pyplot **as** plt

**import** matplotlib.patches **as** mpatches

**if** \_\_name\_\_ == '\_\_main\_\_':

ds = DrawSystem()

*# initialize target and home coordinates*

target = [cp.x[0],cp.y[0]]

home = [cp.x[0],cp.y[0]]

t = cp.t\_begin

clock\_cycles = 8e10

*# draw initial plane and fov*

ds.draw\_airplane(home)

ds.draw\_fov(target)

ds.init\_flag = **False**

t\_temp = 0

start\_time = counter.count()

**while** t < cp.t\_end :

*# Add the patch to the Axes*

plt.pause(.005)

t = t + 1

t\_temp = t\_temp + 1

*# update home coordinates (tracking plane)*

home[0] = cp.x[t] + 105

home[1] = cp.y[t] - 15

*# update plane and fov with new coordinates*

ds.draw\_airplane(home)

*#12.5 is about half a second*

**if** t\_temp > 12.5:

t\_temp = 0

*# update target coordinates (tracking target)*

target[0] = cp.x[t]

target[1] = cp.y[t]

ds.draw\_fov(target)

elapsed\_time = counter.count\_end()-start\_time

print(elapsed\_time)

plt.show()

# ControlsParameters.py

'''

Controls Parameters

'''

**import** numpy **as** np

*# values that can be changed*

linspace\_size = 5000 *# largest value for linspace*

x\_center = 1395 *# x-coordinate for center of airplane's path*

y\_center = 960 *# y-coordinate for center of airplane's path*

radius = 500 *# radius of our current path for the airplane*

t\_begin = 0 *# start time for the while loop in main function*

t\_end = 5000 *# stop time for the while loop in main function*

plot\_width = 2790

plot\_height = 1920

*# parameters for shape of airplane*

long\_l = 100

med\_l = 80

small\_l = 20

*# values that cannot be changed*

fov\_len = 127 *# this is the width of the square for the field of view (fov)*

center\_of\_rect = int(fov\_len/2) *# this allows us to know where the center of the fov is*

*# values that help to create the airplane's path*

N = np.linspace(0, linspace\_size, 5001) *# number of points*

theta = (2\*N\*np.pi)/linspace\_size

*# path of airplane*

x = 2\*radius\*np.cos(theta) + x\_center

y = radius\*np.sin(theta) + y\_center

# DrawSystem.py

'''

DrawSystem class

- draws airplane, fov square, and point in the middle of the fov square

'''

**from** matplotlib **import** pyplot **as** plt

**import** numpy **as** np

**import** matplotlib.patches **as** mpatches

**import** ControlsParameters **as** cp

**class** **DrawSystem** :

**def** **\_\_init\_\_**(self):

self.init\_flag = **True**

self.airplane = 0

self.rect = 0

self.circle = 0

self.fig, self.ax = plt.subplots() *# PLOT*

plt.axis([0, cp.plot\_width, 0, cp.plot\_height])

plt.plot(cp.x,cp.y,'--')

**def** **draw\_airplane**(self, home):

pts =np.matrix([

[home[0],home[1]],

[home[0]-cp.long\_l,home[1]],

[home[0]-cp.long\_l,home[1]-cp.small\_l],

[home[0]-cp.long\_l-cp.small\_l,home[1]-cp.small\_l],

[home[0]-cp.long\_l-cp.small\_l,home[1]],

[home[0]-cp.long\_l-cp.small\_l-cp.long\_l,home[1]],

[home[0]-cp.long\_l-cp.med\_l+cp.small\_l\*2,home[1]+(cp.small\_l\*1.5)],

[home[0]-cp.long\_l-cp.small\_l,home[1]+(cp.small\_l\*1.5)],

[home[0]-cp.long\_l-cp.small\_l,home[1]+(cp.small\_l\*2)],

[home[0]-cp.long\_l,home[1]+(cp.small\_l\*2)],

[home[0]-cp.long\_l,home[1]+(cp.small\_l\*1.5)],

[home[0]-(cp.small\_l),home[1]+(cp.small\_l\*1.5)],

[home[0],home[1]+(cp.small\_l\*3)]]).T

xy = np.array(pts.T)

**if** self.init\_flag :

self.airplane = mpatches.Polygon(xy, facecolor = 'black', edgecolor = 'black')

self.ax.add\_patch(self.airplane)

**else** :

self.airplane.set\_xy(xy)

**def** **draw\_fov**(self, target) :

rectangle\_start\_point = [target[0] - cp.center\_of\_rect, target[1] - cp.center\_of\_rect]

**if** self.init\_flag :

self.rect = mpatches.Rectangle(rectangle\_start\_point, cp.fov\_len, cp.fov\_len,linewidth=1,edgecolor='r',facecolor='none')

self.circle = mpatches.CirclePolygon(target,radius=15, color='r')

self.ax.add\_patch(self.circle)

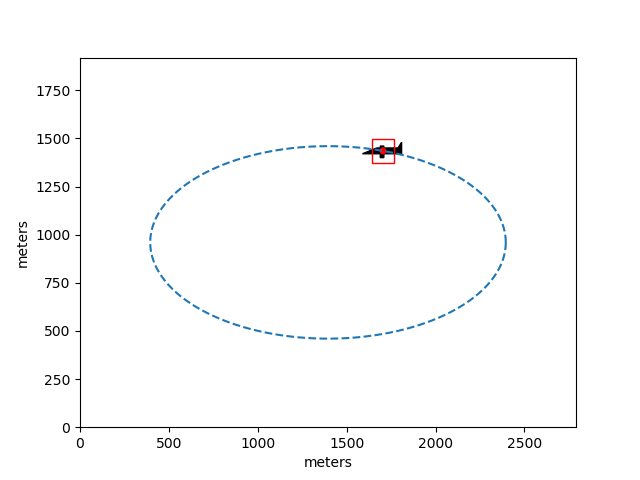
self.ax.add\_patch(self.rect)

**else** :

self.rect.set\_xy(rectangle\_start\_point)

self.circle.\_xy = target

# Simulation Figure

The figure below is an example of the output when we run ControlsMain.py. We are simulating the movement of the plane at a distance of 1/2 a mile away, flying at 64 miles per hour. We then have the red square and red point acting as the field of view for the radar positioning system, which we update at a rate of 2 Hz.