| 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. | |


## 1. Revision History

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

## 2. References

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

3. These files are found in box under the 2019-2020-Capstone-15/Concept Development/Concept Development Packet/Concept_Definitions/Tracking/Controls_Simulations folder.

## 4. 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()
```

## 5. 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
```

## 6. 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
```

## 7. 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.