```
import matplotlib.pyplot as plt
import sys
import matplotlib.patches as mpatches
import numpy as np
sys.path.append('hw_a/')
import param as P

class planarVTOLAnimation:

    def __init__(self):
        self.flagInit = True                  # Used to indicate initialization
        self.fig, self.ax = plt.subplots()    # Initializes a figure and axes object
        self.handle = []                      # Initializes a list object that will
                                              # be used to contain handles to the
                                              # patches and line objects.
        axis_scale = 1.2
        plt.axis([-axis_scale*P.z_max,axis_scale*P.z_max,  # Change the x,y axis limits
                 -(axis_scale-1)*P.h_max, axis_scale*P.h_max])

        # Physical Parameters of Planar VTOL
        self.VTOL = {'w':0.1,      # Width of VTOL center box, m
                     'h':0.1,      # Height of VTOL center box, m
                     'r':0.11}     # Radius of rotors, m

        # Physical Parameters of target
        self.target = {'w':0.1,   # Width of target, m
                       'h':0.1}   # Height of target, m

        # Draw planarVTOL is the main function that will call other draw functions
    def drawPlanarVTOL(self, u):
        # Process inputs to function
        z= u[0]        # Horizontal position of VTOL, m
        h = u[1]       # Vertical position of VTOL, m
        theta = u[2]   # Angle of VTOL, rads

        self.drawRod(z,h,theta)
        self.drawRotors(z,h,theta)
        self.drawCenterBox(z,h,theta)

        if P.includeTarget :
            zv = u[3]  # Horizontal position of target, m
            self.drawTarget(zv)
        # self.ax.axis('equal') # This will cause the image to not distort

        # After each function has been called, initialization is over.
        if self.flagInit == True:
            self.flagInit = False

    def drawCenterBox(self,z,h,theta):
        z_offset = self.VTOL['w']/2.0 # offsets from center
        h_offset = self.VTOL['h']/2.0
        # Vertices of center box
        verts = np.matrix([[-z_offset,-h_offset],
                           [-z_offset,+h_offset],
                           [+z_offset, +h_offset],
                           [+z_offset,-h_offset]])

        # Rotate and translate the vertices
        verts = self.rotate(verts.T,theta).T +[z,h]

        # When the class is initialized, a Rectangle patch object will be
        # created and added to the axes. After initialization, the Rectangle
        # patch object will only be updated.
        if self.flagInit == True:
            # Create the Rectangle patch and append its handle
            # to the handle list
            self.handle.append(mpatches.Polygon(verts,
                closed = True, fc = 'black', ec = 'black',zorder = 1))
            self.ax.add_patch(self.handle[3]) # Add the patch to the axes
        else:
            self.handle[3].set_xy(verts)

    def drawRotors(self,z,h,theta):
        # Left Rotor
        lr_x = z-(P.d+self.VTOL['r'])*np.cos(theta)  # x coordinate
        lr_y = h-(P.d+self.VTOL['r'])*np.sin(theta)  # y coordinate
        lr_xy = (lr_x,lr_y)                          # Center of circle

        # Right Rotor
        rr_x = z+(P.d+self.VTOL['r'])*np.cos(theta)  # x coordinate
        rr_y = h+(P.d+self.VTOL['r'])*np.sin(theta)  # y coordinate
        rr_xy = (rr_x,rr_y)                          # Center of circle

        # When the class is initialized, a CirclePolygon patch object will
        # be created and added to the axes. After initialization, the
        # CirclePolygon patch object will only be updated.
        if self.flagInit == True:
            # Create the CirclePolygon patch and append its handle
            # to the handle list
            self.handle.append(mpatches.CirclePolygon(lr_xy,
                radius = self.VTOL['r'], resolution = 15,
                fc = 'none', ec = 'black'))
            self.ax.add_patch(self.handle[1])  # Add the patch to the axes

            self.handle.append(mpatches.CirclePolygon(rr_xy,
                radius = self.VTOL['r'], resolution = 15,
                fc = 'none', ec = 'black'))
            self.ax.add_patch(self.handle[2])  # Add the patch to the axes
        else:
            self.handle[1]._xy=lr_xy
            self.handle[2]._xy=rr_xy

    def drawRod(self,z,h,theta):
        X = [z-(P.d)*np.cos(theta), z+(P.d)*np.cos(theta)] # X data points
        Y = [h-(P.d)*np.sin(theta), h+(P.d)*np.sin(theta)] # Y data points

        # When the class is initialized, a line object will be
        # created and added to the axes. After initialization, the
        # line object will only be updated.
        if self.flagInit == True:
            # Create the line object and append its handle
            # to the handle list.
            line, =self.ax.plot(X,Y,lw = 2, c = 'gray',zorder = 0)
            self.handle.append(line)
        else:
            self.handle[0].set_xdata(X)               # Update the line
            self.handle[0].set_ydata(Y)

    def drawTarget(self,zv):
        x = zv - self.target['w']/2.0 # x coordinate
        y = 0                         # y coordinate
        xy = (x,y)                    # Bottom left corner of target

        # When the class is initialized, a Rectangle patch object will be
        # created and added to the axes. After initialization, the Rectangle
        # patch object will only be updated.
        if self.flagInit == True:
            # Create the Rectangle patch and append its handle
            # to the handle list
            self.handle.append(mpatches.Rectangle(xy,
                self.target['w'],self.target['h'],
                fc = 'red', ec = 'black'))
            self.ax.add_patch(self.handle[4]) # Add the patch to the axes
        else:
            self.handle[4].set_xy(xy)

    def rotate(self,verts,theta):
        R = np.matrix([[np.cos(theta), -np.sin(theta)],
                       [np.sin(theta),  np.cos(theta)]])
        return R*verts


# Used see the animation.
if __name__ == "__main__":

    simAnimation = planarVTOLAnimation()  # Create Animate object
    z = 2.0                               # Horizontal position of VTOL, m
    h = 2.0                               # Vertical position of VTOL
    theta = 45.0*np.pi/180.0              # Angle of VTOL, rads
    zv = 0.0                              # Horizontal position of target
    simAnimation.drawPlanarVTOL([z,h, theta,zv])  # Draw the Planar VTOL
    plt.show()
```