Munroe Sollog
CSE460 Mobile Robotics
Workshop 1
Github Repository

Elliptical Trajectory

Design a trajectory that follows an elliptial path with a major axis of 4 and a minor axis of 2. Additionally, make the ellipse askew by 30°.
The general form of an ellipse is given by the formula:

```
r = acos(𝚹) + bsin(𝚹)
where:
   a = major axis
   b = minor axis
```

This formula can be converted into independent x and y equations that look like:

```
x = cos(𝚹)(4cos(𝚹) + 2sin(𝚹))
y = sin(𝚹)(4cos(𝚹) + 2sin(𝚹))
```

In order to convert this path into a trajectory we must replace `𝚹` with `t` and take the first order derivative with respect to `t`

```
ux = 2*(cos(2*t-0.5) - 2*sin(2*t-0.5))
uy = 2*(2*cos(2*t) + sin(2*t))
```

I was able to rotate the ellipse by 30° by addint 0.5 to the `t` terms in the `ux` trajectory. I calculated `0.5` by dividing `2π` by `(360/30)`.

Flower

Design a trajectory that follows a 5-petalled course without touching the origin. Additionally we have to add a wind disturbance and attempt to correct for it using a PI-controller

The general formula for a flower is:

```
r = Asin(K𝚹)
where
   A = amplitude of lobes
   K = # of lobes
```

Using this formula and converting it to independent x and y representations we get:

```
x = cos(t)(3sin(5t))
y = sin(t)(3sin(5t))
```

Taking the first derivative provides us with the trajectories:

```
ux = 15*cos(t)*cos(5*t) - sin(t)*(5 + 3*sin(5*t)) + xerr
uy = 15*cos(5*t)*sin(t) + cos(t)*(5 + 3*sin(5*t)) + yerr
```

Finally, including a PI-controller to account for disturbances takes for general form of:

```
u(t) = ɣ(t) + K(γ(t) - x(t)
where
  x(t) = x(t-Δt) + x(t-Δt)Δt
```

Applying this formula to the above trajectory we get:

```
xerr = 5*(cos(t-0.1)*(3*sin(5*(t-0.1))+2) - (x_log[-1][0]))
yerr = 5*(sin(t-0.1)*(3*sin(5*(t-0.1))+2) - (x_log[-1][1]))
```

Helix

Add a z dimension to our simulator and implement a climbing helix. A helix is a basic circle trajectory with a contant z velocity added to it. This results in:

```
x = Ksin(t)
y = Kcos(t)
z = At
```

Converting this path to a trajectory results in:

```
ux = 10*cos(t)
uy = 10*sin(t)
uz = 50
```

Polyline Trajectory

Create a multi-line discontinuous trajectory avoiding obstacles starting and ending at the given points. For this exercise I defined the appropriate waypoints but set my velocity to 0. This allowed for straight-line trajectories around the various rectangles.

```
if (args.path == 'line'):
    # Waypoints
    p1 = [-5.,-7]
    p2 = [10,-7]
    p3 = [9,-4]
    p4 = [4,-2]
    p5 = [3,0]
    p6 = [3,6]
    p7 = [0,6]
    p8 = [0,0]
    p9 = [3,0]
    p10 = [3,10]
    p11 = [9,10]
```

Spline Trajectory

For this exercise, we had to use the same obstacles, but add velocities at waypoints. This allowed me to eliminate some of the waypoints as providing velocities provided curves around the obstacles. I ended up with the following waypoints and velocities

```
if (args.path == 'spline'):
    p1 = [-5.,-7]
    #p2 = [10,-7]
    p3 = [10,-4.5]
    #p4 = [4,-2]
    p5 = [3,0]
    #p6 = [3,6]
    p7 = [2,6]
    #p8 = [0,0]
    p9 = [2,0]
    p10 = [3,10]
    p11 = [9,10]

    v1 = [2,0]
    v3 = [-2,2]
    v5 = [0,1]
    v7 = [-2,0]
    v9 = [3,1]
    v10 = [1,0]
    v11 = [0.001,0]
```

A quick note of confusion around both polyline and spline trajectories. I don't have a clear undertanding of how the `T` variable factors in. Through experimentation I wasn't able to gain a clear understanding how how it affects the outcome.