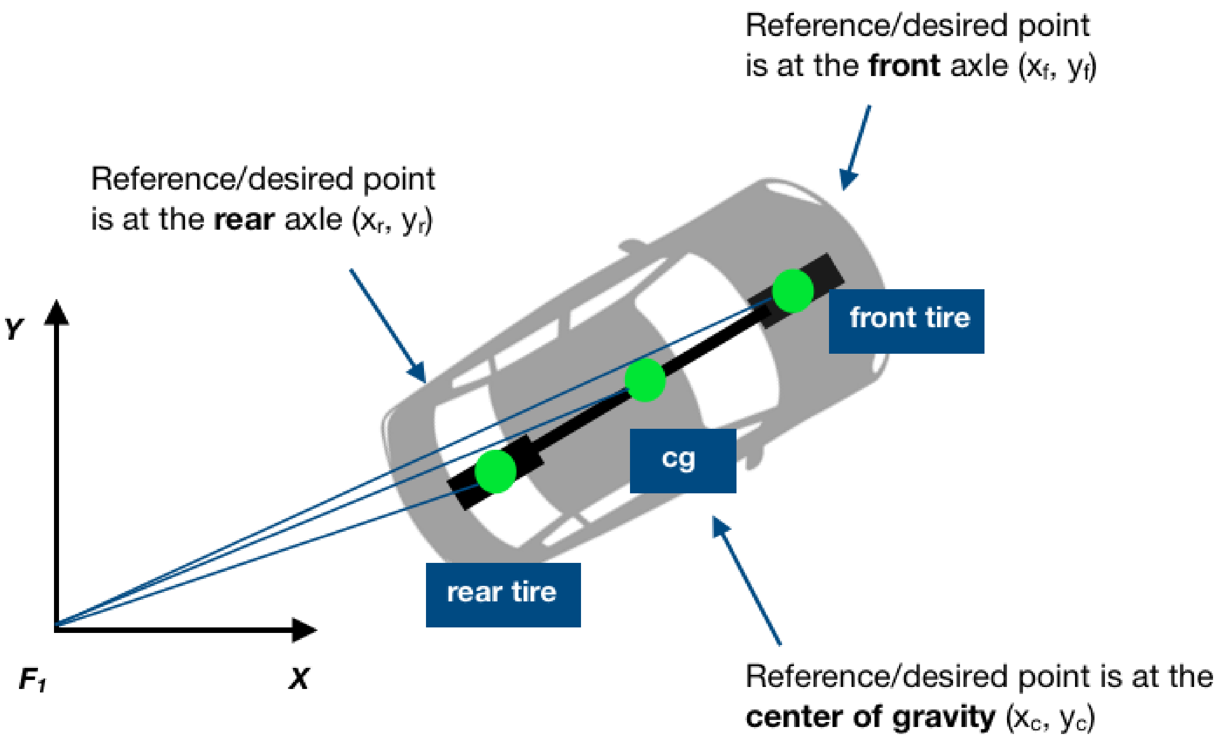
Kinematic Bicycle Model

The well-known kinematic bicycle model has long been used as a suitable control-oriented model for representing vehicles because of its simplicity and adherence to the nonholonomic constraints of a car. Before we derive the model, let's define some variables.

Front Wheel Steering Model

The bicycle model we'll develop is called the **front wheel steering model**, as the front wheel *orientation* can be controlled relative to the heading of the vehicle. We assume the vehicle operates on a 2D plane denoted by the inertial frame F_1.
In the proposed bicycle model:

- the front wheel represents the front right and left wheels of the car
- and the rear wheel represents the rear right and left wheels of the car.



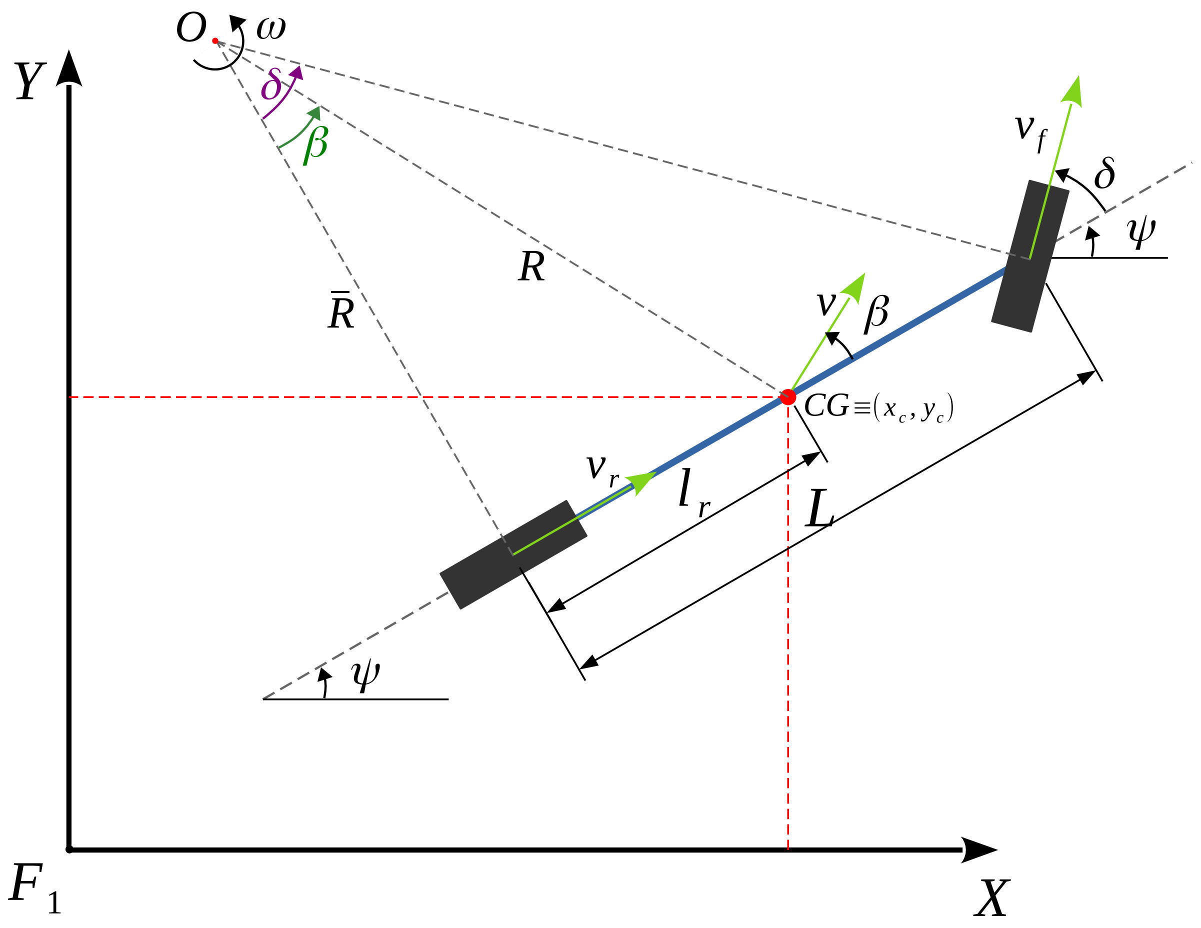
Reference/desired point is at the **front** axle (x_f, y_f)
Reference/desired point is at the **rear** axle (x_r, y_r)
Y
front tire
cg
rear tire
F_1
X
Reference/desired point is at the **center of gravity** (x_c, y_c)

Reference Point

To analyze the kinematics of the bicycle model, we must select a reference point (x, y) on the vehicle which can be placed at:

- the center of the rear axle (x_r, y_r),
- the center of the front axle (x_l, y_l),
- or at the center of gravity CG (x_c, y_c).
  NOTE:
- The selection of the reference point changes the kinematic equations that result,
- which in turn change the controller designs that we'll use.

Center of Mass as a Reference Point & Main States



$O \equiv$ instantaneous center of rotation of the bicycle

$CG \equiv (x_c, y_c)$ location of the rear axle reference point

$\psi \equiv$ heading angle of the bicycle

$L \equiv$ length of the bicycle, measured between the two wheel axes

$l_r \equiv$ rear length of the bicycle, from the CG to the center of the rear axle

$l_f \equiv$ front length of the bicycle, from the CG to the center of the front axle

$\delta \equiv$ steering angle for the front wheel (measured relative to the forward direction of the bicycle)

$v \equiv$ velocity, points in the same direction as each wheel (This is an assumption referred to as the no slip condition, which requires that the wheels cannot move laterally or slip longitudinally)

$\omega \equiv$ rotation rate of the bicycle

$\beta \equiv$ side slip angle of the vehicle

$R \equiv$ radius of rotation of the bicycle

Because of the no slip condition:
$\omega = v/R$

From the similar triangles formed by L and R, and v and δ,
we see that: $\tan(\delta) = L/\bar{R}$

By combining both equations, we can find the relation between the rotation rate of the vehicle ω, and the steering angle δ
$\dot{\psi}(t) = \omega = v/R$

By some trigonometry:

$$\tan(\delta) = \frac{L}{\bar{R}} = \frac{l_r + l_f}{\bar{R}} \implies \bar{R} = \frac{L}{\tan(\delta)}$$

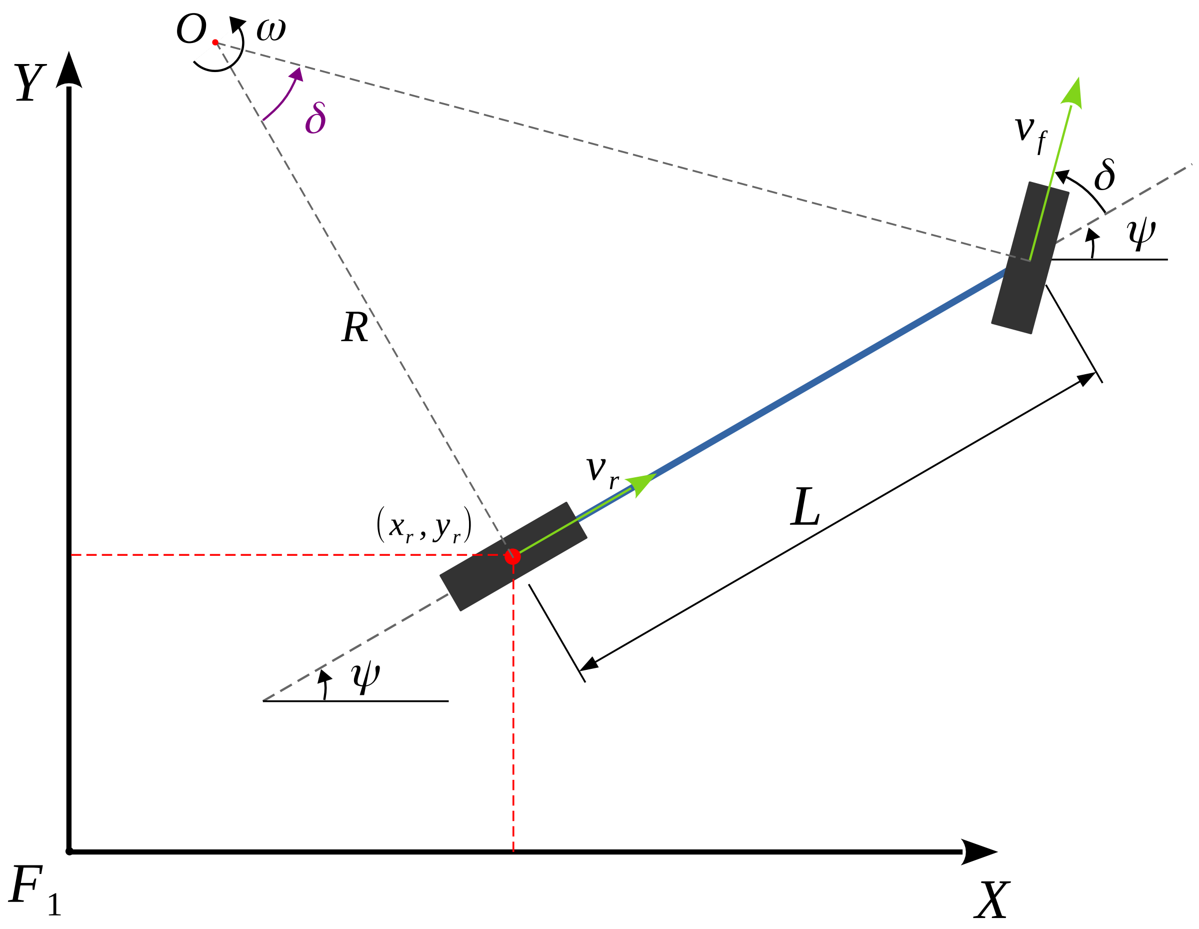
$$\tan(\beta) = \frac{l_r}{\bar{R}} \implies \beta = \tan^{-1}\left(\frac{l_r \cdot \tan(\delta)}{l_r + l_f}\right)$$

$$\cos(\beta) = \frac{\bar{R}}{R} \implies R = \frac{L}{\cos(\beta) \cdot \tan(\delta)}$$

from the above diagram and equations we can formulate our kinematic bicycle model as follows:

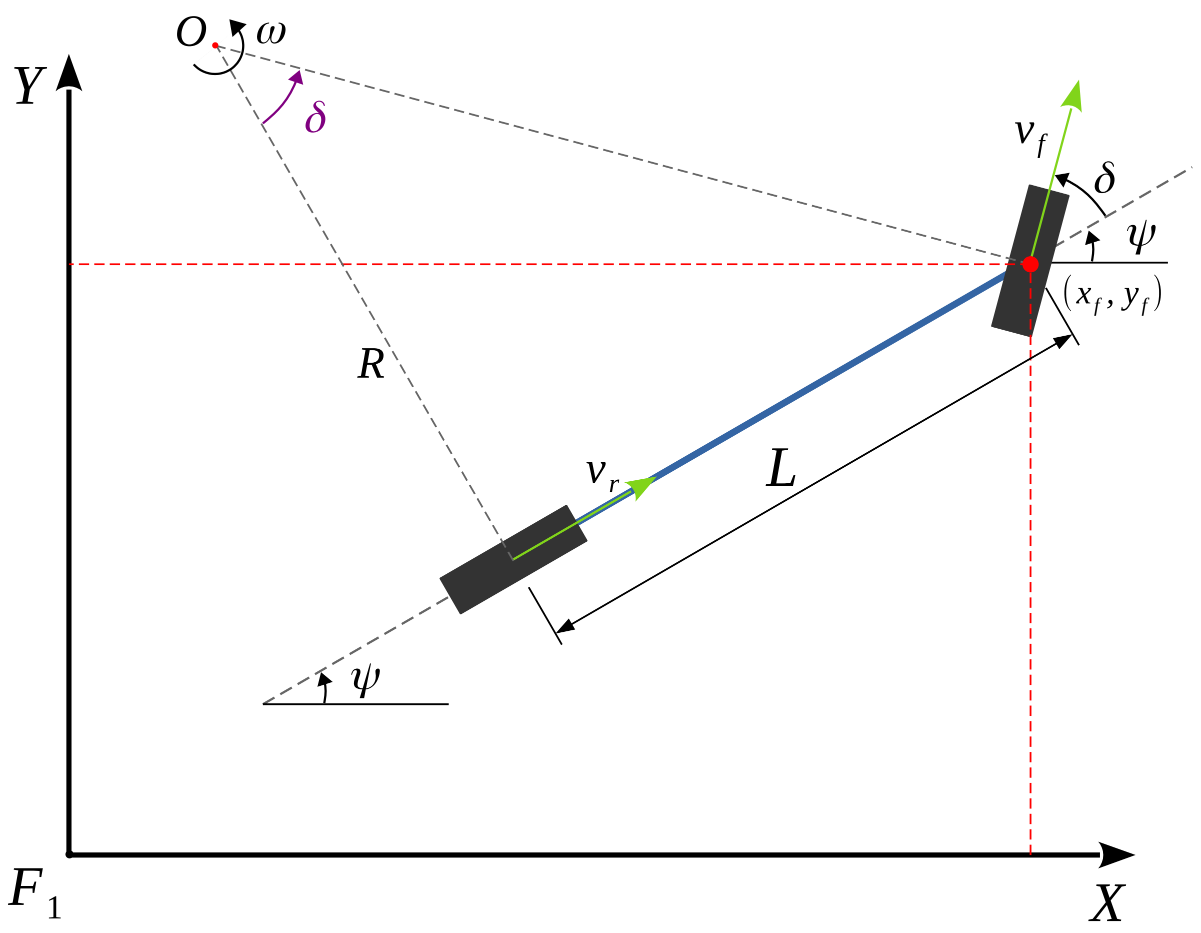
$$\dot{x}_c(t) = v\cos(\psi + \beta)$$
$$\dot{y}_c(t) = v\sin(\psi + \beta)$$
$$\dot{\psi}(t) = \omega = \frac{v}{R}, \quad R = \frac{L}{\cos(\beta) \cdot \tan(\delta)}$$
$$\omega = \dot{\delta} = \dot{\psi}(t) = \frac{v \cdot \cos(\beta) \cdot \tan(\delta)}{L}$$
$$\beta = \tan^{-1}\left(\frac{l_r \cdot \tan(\delta)}{L}\right)$$

Rear Axle Center as a Reference Point



$$\dot{x}_r(t) = v\cos(\psi)$$
$$\dot{y}_r(t) = v\sin(\psi)$$
$$\dot{\psi}(t) = \omega = \frac{v}{R}, \quad R = \frac{L}{\tan(\delta)}$$
$$\dot{\psi}(t) = \frac{v \cdot \tan(\delta)}{L}$$
$$\dot{\delta} = \omega$$

Front Axle Center as a Reference Point



$$
\begin{aligned}
\dot{x}_f(t) &= v\cos(\psi + \delta) \\
\dot{y}_f(t) &= v\sin(\psi + \delta) \\
\dot{\psi}(t) &= \omega = \frac{v}{R}, \quad R = \frac{L}{\sin(\delta)} \\
\dot{\psi}(t) &= \frac{v \cdot \sin(\delta)}{L} \\
\dot{\delta} &= \omega
\end{aligned}
$$

State-space Representation

States: $[x,\ y,\ \psi,\ \delta]^T$
Input: $[v,\ \omega]^T$

In this notebook, you will implement the kinematic bicycle model. The model accepts velocity and steering rate inputs and steps through the bicycle kinematic equations. Once the model is implemented, you will provide a set of inputs to drive the bicycle in a figure 8 trajectory.

The bicycle kinematics are governed by the following set of equations:

$$\dot{x}_c = v\cos(\theta + \beta)$$
$$\dot{y}_c = v\sin(\theta + \beta)$$
$$\dot{\theta} = \omega = \frac{v}{R}, R = \frac{L}{\cos\beta\tan\delta}$$
$$\dot{\theta} = \frac{v\cos\beta\tan\delta}{L}$$
$$\dot{\delta} = \omega$$
$$\beta = \tan^{-1}(\frac{l_r\tan\delta}{L})$$

where the inputs are the bicycle speed v and steering angle rate ω. The input can also directly be the steering angle δ rather than its rate in the simplified case. The Python model will allow us both implementations.

In order to create this model, it's a good idea to make use of Python class objects. This allows us to store the state variables as well as make functions for implementing the bicycle kinematics.

The bicycle begins with zero initial conditions, has a maximum turning rate of 1.22 rad/s, a wheelbase length of 2m, and a length of 1.2m to its center of mass from the rear axle.

A sample time is required for numerical integration when propagating the kinematics through time. This is set to 10 milliseconds. We also have a reset function which sets all the state variables back to 0.

With this sample time, implement the kinematic model using the function $step$ defined in the next cell. The function should take speed + angular rate as inputs and update the state variables. Don't forget about the maximum turn rate on the bicycle!

From these conditions, we initialize the Python class as follows:

In [ ]:
```python
# from notebook_grader import BicycleSolution, grade_bicycle
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import math as math
class Bicycle():
    def __init__(self):
        self.xr = 0
        self.yr = 0
        self.psi = 0
        self.delta = 0
        # self.beta = 0

        self.L = 2
        self.lr = 1.2
        self.w_max = 1.22 # rad/s

        self.sample_time = 0.01

    def reset(self):
        self.xc = 0
        self.yc = 0
        self.psi = 0
        self.delta = 0
        # self.beta = 0

    def step(self, v, w):
        # w = max(-self.w_max,min(w, self.w_max))
        if (abs(w) <= abs(self.w_max)):
            # model equations:
```

```
            xr_dot = v*np.cos(self.psi)
            yr_dot = v*np.sin(self.psi)
            psi_dot = v*np.tan(self.delta)/self.L
            # xc_dot = v*np.cos(self.psi)
            # yc_dot = v*np.sin(self.psi)
            # psi_dot = v*np.tan(self.delta)/self.L
            delta_dot = w
        else:
            print("max w = " + str(self.w_max))
            return
        # update variables
        self.xr += xr_dot*self.sample_time
        self.yr += yr_dot*self.sample_time
        self.delta += delta_dot*self.sample_time
        self.psi += psi_dot*self.sample_time
        # self.beta = np.arctan(self.lr*np.tan(self.delta)/self.L)
    def printAll(self, i, t):
        print(  '\n' + "Iter. #"  + str(i) + " t = " + str(t) + "s"
              + '\n' + "Xr = "    + str(self.xr)
              + '\n' + "Yr = "    + str(self.yr)
              + '\n' + "psi = " + str(self.psi)
              + '\n' + "Delta = " + str(self.delta)
              + '\n' + "Beta = "  + str(self.beta))
```

With the model setup, we can now start giving bicycle inputs and producing trajectories.

Suppose we want the model to travel a circle of radius 10 m in 20 seconds. Using the relationship between the radius of curvature and the steering angle, the desired steering angle can be computed.

$$\begin{aligned}\tan\delta &= \frac{L}{r} \\ \delta &= \tan^{-1}(\frac{L}{r}) \\ &= \tan^{-1}(\frac{2}{10}) \\ &= 0.1974\end{aligned}$$

If the steering angle is directly set to 0.1974 using a simplied bicycled model, then the bicycle will travel in a circle without requiring any additional steering input.

The desired speed can be computed from the circumference of the circle:

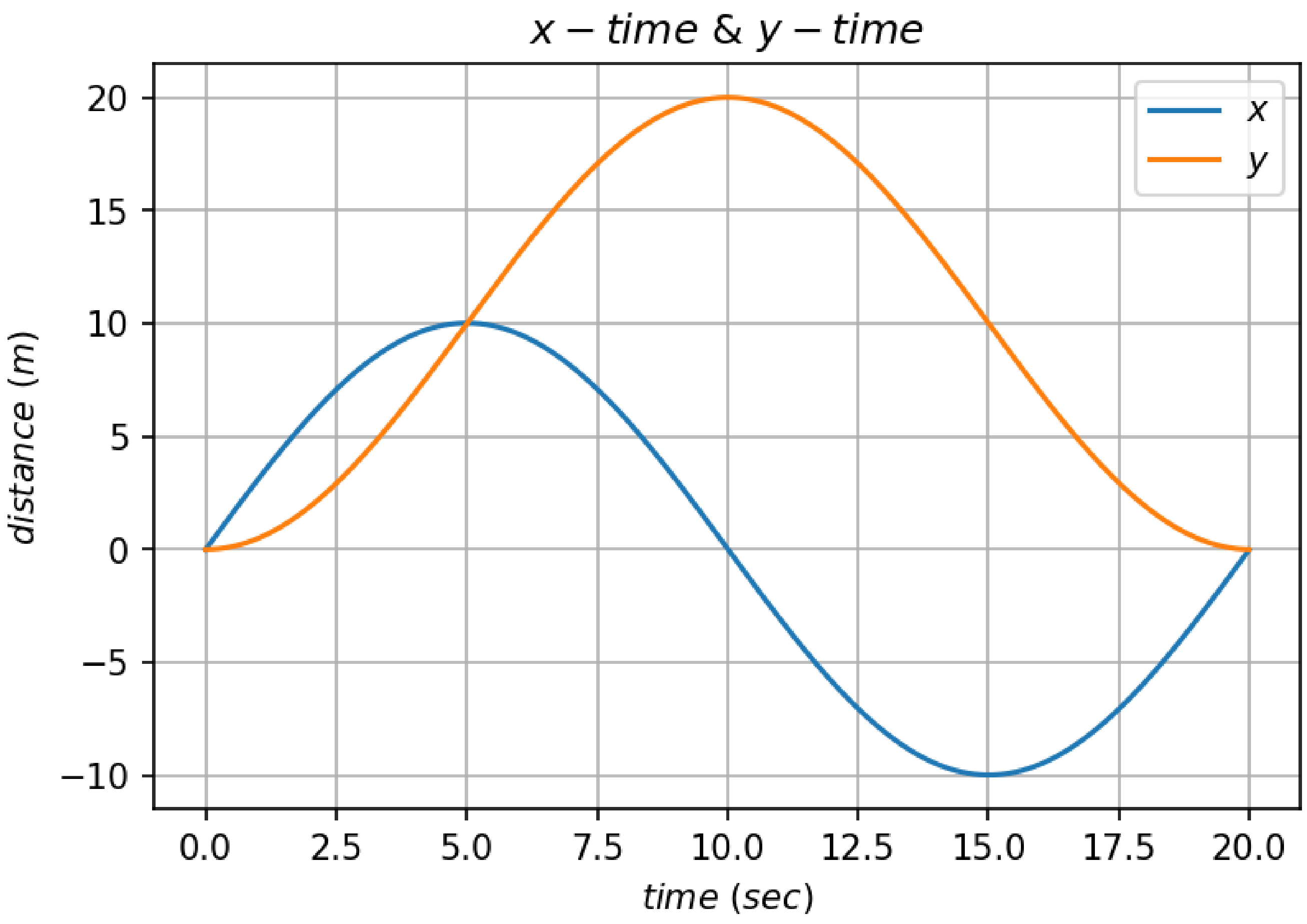
$$\begin{aligned}v &= \frac{d}{t} \\ &= \frac{2\pi 10}{20} \\ &= \pi\end{aligned}$$

We can now implement this in a loop to step through the model equations. We will also run our bicycle model solution along with your model to show you the expected trajectory. This will help you verify the correctness of your model.

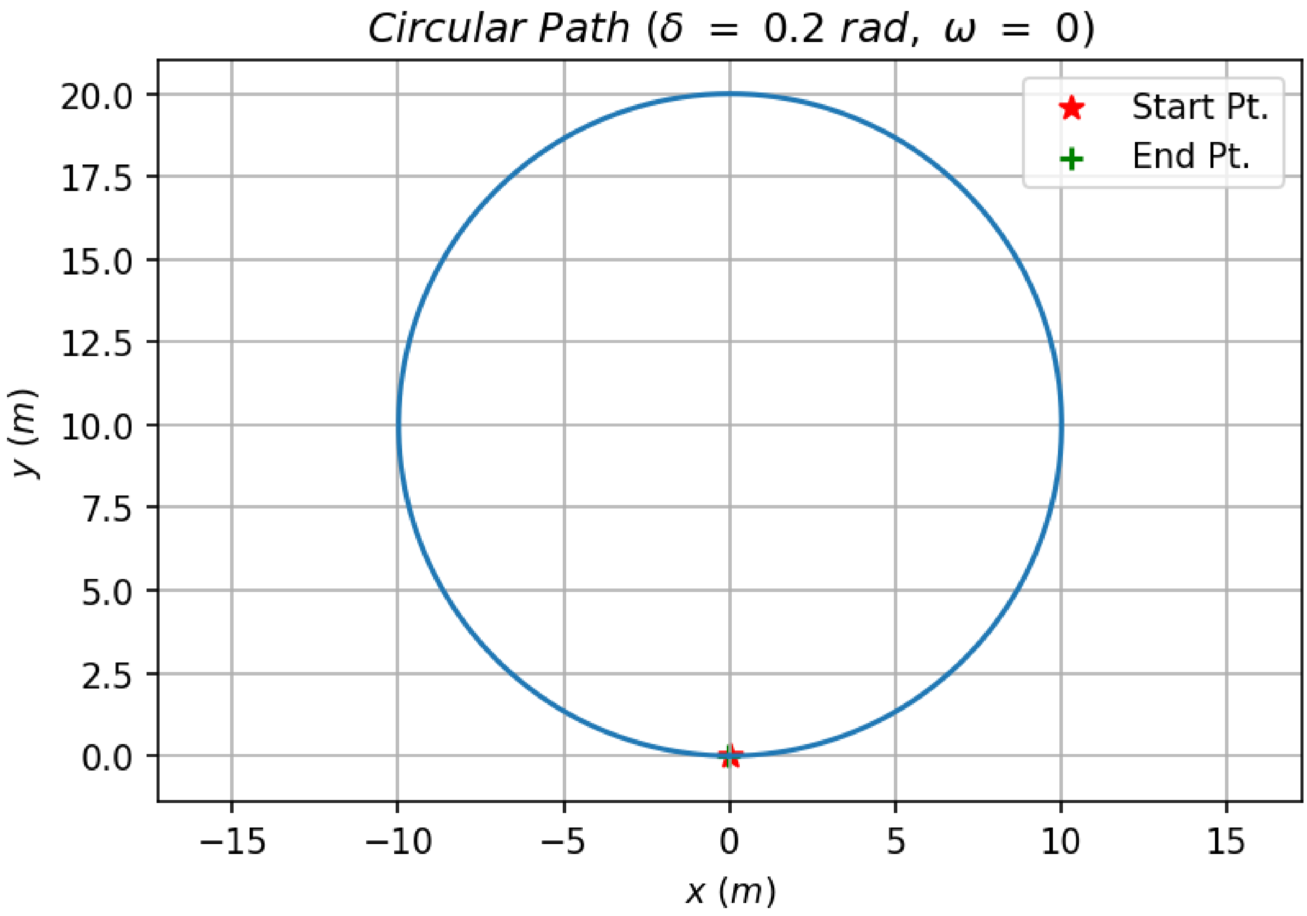
In [ ]:

```
sample_time = 0.01
time_end = 20
model_circl1 = Bicycle()

# set delta directly
model_circl1.delta = np.arctan(2/10)

t_data = np.arange(0,time_end,sample_time)
x_data = np.zeros_like(t_data)
```

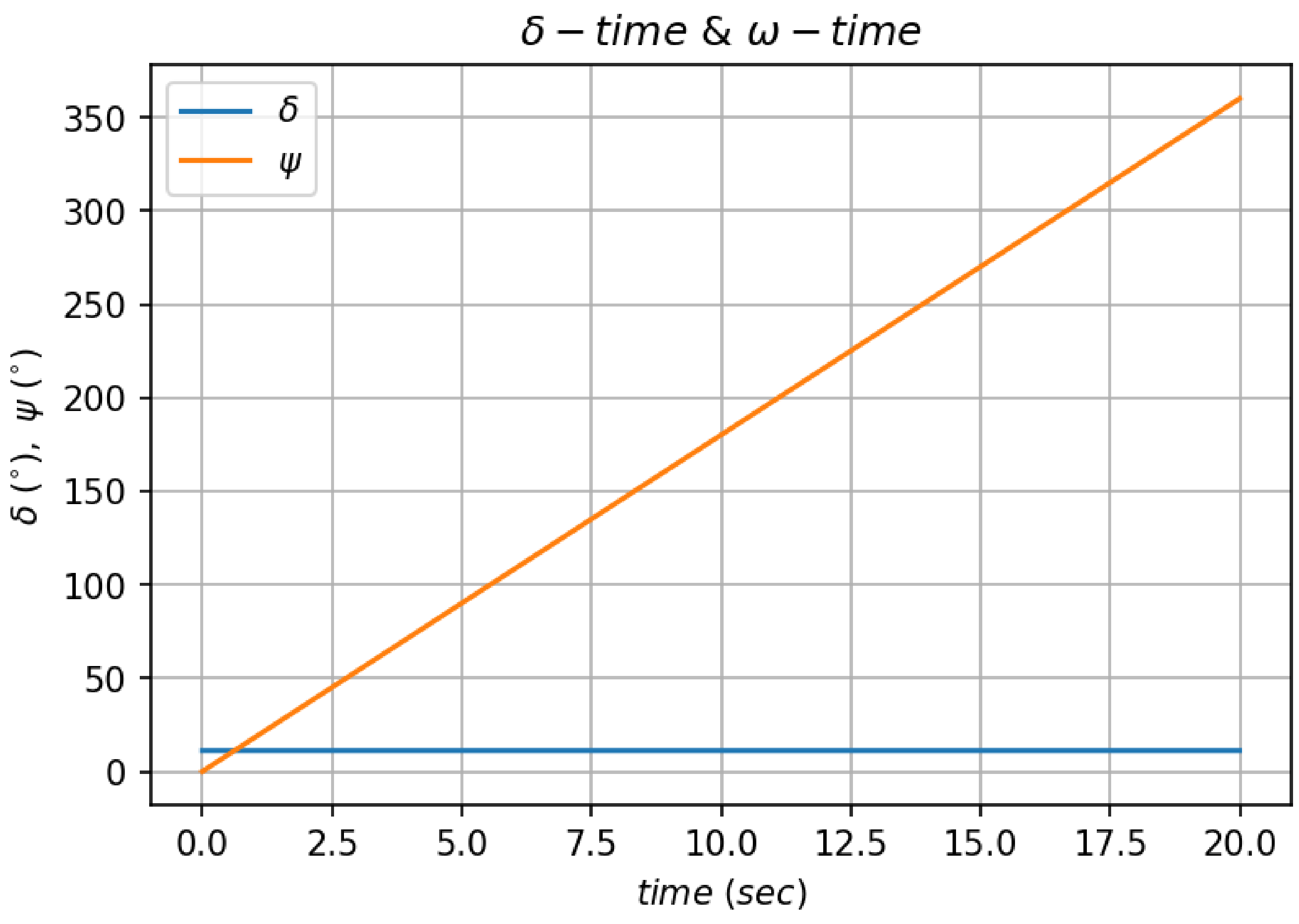
```
y_data = np.zeros_like(t_data)
psi_data = np.zeros_like(t_data)
delta_data = np.zeros_like(t_data)

n = t_data.shape[0]
for i in range(n):
    x_data[i] = model_circl1.xr
    y_data[i] = model_circl1.yr
    model_circl1.step(np.pi, 0)
    psi_data[i] = model_circl1.psi
    delta_data[i] = model_circl1.delta

plt.plot(t_data[0:n], x_data[0:n],label=r'$x$')
plt.plot(t_data[0:n], y_data[0:n],label=r'$y$')
plt.title(r'$x-time\ &\ y-time$')
plt.xlabel(r'$time\ (sec)$')
plt.ylabel(r'$distance\ (m)$')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

plt.plot(x_data[0:n], y_data[0:n])
plt.scatter(x_data[0], y_data[0],label='Start Pt.' , s=50, c='r', marker='*')
plt.scatter(x_data[-1], y_data[-1],label='End Pt.' , s=50, c='g', marker='+')
plt.title(r'$Circular\ Path\ (\delta\ =\ 0.2\ rad,\ \omega\ =\ 0)$')
plt.xlabel(r'$x\ (m)$')
plt.ylabel(r'$y\ (m)$')
plt.axis('equal')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

plt.plot(t_data, delta_data*180.0/np.pi,label=r'$\delta$')
plt.plot(t_data, psi_data*180.0/np.pi,label=r'$\psi$')
plt.title(r'$\delta-time\ &\ \omega-time$')
plt.xlabel(r'$time\ (sec)$')
plt.ylabel(r'$\delta\ (^{\circ}),\ \psi\ (^{\circ})$')
plt.grid()
plt.legend()
# plt.gcf().set_size_inches(18,4)
plt.gcf().set_dpi(150)
plt.show()
```



$x - time$ & $y - time$
x
y
distance (m)
time (sec)



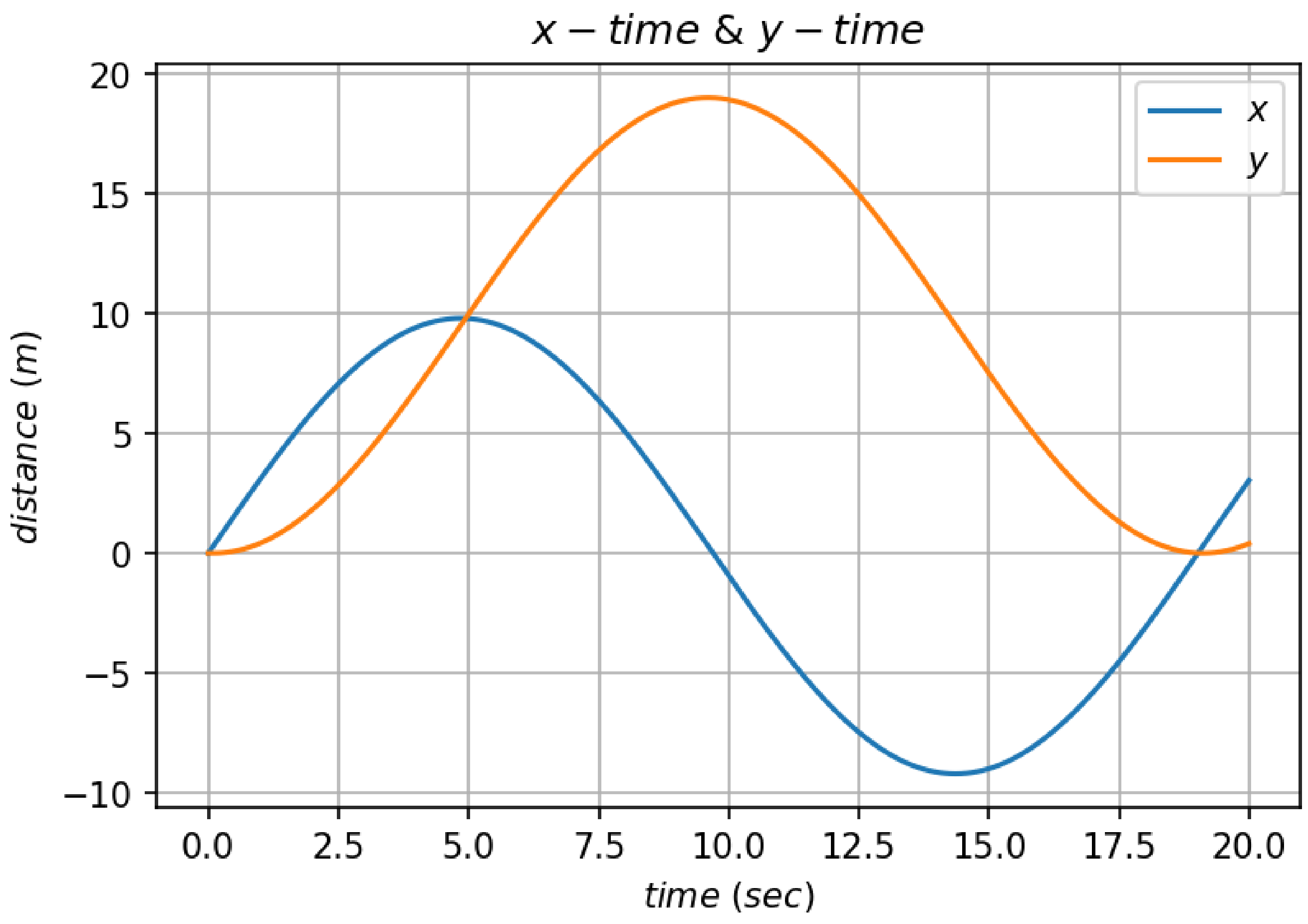
Circular Path (δ = 0.2 rad, ω = 0)
Start Pt.
End Pt.
y (m)
x (m)

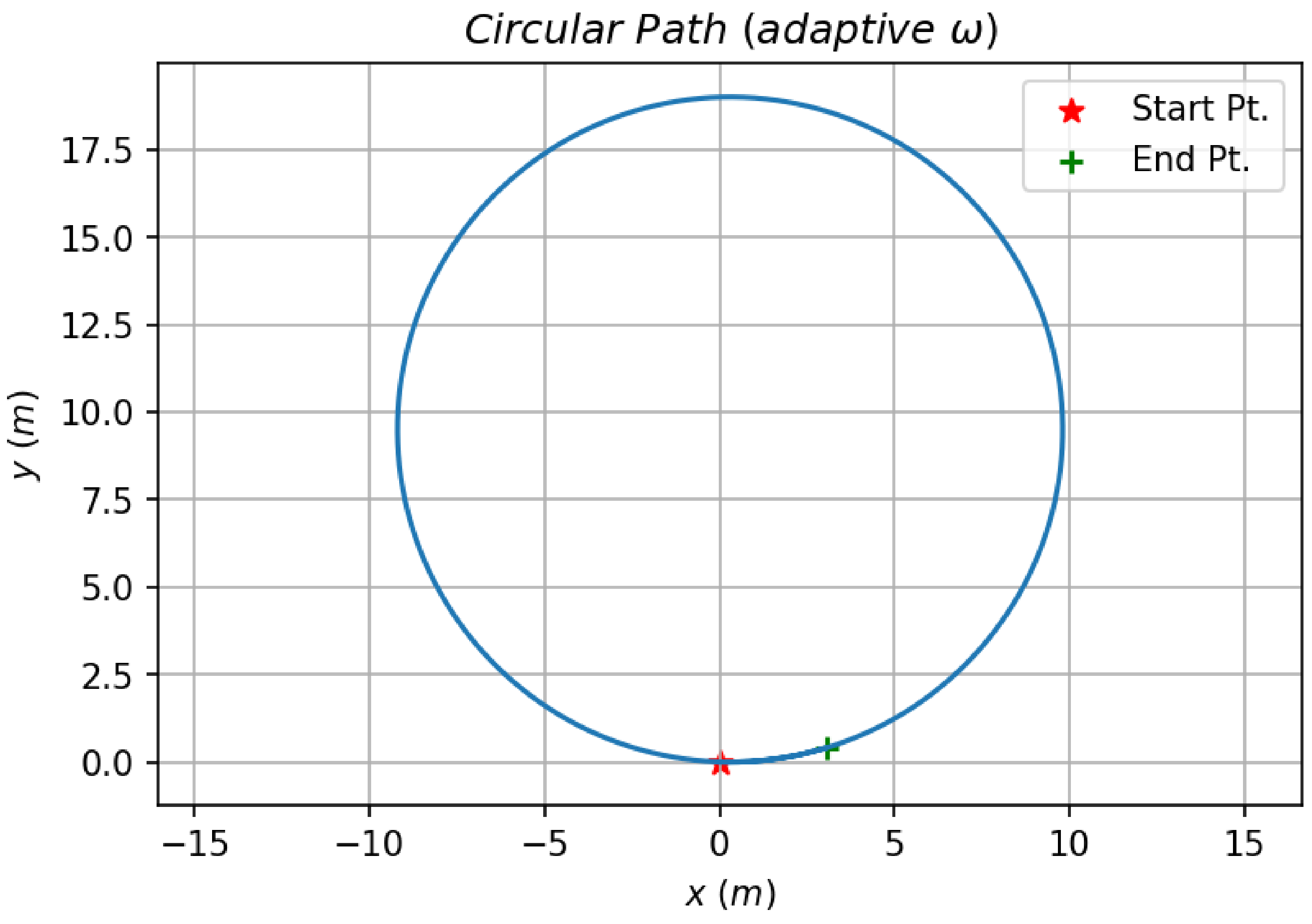


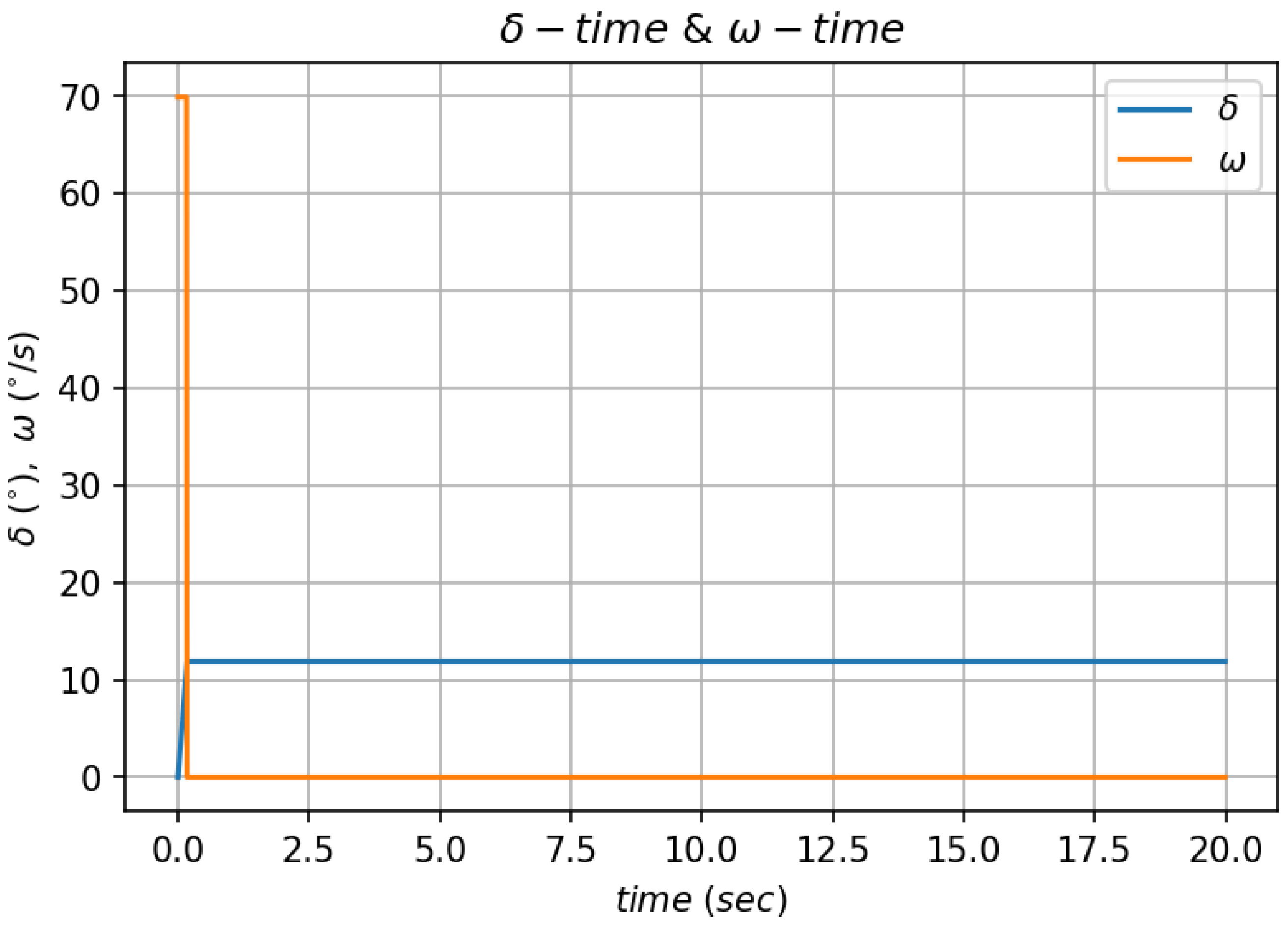
The plot above shows the desired circle of 10m radius. The path is slightly offset which is caused by the sideslip effects due to β. By forcing $\beta = 0$ through uncommenting the last line in the loop, you can see that the offset disappears and the circle becomes centered at (0,10).

However, in practice the steering angle cannot be directly set and must be changed through angular rate inputs ω. The cell below corrects for this and sets angular rate inputs to generate the same circle trajectory. The speed v is still maintained at π m/s.

In [ ]:
```
sample_time = 0.01
time_end = 20
model_circl2 = Bicycle()

t_data = np.arange(0,time_end,sample_time)
x_data = np.zeros_like(t_data)
y_data = np.zeros_like(t_data)
w_data = np.zeros_like(t_data)
delta_data = np.zeros_like(t_data)

n = t_data.shape[0]
for i in range(n):
    x_data[i] = model_circl2.xr
    y_data[i] = model_circl2.yr
    delta_data[i] = model_circl2.delta

    if model_circl2.delta < np.arctan(2/10):
        model_circl2.step(np.pi, model_circl2.w_max)
        w_data[i] = model_circl2.w_max
    else:
        model_circl2.step(np.pi, 0)
        w_data[i] = 0

plt.plot(t_data[0:n], x_data[0:n],label=r'$x$')
plt.plot(t_data[0:n], y_data[0:n],label=r'$y$')
plt.title(r'$x-time\ &\ y-time$')
plt.xlabel(r'$time\ (sec)$')
plt.ylabel(r'$distance\ (m)$')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

plt.plot(x_data[0:n], y_data[0:n])
plt.scatter(x_data[0], y_data[0],label='Start Pt.' , s=50, c='r', marker='*')
plt.scatter(x_data[-1], y_data[-1],label='End Pt.' , s=50, c='g', marker='+')
plt.title(r'$Circular\ Path\ (adaptive\ \omega)$')
plt.xlabel(r'$x\ (m)$')
plt.ylabel(r'$y\ (m)$')
plt.axis('equal')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

plt.plot(t_data, delta_data*180/np.pi, label=r'$\delta$')
plt.plot(t_data, w_data*180/np.pi, label=r'$\omega$')
plt.title(r'$\delta-time\ &\ \omega-time$')
plt.xlabel(r'$time\ (sec)$')
plt.ylabel(r'$\delta\ (^{\circ}),\ \omega\ (^{\circ}/s)$')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()
```







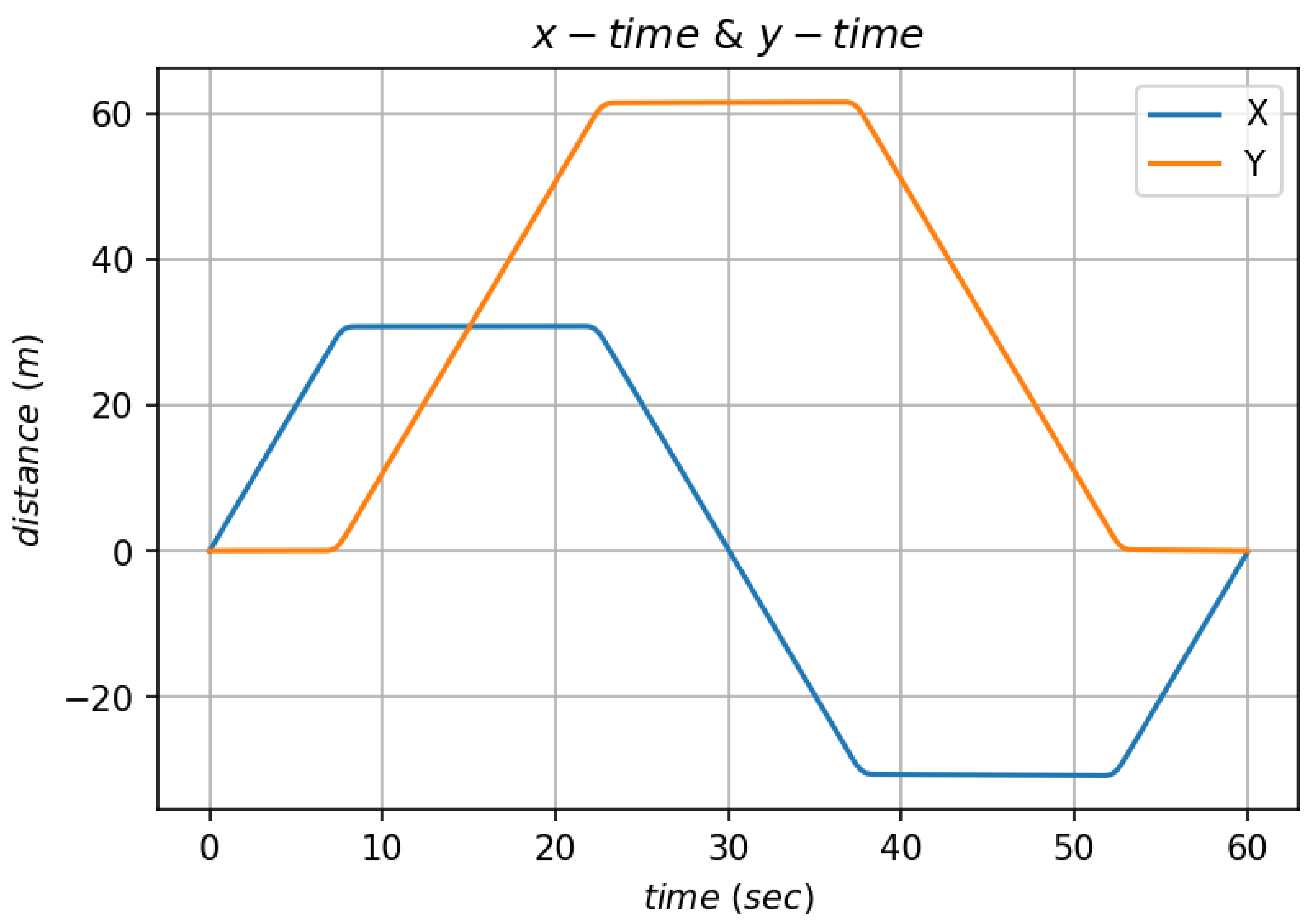
+Here are some other example trajectories: a square path, a spiral path, and a wave path. Uncomment each section to view.

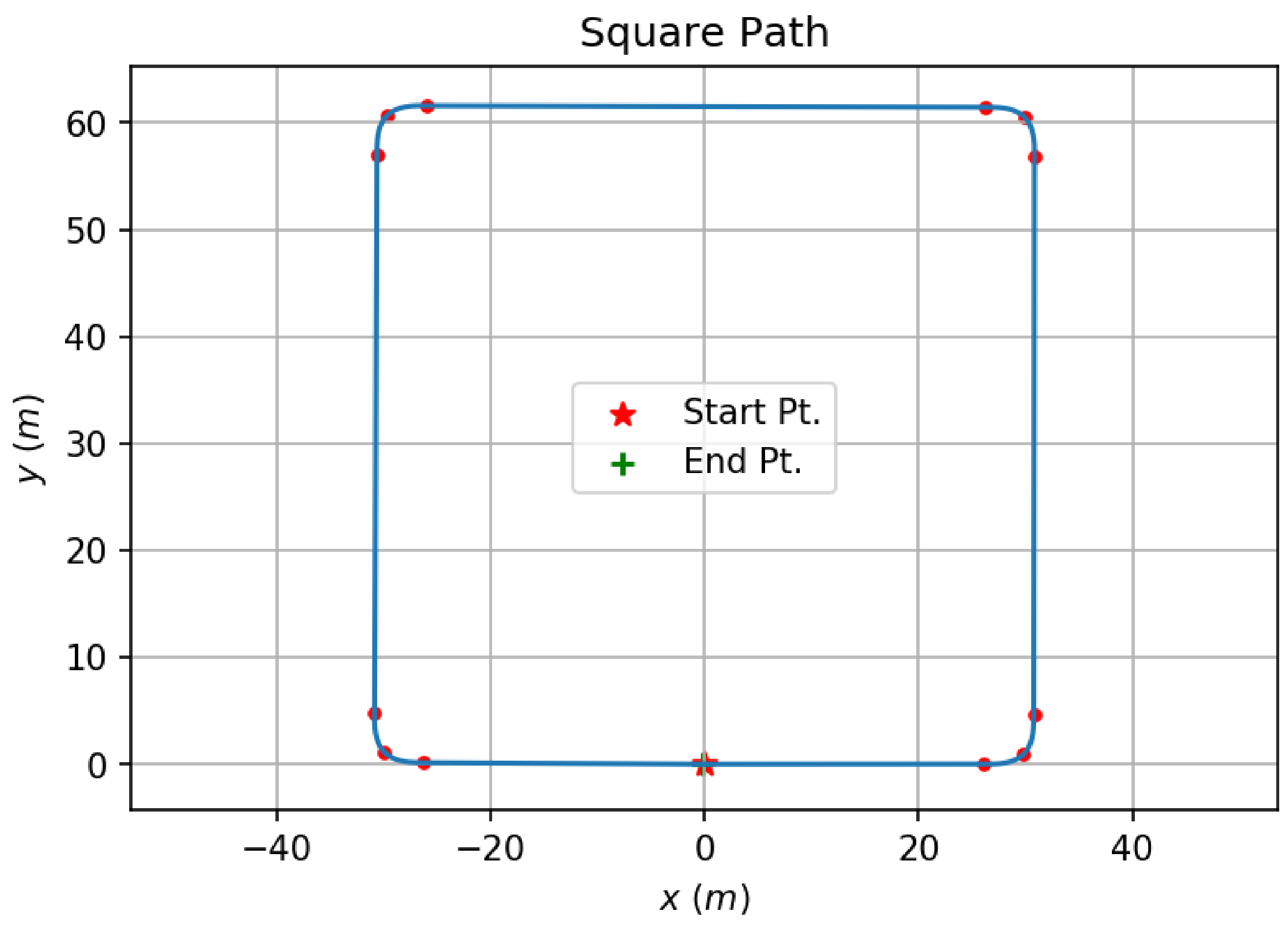
Square Path:

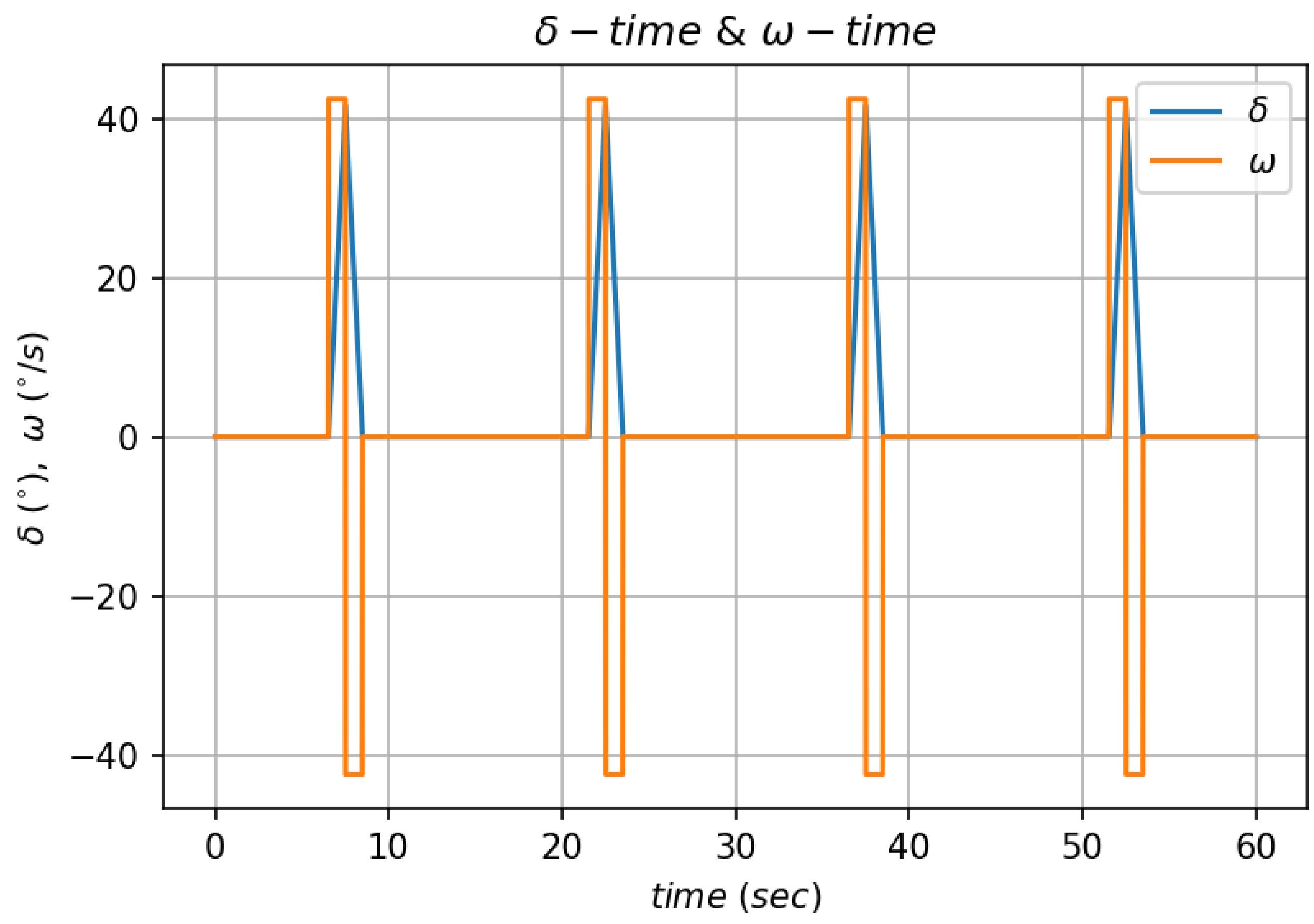
In [ ]:
```
sample_time = 0.01
time_end = 60

model_square = Bicycle()

t_data = np.arange(0,time_end,sample_time)
x_data = np.zeros_like(t_data)
y_data = np.zeros_like(t_data)
delta_data = np.zeros_like(t_data)

# maintain velocity at 4 m/s
v_data = np.zeros_like(t_data)
v_data[:] = 4

w_data = np.zeros_like(t_data)

v = v_data[0]
w = 0.8
r = v/w
print("r = " + str(r))

l = math.pi/2 * r
d = 60.0 - l
print("l = " + str(l))
print("d = " + str(d))

l_steps = round(1500.0 * (l / 60.0))
d_steps = round(1500.0 * (d / 60.0))
print("l_steps = " + str(l_steps))
print("d_steps/2 = " + str(d_steps/2))

p = np.zeros(12)
text = ['p01', 'p02', 'p03', 'p04', 'p05', 'p06',
        'p07', 'p08', 'p09', 'p10', 'p11', 'p12']

p[0] = int(d_steps/2)
p[1] = int(p[0] + l_steps/2)
p[2] = int(p[0] + (l_steps/2) * 2)
p[3] = int(p[2] + d_steps)
p[4] = int(p[3] + l_steps/2)
p[5] = int(p[3] + (l_steps/2) * 2)
p[6] = int(p[5] + d_steps)
p[7] = int(p[6] + l_steps/2)
p[8] = int(p[6] + (l_steps/2) * 2)
p[9] = int(p[8] + d_steps)
p[10] = int(p[9] + l_steps/2)
p[11] = int(p[9] + (l_steps/2) * 2)

for i, txt in enumerate(text):
    print(txt + ' @ ' + str(p[i]/1000) + ' sec  \t= Step #' + str(int(p[i])))


# Bottom right corner
w_data[int(p[0]):int(p[1])] = 0.741 # calculated at 0.8 rad/s
w_data[int(p[1]):int(p[2])] = -0.741

# Top right corner
w_data[int(p[3]):int(p[4])] = 0.741
w_data[int(p[4]):int(p[5])] = -0.741

# Top left corner
w_data[int(p[6]):int(p[7])] = 0.741
w_data[int(p[7]):int(p[8])] = -0.741

# Bottom left corner
w_data[int(p[9]):int(p[10])] = 0.741
w_data[int(p[10]):int(p[11])] = -0.741
```

```
n = 6000
for i in range(n):
    x_data[i] = model_square.xr
    y_data[i] = model_square.yr
    delta_data[i] = model_square.delta
    model_square.step(v_data[i], w_data[i])

plt.plot(t_data[0:n], x_data[0:n],label='X')
plt.plot(t_data[0:n], y_data[0:n],label='Y')
plt.title(r'$x-time\ &\ y-time$')
plt.xlabel(r'$time\ (sec)$')
plt.ylabel(r'$distance\ (m)$')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

plt.plot(x_data[0:n], y_data[0:n])
plt.scatter(x_data[0], y_data[0],label='Start Pt.' , s=50, c='r', marker='*')
plt.scatter(x_data[-1], y_data[-1],label='End Pt.' , s=50, c='g', marker='+')
for i in range(len(p)):
    plt.scatter(x_data[int(p[i])], y_data[int(p[i])], s=10, c='r')
plt.title('Square Path')
plt.xlabel(r'$x\ (m)$')
plt.ylabel(r'$y\ (m)$')
plt.axis('equal')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

plt.plot(t_data, delta_data*180/math.pi,label=r'$\delta$')
plt.plot(t_data, w_data*180/math.pi,label=r'$\omega$')
plt.title(r'$\delta-time\ &\ \omega-time$')
plt.xlabel(r'$time\ (sec)$')
plt.ylabel(r'$\delta\ (^{\circ}),\ \omega\ (^{\circ}/s)$')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

print('Max delta = ' + str(math.degrees(max(delta_data))))
```

```
r = 5.0
l = 7.853981633974483
d = 52.146018366025515
l_steps = 196
d_steps/2 = 652.0
p01 @ 0.652 sec         = Step #652
p02 @ 0.75 sec          = Step #750
p03 @ 0.848 sec         = Step #848
p04 @ 2.152 sec         = Step #2152
p05 @ 2.25 sec          = Step #2250
p06 @ 2.348 sec         = Step #2348
p07 @ 3.652 sec         = Step #3652
p08 @ 3.75 sec          = Step #3750
p09 @ 3.848 sec         = Step #3848
p10 @ 5.152 sec         = Step #5152
p11 @ 5.25 sec          = Step #5250
p12 @ 5.348 sec         = Step #5348
```

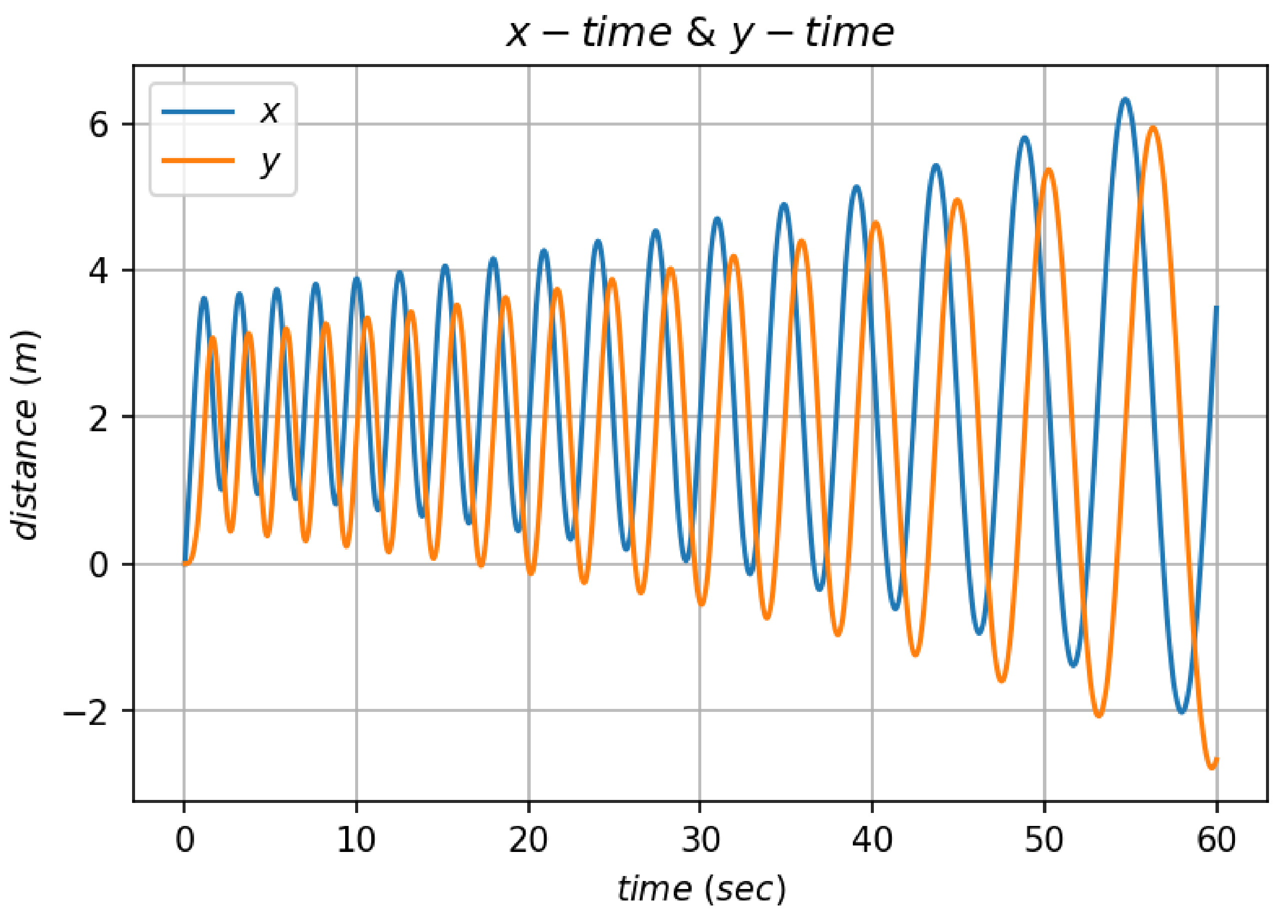

$x - time$ & $y - time$
X
Y
distance (m)
time (sec)
60
40
20
0
−20
0
10
20
30
40
50
60

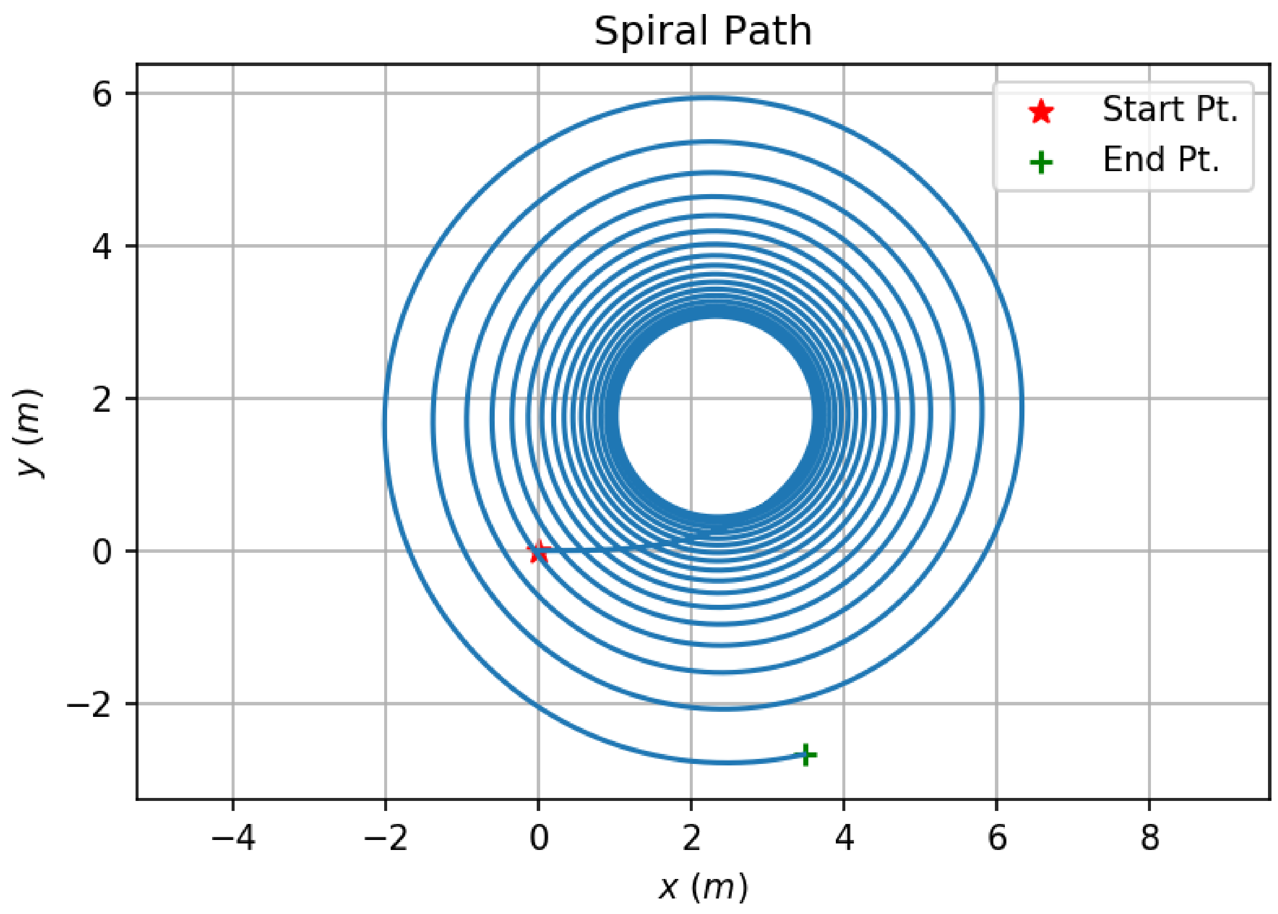

Square Path
Start Pt.
End Pt.
y (m)
x (m)
60
50
40
30
20
10
0
−40
−20
0
20
40

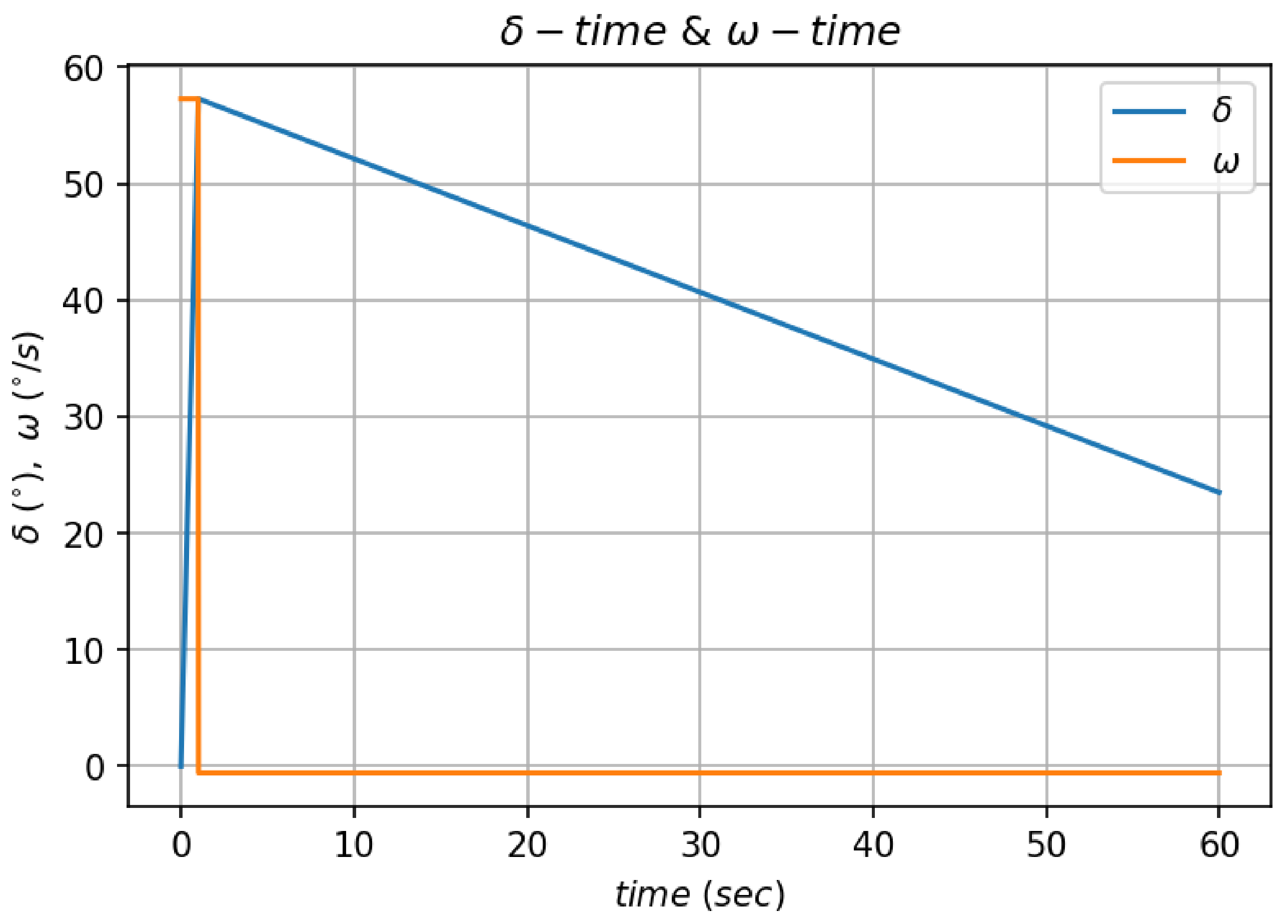


```
Max delta = 41.60704916681022
```

In [ ]:
```
sample_time = 0.01
time_end = 60
model_spiral = Bicycle()

t_data = np.arange(0,time_end,sample_time)
x_data = np.zeros_like(t_data)
y_data = np.zeros_like(t_data)
delta_data = np.zeros_like(t_data)

# maintain velocity at 4 m/s
v_data = np.zeros_like(t_data)
v_data[:] = 4

w_data = np.zeros_like(t_data)

# ==================================
#  Spiral Path: high positive w, then small negative w
# ==================================
w_data[:] = -1/100
w_data[0:100] = 1

n = 6000
for i in range(n):
    x_data[i] = model_spiral.xr
    y_data[i] = model_spiral.yr
    delta_data[i] = model_spiral.delta
    model_spiral.step(v_data[i], w_data[i])

plt.plot(t_data[0:n], x_data[0:n],label=r'$x$')
plt.plot(t_data[0:n], y_data[0:n],label=r'$y$')
plt.title(r'$x-time\ &\ y-time$')
plt.xlabel(r'$time\ (sec)$')
plt.ylabel(r'$distance\ (m)$')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

plt.plot(x_data[0:n], y_data[0:n])
plt.scatter(x_data[0], y_data[0],label='Start Pt.' , s=50, c='r', marker='*')
plt.scatter(x_data[-1], y_data[-1],label='End Pt.' , s=50, c='g', marker='+')
plt.title('Spiral Path')
plt.xlabel(r'$x\ (m)$')
plt.ylabel(r'$y\ (m)$')
plt.axis('equal')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

plt.plot(t_data, delta_data*180/math.pi,label=r'$\delta$')
plt.plot(t_data, w_data*180/math.pi,label=r'$\omega$')
plt.title(r'$\delta-time\ &\ \omega-time$')
plt.xlabel(r'$time\ (sec)$')
plt.ylabel(r'$\delta\ (^{\circ}),\ \omega\ (^{\circ}/s)$')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

print('Max delta = ' + str(math.degrees(max(delta_data))))
```

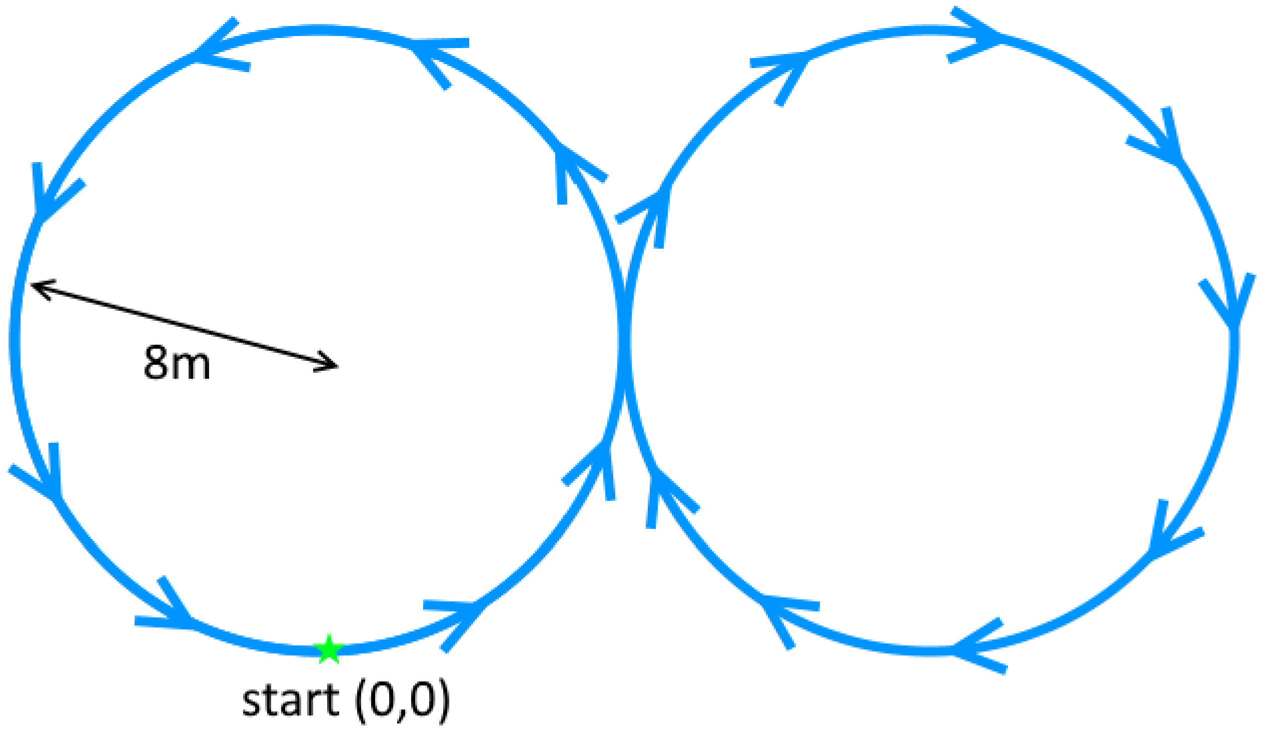






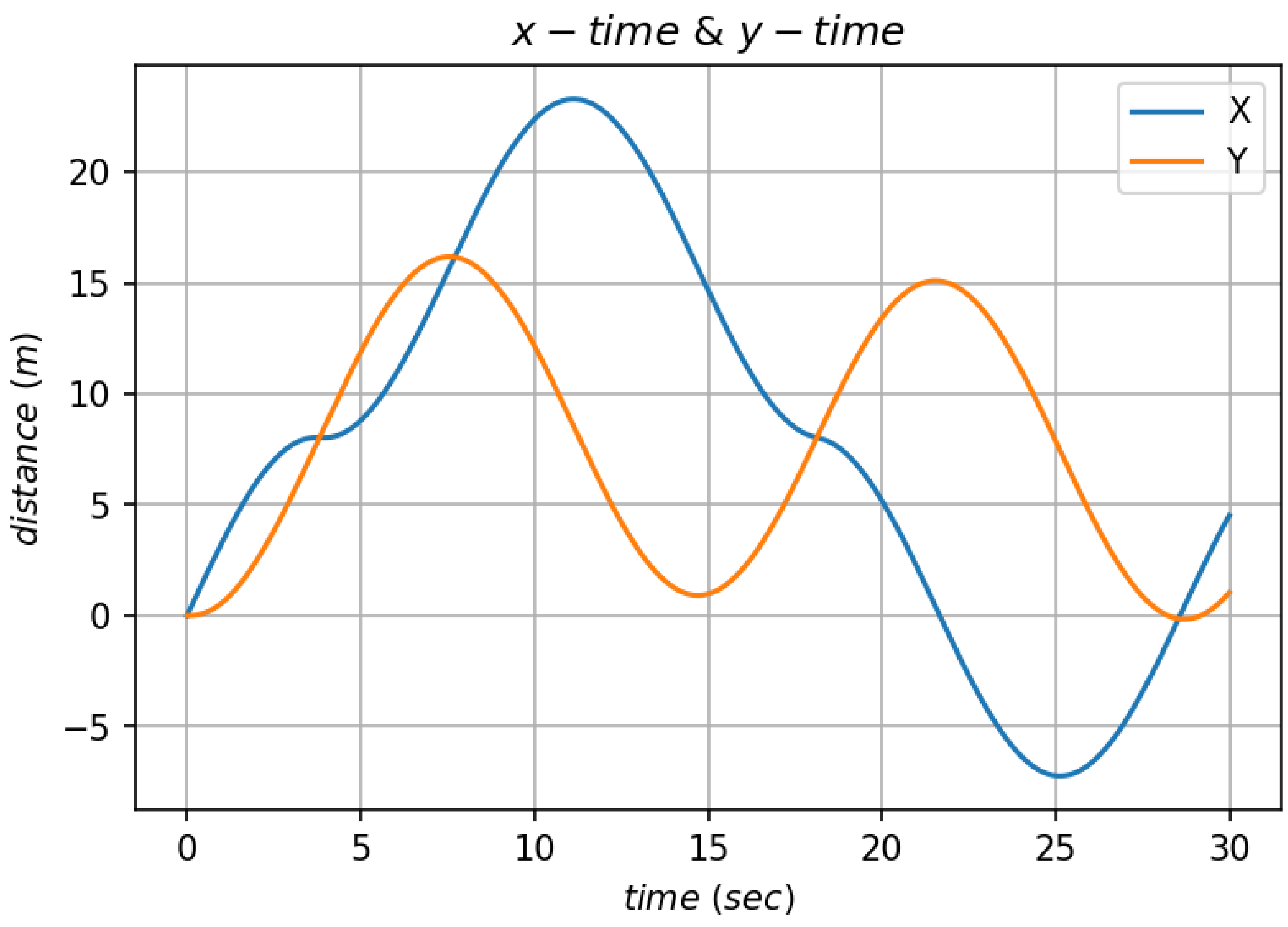
```
Max delta = 57.29577951308236
```

We would now like the bicycle to travel a figure eight trajectory. Both circles in the figure eight have a radius of 8m and the path should complete in 30 seconds. The path begins at the bottom of the left circle and is shown in the figure below:



Determine the speed and steering rate inputs required to produce such trajectory and implement in the cell below. Make sure to also save your inputs into the arrays v_data and w_data, these will be used to grade your solution. The cell below also plots the trajectory generated by your own model.

In [ ]:
```python
sample_time = 0.01 # sec
time_end = 30.0 # sec
model_8 = Bicycle()

t_data = np.arange(0,time_end,sample_time)
x_data = np.zeros_like(t_data)
y_data = np.zeros_like(t_data)
v_data = np.zeros_like(t_data)
w_data = np.zeros_like(t_data)
delta_data = np.zeros_like(t_data)

r = 8.0 # m
v = 4.0 * np.pi * r / time_end # v = d/t = 2(2*pi*r) / 30.0
v_data[:] = v
delta = np.arctan(model_8.L / 8.0)
n = 3000
p1 = 360
p2 = 1790
for i in range(n):
    x_data[i] = model_8.xr
    y_data[i] = model_8.yr
    delta_data[i] = model_8.delta

    if i < p1:
        if model_8.delta < delta:
            w_data[i] = model_8.w_max

    elif i > p1 and i < p2:
        if model_8.delta > -delta:
            w_data[i] = -model_8.w_max
    else:#if i > 1875 and i < n:
        if model_8.delta < delta:
            w_data[i] = model_8.w_max

    model_8.step(v_data[i], w_data[i])

plt.plot(t_data[0:n], x_data[0:n],label='X')
plt.plot(t_data[0:n], y_data[0:n],label='Y')
plt.title(r'$x-time\ &\ y-time$')
plt.xlabel(r'$time\ (sec)$')
plt.ylabel(r'$distance\ (m)$')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

plt.plot(x_data[0:n], y_data[0:n])
plt.scatter(x_data[0], y_data[0],label='Start Pt.' , s=50, c='r', marker='*')
plt.scatter(x_data[-1], y_data[-1],label='End Pt.' , s=50, c='g', marker='+')
plt.title('Figure 8 Path')
plt.xlabel(r'$x\ (m)$')
plt.ylabel(r'$y\ (m)$')
plt.axis('equal')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()

plt.plot(t_data, delta_data*180/math.pi, label=r'$\delta$')
plt.plot(t_data, w_data*180/math.pi, label=r'$\omega$')
plt.title(r'$\delta-time\ &\ \omega-time$')
plt.xlabel(r'$time\ (sec)$')
plt.ylabel(r'$\delta\ (^{\circ}),\ \omega\ (^{\circ}/s)$')
plt.grid()
plt.legend()
plt.gcf().set_dpi(150)
plt.show()
```



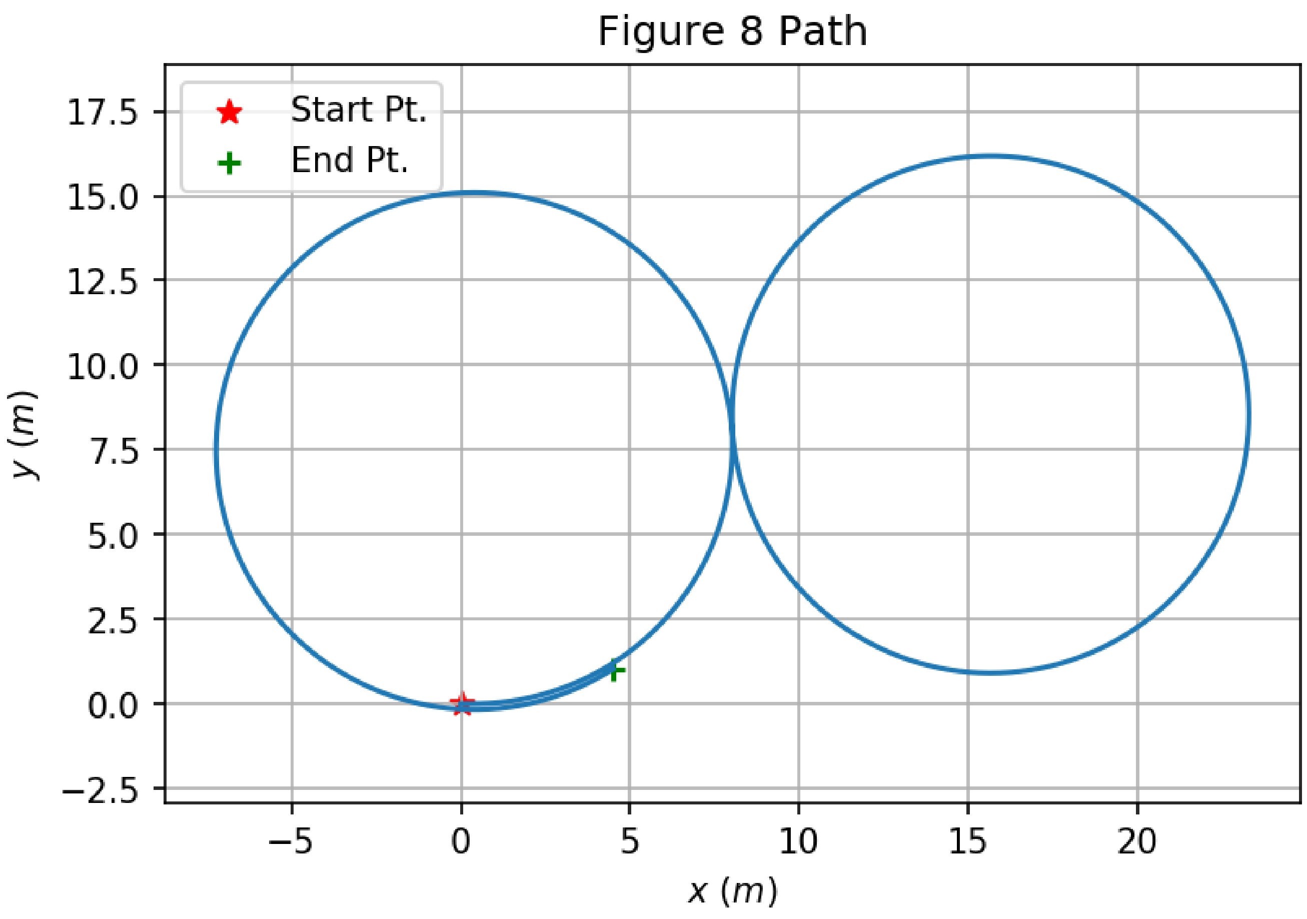
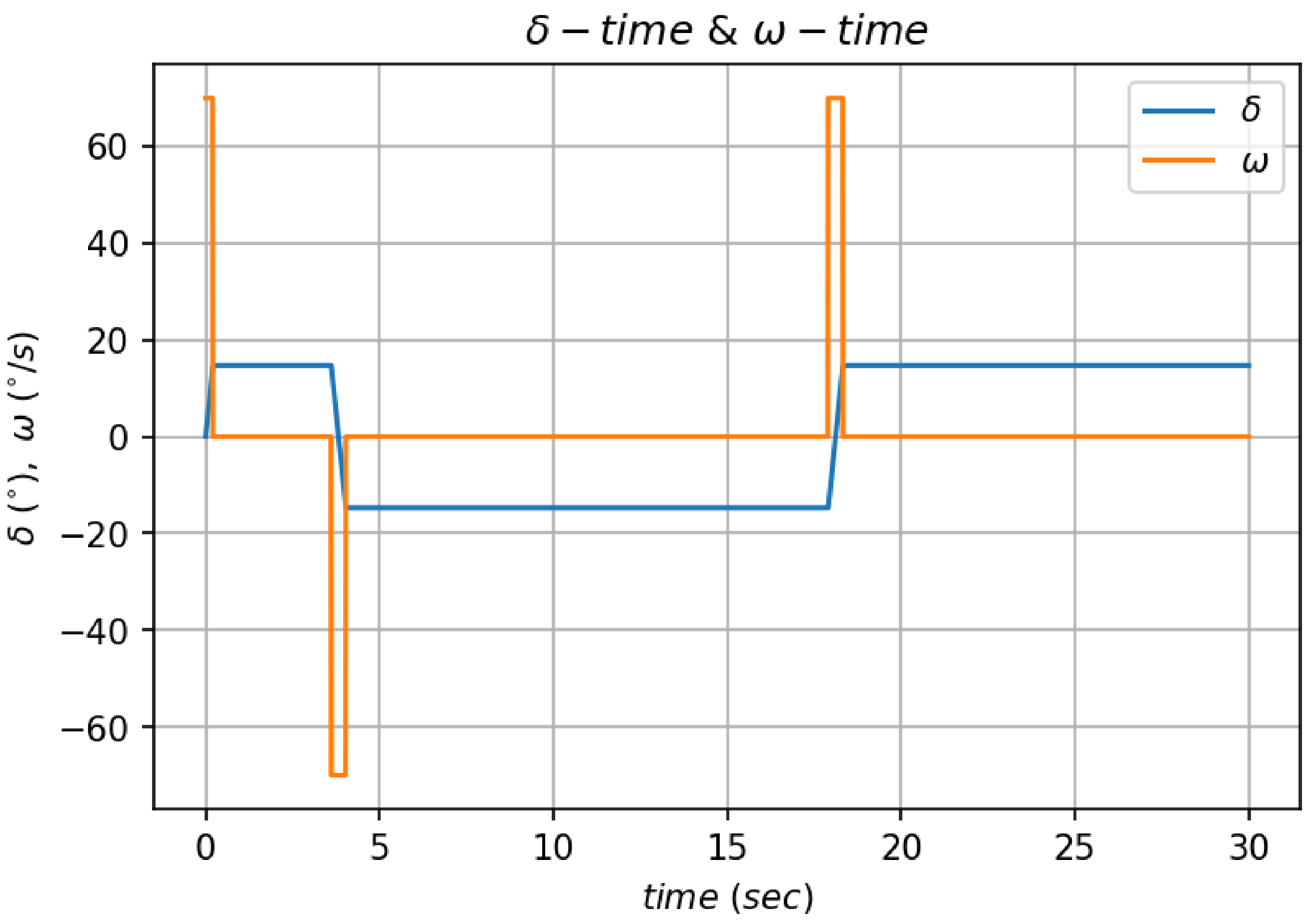


Figure 8 Path
Start Pt.
End Pt.
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
y (m)
−5
0
5
10
15
20
x (m)


$\delta - time$ & $\omega - time$
δ
ω
60
40
20
0
−20
−40
−60
$\delta\ (^\circ),\ \omega\ (^\circ/s)$
0
5
10
15
20
25
30
time (sec)