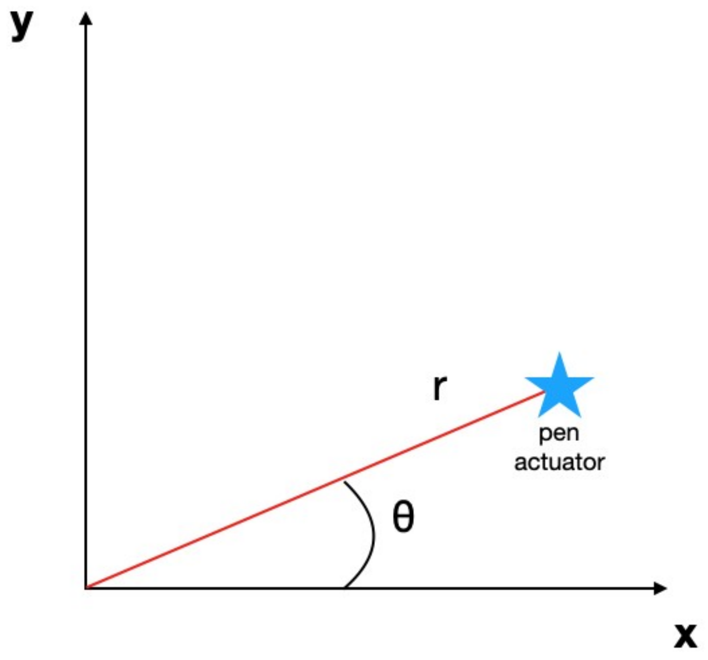
ME 405 - Homework 2

Robot Kinematics

Manual Calculations:

Overall Design Drawing:



Analysis:

The design of our robot follows the standard polar coordinate system. The two parameters are a radius, r, and an angle, θ. Our robot will be mounted at the bottom left corner of the drawing paper.

Two stepper motors will be used to drive both these variables. Thus, the proper stepping angles must be calculated to operate the system correctly.

Our first parameter, θ, will be used by the first stepper motor to move the system angularly (denoted as θ_1).

The radius parameter, r, moves along a lead screw with a 2mm pitch. By using the second stepper motor to rotate the lead screw, we can move to our desired radius. As such, we must calculate the stepper motor rotation needed to move a desired distance along the lead screw (denoted as θ_2). We know that there are 25.4mm per inch, so that implies there are 12.7 threads per inch. Furthermore, $\theta_2 \cdot \frac{2mm}{2\pi} \cdot \frac{1in}{25.4mm} = \frac{\theta_2}{\pi 25.4in}$. This constant, $\pi \cdot 25.4$, (denoted as α), is used below.

Recall the trigonometric relationship between cartesian and polar coordinates: $x=r\cos(\theta_1)$ and $y=r\sin(\theta_1)$

Initial Matrix:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{\theta_2}{\alpha} & \cos(\theta_1) \\ \frac{\theta_2}{\alpha} & \sin(\theta_1) \end{bmatrix}$$

$x=f(\theta)$

Jacobian Matrix:

$$\frac{\partial f}{\partial \theta} = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} \end{bmatrix}$$

$\frac{\partial f_1}{\partial \theta_1} = -\frac{\theta_2}{\alpha}sin(\theta_1)$

$\frac{\partial f_1}{\partial \theta_2} = \frac{1}{\alpha}cos(\theta_1)$

$\frac{\partial f_2}{\partial \theta_1} = \frac{\theta_2}{\alpha}cos(\theta_1)$

$\frac{\partial f_2}{\partial \theta_2} = \frac{1}{\alpha}sin(\theta_1)$

Find the velocity kinematics by differentiating x = f(θ) with respect to time:

$$x=\frac{d}{dt}(f(\theta))$$

$\frac{df_1}{dt} = \frac{d}{dt}[\frac{\theta_2}{\alpha}cos(\theta_1)] = \frac{1}{\alpha}cos(\theta_1) - \frac{\theta_2}{\alpha}sin(\theta_1)$

$\frac{df_2}{dt} = \frac{d}{dt}[\frac{\theta_2}{\alpha}sin(\theta_1)] = \frac{1}{\alpha}sin(\theta_1) + \frac{\theta_2}{\alpha}cos(\theta_1)$

$$x = \begin{bmatrix} -\frac{\theta_2}{\alpha}\sin(\theta_1) & \frac{1}{\alpha}\cos(\theta_1) \\ \frac{\theta_2}{\alpha}\cos(\theta_1) & \frac{1}{\alpha}\sin(\theta_1) \end{bmatrix} ¿$$

x = ¿

Inverse Kinematics:

$y=x-f(\theta)=g(\theta)$

$ \frac{\partial g(\theta))}{\partial \theta} = \frac{\partial}{\partial \theta}[0-f(\theta)] $

$ \frac{\partial g(\theta))}{\partial \theta} = -\frac{\partial}{\partial \theta}f(\theta) $

Source Code:

```
    Import Statements:
```

```
import numpy as np
import math
import matplotlib.animation as ani
import matplotlib.pyplot as plt
from PIL import Image
import math
import numpy as np
```

```
    Constant Values:
```

```
alpha = math.pi * 25.4
```

Implementing $g(\theta)$ for a particular value of x:

```
def g(x, theta):
    f1 = theta[1]*math.cos(math.radians(theta[0]))/alpha
    f2 = theta[1]*math.sin(math.radians(theta[0]))/alpha

    Theta = np.array([f1,f2])
    return np.subtract(x, Theta)
```

Implementing $\frac{\partial g(\theta)}{\partial \theta}$:

```
def dg_dtheta(theta):
    df1dt1 = -theta[1]*math.sin(math.radians(theta[0]))/alpha
    df1dt2 = math.cos(math.radians(theta[0]))/alpha
    df2dt1 = theta[1]*math.cos(math.radians(theta[0]))/alpha
    df2dt2 = math.sin(math.radians(theta[0]))/alpha
    jacob = np.array([[-1*df1dt1, -1*df1dt2],[-1*df2dt1,-1*df2dt2]])
    return jacob
```

A generic Newton-Raphson method for root finding:

```
def NewtonRaphson(fcn, jacobian, guess, thresh):
    steps = 1

    while (abs(fcn(guess)[0]) > thresh or abs(fcn(guess)[1]) >
thresh):
        math = np.subtract(guess,
np.matmul(np.linalg.inv(jacobian(guess)),fcn(guess)))
        guess = math
        steps += 1
    return guess
```

The following is a helper function for data plotting:

```python
def plotfunc(i):
    plt.clf()
    plt.xlim([0,10])
    plt.ylim([0,10])
    plt.ylabel('Y axis')
    plt.xlabel('X axis')
    p = plt.plot(datax[:i], datay[:i], color = 'red')
```

First, generate a circle of datapoints in cartesian coordinates. Then, call Newton-Raphson to calculate the respective motor angles. Lastly, convert these values back into cartesian coordinates for plotting:

```python
x = [1,0]
theta = [0, 0]
guess = [0.5,0.5]
datax = []
datay = []
values = []
for t in range(0, 120):
    x = 5 + 3*math.cos(math.pi*t*math.pi/180)
    y = 5 + 3*math.sin(math.pi*t*math.pi/180)
    X = [x,y]
    temp = NewtonRaphson(lambda theta : g(X, theta), dg_dtheta, guess,
.001)
    f1 = temp[1]*math.cos(math.radians(temp[0]))/alpha
    f2 = temp[1]*math.sin(math.radians(temp[0]))/alpha
    datax.append(f1)
    datay.append(f2)
```

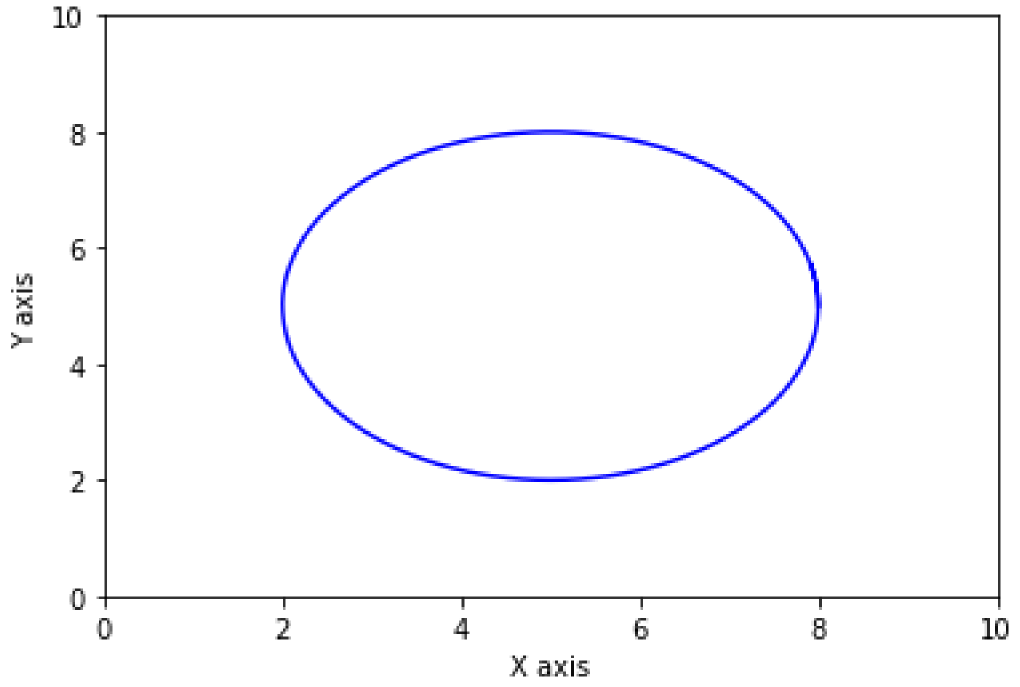
Plot the data:

```python
filenames = []
for i in range(len(datay)):
    fig  = plt.figure()
    plt.xlim([0,10])
    plt.ylim([0,10])
    plt.ylabel('Y axis')
    plt.xlabel('X axis')
    plt.plot(datax[:i], datay[:i], color = 'red')
    plt.plot([datax[i-1], 0], [datay[i-1], 0], color = 'blue')
    filename = f'{i}.png'
    plt.savefig(filename)
    filenames.append(filename)
    plt.close()
```

Generate a .gif of the robot drawing the circle from the images produced.

```python
frames =[Image.open(image) for image in filenames]
frame_one = frames[0]
```

```
frame_one.save('mygif.gif', format = "GIF", append_images = frames,
save_all = True, duration = 100, loop = 1)
fig  = plt.figure()
plt.xlim([0,10])
plt.ylim([0,10])
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.plot(datax, datay, color = 'blue')
plt.show()
```



Please see the final plot above, and the .gif I submitted via Canvas. Thank you.