ME 405 - 2.5 Degree of Freedom Plotter Kinematics

Author: Logan Williamson

Date: 5/7/2022

Description:

This file establishes the system kinematics describing operation of a 2.5 degree of freedom plotting robot. This machine consists of an aluminum extrude frame which supports a stepper-motor driven turntable drawing surface. Additionally, the upright supports of this frame support a ball screw linear actuator, also driven by a stepper motor. These two motion devices constitute the two degrees of freedom of this plotting machine's motion. On the ball screw, a 3D printed carriage conveys a ballpoint pen and a linear actuator. This linear actuator will extend or retract the pen, comprising the $\frac{1}{2}$ degree of freedom.

The motion of this device is described using a simplified schematic representation as seen in Appendix A. The two motion-producing mechanisms are resolved into a schematic wherein the motion of the mechanism can be represented by two variables, r and θ_2. The variable r is the linear/radial motion of the carriage subassembly on the ballscrew. This factor can be related to the rotational motion of the driving motor using the screw pitch of 5mm. This yields a driving system parameter, θ_1, related to the radial coordiate by the relationship $r = \frac{5}{2*\pi}\theta_1$.

From this setup, the system kinematics are analyzed by establishing uncoupled x- and y-coordinate equations representing the polar-to-Cartesian coordinate transform of the simplified schematic actuator "arm" vector. This yields the matrix equation $\mathbf{x} = f(\boldsymbol{\theta})$. Taking the Jacobian of the vectorized form of f yields a conversion matrix between our motor actuation angles, θ_1 and θ_2 and the corresponding Cartesian coordinates of the plotting pen tip, x and y.

Using this Jacobian, we can set up a Newton-Raphson iterative solver of the form

$$\boldsymbol{\theta_{n+1}} = \boldsymbol{\theta_n} - \left(\frac{\delta g(\boldsymbol{\theta_n})}{\delta(\boldsymbol{\theta_n})}\right)^{-1} g(\boldsymbol{\theta_n})$$

Upon passing the desired input vector $\mathbf{x}$ (containing the desired x and y Cartesian coordiantes) into the solver, along with an initial guess value for theta, the minimization fuction $g(\mathbf{x})$, the Jacobian of the system (or a function which computes it), and the acceptable error for a given solution, the solver will compute the associated vector $\boldsymbol{\theta}$ containing θ_1 and θ_2 motor actuation angles required to achieve that position.

To demonstrate how this will be used in the plotting robot's operation, the following script calls upon the g(x,theta) function, the dg_dtheta(theta) function, and the NewtonRaphson(fcn, jacobian, guess, thresh) to solve for the theta values associated with a series of points representing a parametrically generated ellipse. It then iterates through the parametric data representing these points and generates an actuator arm vector which 'traces' the elliptical path out frame-by-frame. Finally, these plots are saved as images, compiled, and used to generate a .gif animation as seen in Figure 1 below.

In [5]:
```
## Main Script - Newton Raphson and Plotter Motion gif Generation

import numpy as np
import time, math, imageio
import matplotlib.pyplot as plt

#Number of points in ellipse
n = 64
#Half of major axis length
a = 100
#Half of minor axis length
b = 50
#Lead screw pitch, [mm/revolution]
N = 5/(2*math.pi)
theta_guess = np.array([[125],[0.01]])
i=0
theta_store = []
#Generate n+1 length, equally-spaced time vector for
#tracing a full rotation in theta_2 direction
t = np.linspace(0,2*np.pi,n+1)
#Generate points in the ellipse
x = np.array([a*np.cos(t)])
y = np.array([b*np.sin(t)])
#Store ellipse position values in a 2 x n+1 array
x_desarray = np.vstack((x,y))

#Iterate through the points of the ellipse, solving for the associated
#theta values required to actuate the mechanism to each point. Convert
#results back to Cartesian coordinates for plot readability and ease of
#plotting. Plot the results parametrically.
filenames = []
for i in range(0,n+1):
    x_des = np.array([[x_desarray[0][i]],[x_desarray[1][i]]])
    theta = NewtonRaphson(lambda theta:g(x_des,theta),dg_dtheta,theta_guess,1e-6
    theta_guess = theta
    theta_store.append([theta[0][0],theta[1][0]])
#print(theta_store)

##Polar Plotting Attempts
#     f = plt.figure()
#     ax = f.add_subplot(111, polar='True')
#     ax.quiver([0, theta_store[i][1]], 0, N*theta_store[i][0])

#     fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
#     ax.plot(theta_store[i][1], N*theta_store[i][0],marker="D")
#     ax.set_rmax(110)
#     ax.grid(True)
#     ax.set_title("A line plot on a polar axis", va='bottom')
#     plt.show()

    plt.axis([-110,110,-110,110])
    plt.grid()
    plt.plot(x[0][:i+1],y[0][:i+1])
    #Convert theta values produced by Newton Raphson back to Cartesian for plotti
    x1 = [0, N*theta_store[i][0]*math.cos(theta_store[i][1])]
    y1 = [0, N*theta_store[i][0]*math.sin(theta_store[i][1])]
    plt.plot(x1,y1)

    #Create a file name and append it to a list
    filename = f'{i}.png'
    filenames.append(filename)

    #Save .png file of each plotting frame
    plt.savefig(filename)
    plt.close()

#Build gif
with imageio.get_writer('mygif.gif', mode='I') as writer:
    for filename in filenames:
        image = imageio.imread(filename)
        writer.append_data(image)
```

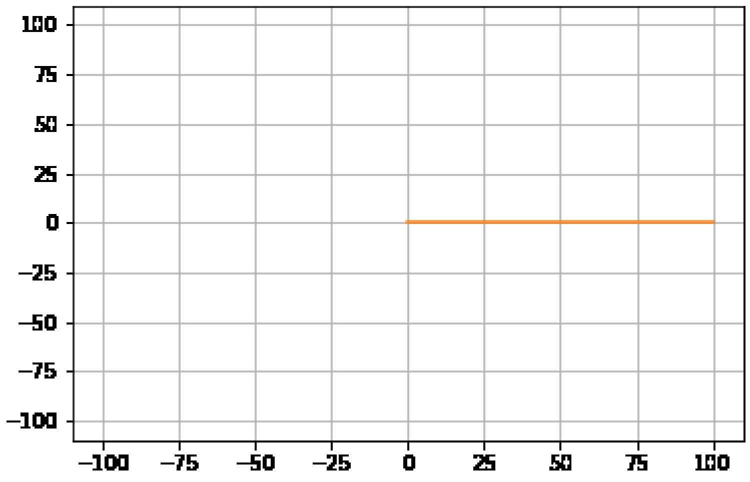


Figure 1: Animation of plotter motion, as seen from the drawing surface reference frame

The following function g(x,theta) takes an input np.array vector, x, containing two elements representing Cartesian x- and y-coordinates and an input np.array vector, theta, containing two elements (θ_1 and θ_2, corresponding to theta[0][0] and theta[1][0], respectively). It then computes the entries of the function vector $f(\theta)$ which is the polar to Cartesian coordinate transform. Finally, it carries out element-wise subtraction of f from x to generate the vector g.

In [1]:
```
def g(x, theta):
    f_arr = np.array([[N*theta[0][0]*math.cos(theta[1][0])],
                      [N*theta[0][0]*math.sin(theta[1][0])]])
    g = np.subtract(x,f_arr)
    return g
```

The function dg_dtheta(theta) takes an input numpy.array column vector object, theta, containing θ_1 and θ_2, corresponding to theta[0][0] and theta[1][0], respectively). It uses these polar coordinates to compute the partial derivative of the function g(x,theta) with respect to theta. The result is the Jacobian of the polar to Cartesian transform, negated. This will be used in the Newton-Raphson iterative solver for incremental calculation of polar to Cartesian coordinate transforms.

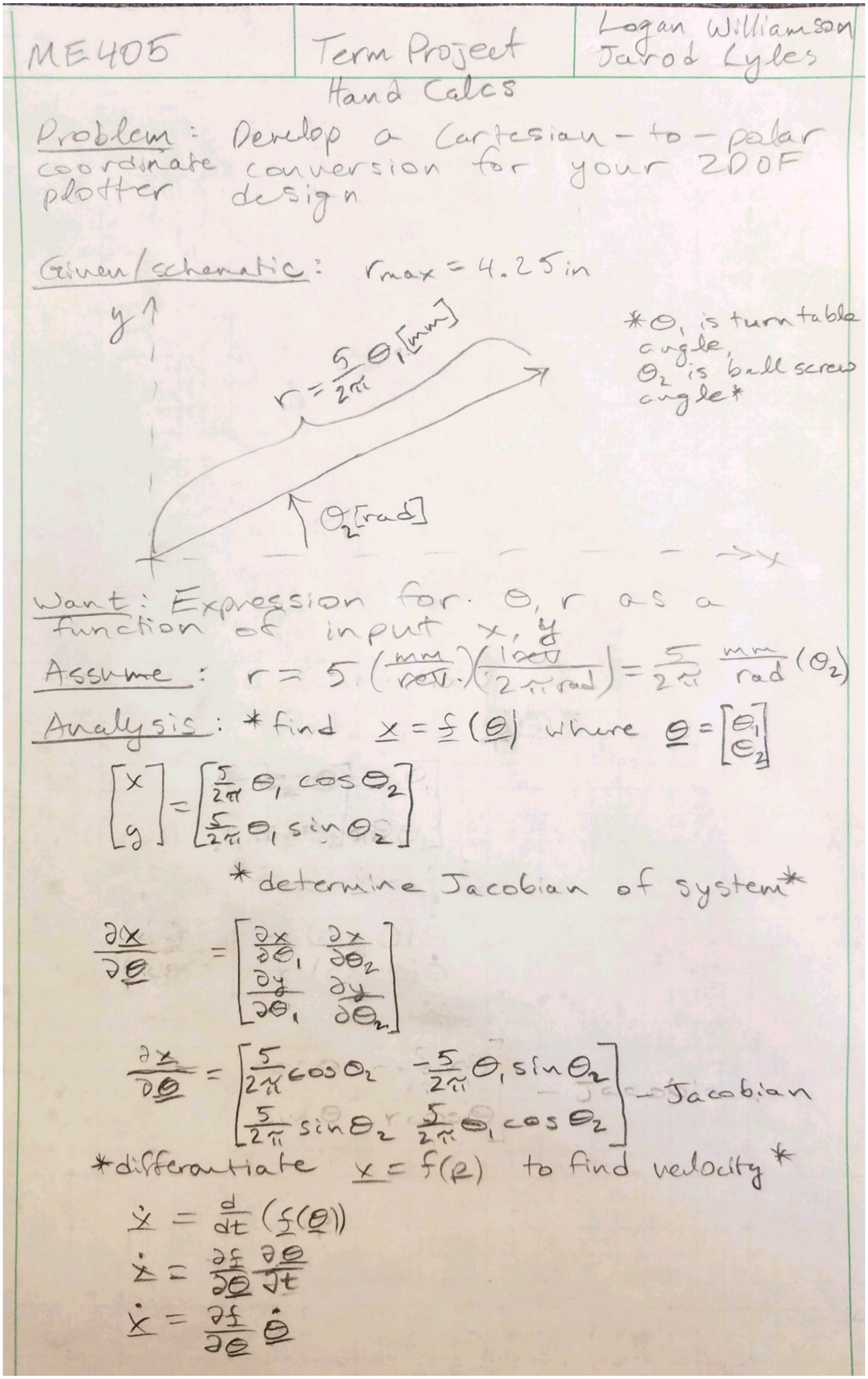
In [2]:
```
def dg_dtheta(theta):
    dg_dtheta = np.array([[-N*math.cos(theta[1][0]),N*theta[0][0]*math.sin(theta[
                          [-N*math.sin(theta[1][0]),-N*theta[0][0]*math.cos(theta
    return dg_dtheta
```

The function NewtonRaphson(fcn,jacobian,guess,thresh) takes in four arguments. These are:

- fcn: an anonymous function handle containing the desired output field of the function, and the input field to be varied by the solver
- jacobian: the Jacobian of the system being analyzed (which can be found by passing in the function dg_dtheta(theta)
- guess: an initial guess value used as a starting point for the first solver iteration
- thresh: an error threshold value used to determine if the output error of the function has reached an adequately close approximation of the theta value required to yield the static desired output of the anonymous function.

In [3]:
```
def NewtonRaphson(fcn,jacobian,guess,thresh):
    step = 0
    error = math.inf
    theta = guess
    while error > thresh:
        g = fcn(theta)
        error = np.linalg.norm(g)
        theta = np.subtract(theta,np.matmul(np.linalg.inv(jacobian(theta)),g))
        step+=1
    return theta
```

Appendix A: Hand Calculations

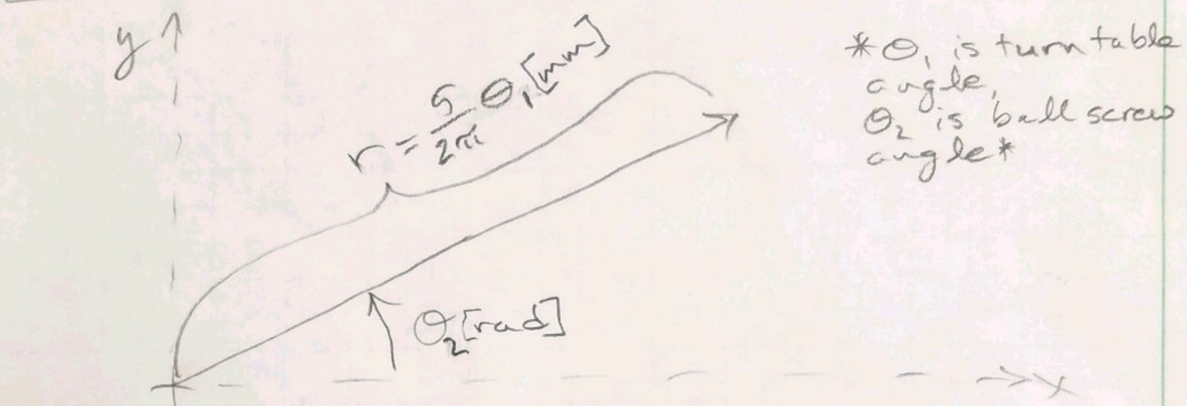


ME405	Term Project	Logan Williamson Jarod Lyles

Hand Calcs

Problem: Develop a Cartesian-to-polar coordinate conversion for your 2DOF plotter design

Given/schematic: $r_{max} = 4.25 in$


y
$r = \frac{5}{2\pi}\theta_1$ [mm]
θ_2 [rad]
x
θ_1 is turntable angle, θ_2 is ball screw angle

Want: Expression for θ, r as a function of input x, y

Assume: $r = 5\left(\frac{mm}{rev}\right)\left(\frac{1 rev}{2\pi rad}\right) = \frac{5}{2\pi} \frac{mm}{rad} (\theta_2)$

Analysis: * find $\underline{x} = \underline{f}(\underline{\theta})$ where $\underline{\theta} = \begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}$

$$\begin{bmatrix}x \\ y\end{bmatrix} = \begin{bmatrix}\frac{5}{2\pi}\theta_1 \cos\theta_2 \\ \frac{5}{2\pi}\theta_1 \sin\theta_2\end{bmatrix}$$

determine Jacobian of system

$$\frac{\partial \underline{x}}{\partial \underline{\theta}} = \begin{bmatrix}\frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2}\end{bmatrix}$$

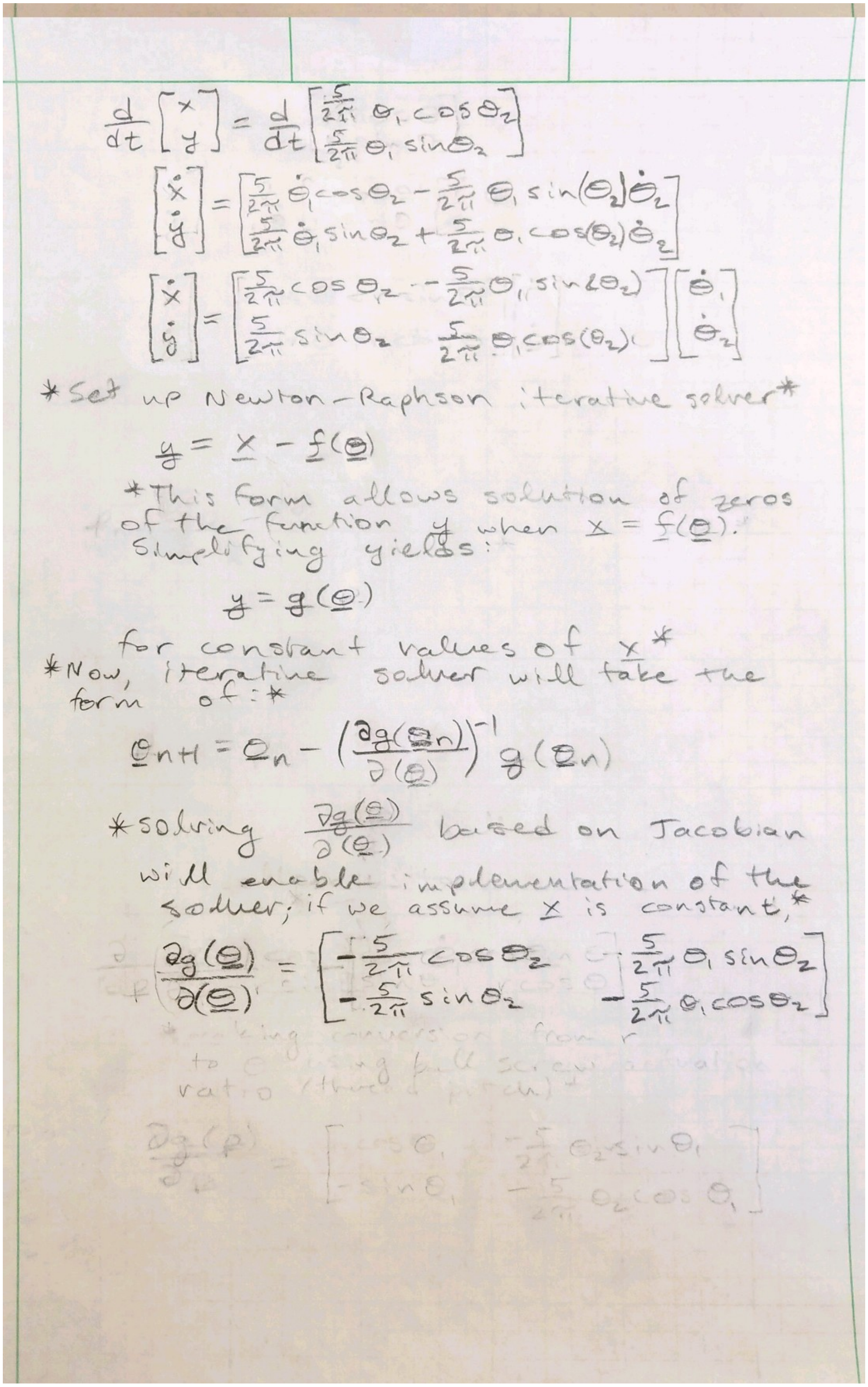
$$\frac{\partial \underline{x}}{\partial \underline{\theta}} = \begin{bmatrix}\frac{5}{2\pi}\cos\theta_2 & -\frac{5}{2\pi}\theta_1 \sin\theta_2 \\ \frac{5}{2\pi}\sin\theta_2 & \frac{5}{2\pi}\theta_1 \cos\theta_2\end{bmatrix} \text{— Jacobian}$$

differentiate $\underline{x} = f(\underline{\theta})$ to find velocity

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

$$\dot{\underline{x}} = \frac{\partial f}{\partial \underline{\theta}} \frac{\partial \underline{\theta}}{\partial t}$$

$$\dot{\underline{x}} = \frac{\partial f}{\partial \underline{\theta}} \dot{\underline{\theta}}$$



$$\frac{d}{dt}\begin{bmatrix} x \\ y \end{bmatrix} = \frac{d}{dt}\begin{bmatrix} \frac{5}{2\pi}\theta_1 \cos\theta_2 \\ \frac{5}{2\pi}\theta_1 \sin\theta_2 \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \frac{5}{2\pi}\dot{\theta}_1 \cos\theta_2 - \frac{5}{2\pi}\theta_1 \sin(\theta_2)\dot{\theta}_2 \\ \frac{5}{2\pi}\dot{\theta}_1 \sin\theta_2 + \frac{5}{2\pi}\theta_1 \cos(\theta_2)\dot{\theta}_2 \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \frac{5}{2\pi}\cos\theta_2 & -\frac{5}{2\pi}\theta_1 \sin(\theta_2) \\ \frac{5}{2\pi}\sin\theta_2 & \frac{5}{2\pi}\theta_1 \cos(\theta_2) \end{bmatrix}\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

Set up Newton-Raphson iterative solver

$$\underline{y} = \underline{x} - \underline{f}(\underline{\theta})$$

*This form allows solution of zeros of the function $\underline{y}$ when $\underline{x} = \underline{f}(\underline{\theta})$. Simplifying yields:

$$\underline{y} = \underline{g}(\underline{\theta})$$

for constant values of $\underline{x}$*

Now, iterative solver will take the form of:

$$\underline{\theta}_{n+1} = \underline{\theta}_n - \left(\frac{\partial \underline{g}(\underline{\theta}_n)}{\partial(\underline{\theta})}\right)^{-1} \underline{g}(\underline{\theta}_n)$$

solving $\frac{\partial \underline{g}(\underline{\theta})}{\partial(\underline{\theta})}$ based on Jacobian will enable implementation of the solver; if we assume $\underline{x}$ is constant,

$$\frac{\partial \underline{g}(\underline{\theta})}{\partial(\underline{\theta})} = \begin{bmatrix} -\frac{5}{2\pi}\cos\theta_2 & \frac{5}{2\pi}\theta_1 \sin\theta_2 \\ -\frac{5}{2\pi}\sin\theta_2 & -\frac{5}{2\pi}\theta_1 \cos\theta_2 \end{bmatrix}$$

making conversion from r to θ using ball screw actuation ratio (thread pitch)

$$\frac{\partial g(p)}{\partial [illegible]} = \begin{bmatrix} \cos\theta_1 & -\frac{5}{2\pi}\theta_2 \sin\theta_1 \\ -\sin\theta_1 & -\frac{5}{2\pi}\theta_2 \cos\theta_1 \end{bmatrix}$$