Homework Documentation

Kyle MacMillan

Aug 31, 2018

CONTENTS:

1 Homework 1 **1**
1.1 Chapter 1 1
1.2 Chapter 2 2

CHAPTER ONE

HOMEWORK 1

Kyle MacMillan

Problems:

Ch1: 1, 8, 15

Ch2: 9, 10, 11, 29, 31

Text Addition

1.1 Chapter 1

1. **How would you define a robot?**

   The book defines a robot as follows, “A robot is seen as a sophisticated machine that replaces human effort”. In class we further defined a “robot” as a construct that has a changing meaning over time. “I” would define a robot as a mechatronic entity that is capable of performing actions. Yes, this is very broad because robot is a general term.

8. **Do you think the Robotic Appliance and Robotic Agent partitioning is a more effective way to classify robots? Why or why not?**

   I do like the classification. Grouping is essential so robots can fit certain functions of the end-user. This also helps people less familiar with the art understand that robots are built for specific purposes, which isn’t always apparent. The book makes note that when a classification restricts innovation it is dead to us. I absolutely agree, but again, bounding things really helps to limit scope and so it definitely has a purpose.

15. **Work in robotics can replace people with machines. This results in job loss. Discuss the ethics of working in the robotics industry.**

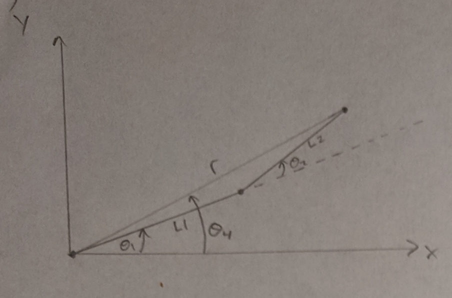
    I could literally write a book on this topic, and if you summed all the posts I’ve made across all the social media I use I’m sure you could piece together said book. That being said, I’ll keep this “brief”.

    Working for a living is a concept that will die within the millenial’s lifetime. It is what I believe to be an inevitability, accelerated by capitalism’s desires to make maximum profit. Coming to this conclusion I decided I want to work in AI or robotics so I can make the machines that replace humans; I want to accelerate it, to progress humanity to the next epoch. That being said I am fully aware it is going to cause immense issues as people are unable to find jobs but that is quite frankly not my problem. That is a societal issue. Planned obsolecense, consumerism, food shortages, and a lot of what is wrong with the world can go away if we accelerate robotic growth. Accelerating job loss is a good thing if the government did their job. Automated vehicles will save tens of thousands of lives annually, the job loss is a concern but is heavily outweighed by the benefit. In my opinion it is on the government’s shoulders to recognize (listen to the very loud groups that have been saying it for years) and effect change that will benefit the people, not the corporations.

1.2 Chapter 2

9.

a.)



arm 1

$x = L1 \cos\theta_1$

$y = L1 \sin\theta_1$

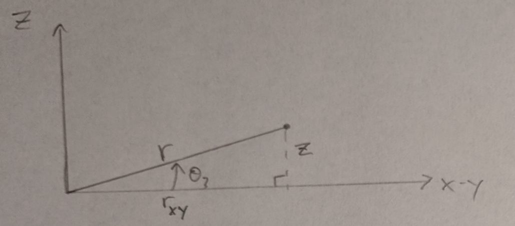
arm 2

$x = L2 \cos(\theta_1 + \theta_2)$

$y = L2 \sin(\theta_1 + \theta_2)$

$r = \text{arm1} + \text{arm2}$

$\theta_4 = \tan^{-1}\left(\frac{r_y}{r_x}\right)$



$$\boxed{\left.\begin{aligned} x &= r\cos\theta_4 \\ y &= r\sin\theta_4 \\ z &= r\sin\theta_3 \end{aligned}\right\} \text{Forward Kinematic equations}}$$

b.) $r = \sqrt{x^2 + y^2}$ $\quad \therefore \boxed{\theta_3 = \tan^{-1}\left(\frac{z}{r_{xy}}\right)}$

$r_{xy} = \sqrt{r^2 - z^2}$

From the book:

$$D \equiv \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2}$$

$$\boxed{\theta_2 = \tan^{-1}\left(\frac{\pm\sqrt{1 - D^2}}{D}\right)}$$

$\pm$ Because of the joint types. A two link manipula with something like spherical joints could have "infinite" for each θ.

Also from the book: $\gamma = \tan^{-1}\left(\frac{a_2 \sin\theta_2}{a_1 + a_2 \sin\theta_2}\right)$

$$\boxed{\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \gamma}$$

10. Code:

```
import numpy as np
import matplotlib.pyplot as plt


class twolink():
        """ This class is meant for fk and ik operations around a 2-link␣
↪manipulator. """

    def __init__(self, length1, length2):
        """ Class initialization """
        self.a1 = length1
        self.a2 = length2

    def fk(self, theta1, theta2):
        """ Calculate the forward kinematics to determine the x & y values """
            x = self.a2 * np.cos(theta1 + theta2) + self.a1 * np.cos(theta1)
            y = self.a2 * np.sin(theta1 + theta2) + self.a1 * np.sin(theta1)
            return x,y

    def ik(self, x, y):
        """ Calculates the inverse kinematics to determine the theta1 & theta2␣
↪values """
        D = (x*x + y*y - self.a1 * self.a1 - self.a2 * self.a2) / (2 * self.a1 *␣
↪self.a2)
        theta2 = np.arctan2(np.sqrt(1 - D * D), D)
        gamma = np.arctan2((self.a2 * np.sin(theta2)), (self.a1 + self.a2 * np.
↪cos(theta2)))
        theta1 = np.arctan2(y, x) - gamma

        return theta1, theta2

    def plot(self, funct, show = True):
        """ Adds a function to the plot and shows it or not """
        plt.plot(funct[0], funct[1])
        if show:
            plt.show()




def func1():
    x = np.arange(0, 25, 0.05)
    y = 25 - x
    return x,y

def func2():
    t = np.arange(0, np.pi, 0.05)
    x = 10 * np.cos(t) + 15
    y = 10 * np.sin(t)
    return x,y


if __name__ == "__main__":
    t = twolink(15, 15)
    f1 = func1()
    f2 = func2()
```

(continues on next page)

(continued from previous page)

```
    # Plot function one
    t.plot(f1)
    ikf1 = t.ik(f1[0], f1[1])
    # Plot the arm movement
    t.plot(ikf1)

    # Plot function one
    t.plot(f2)
    ikf2 = t.ik(f2[0], f2[1])
    # Plot the arm movement
    t.plot(ikf2)
```

11. Code:

```
import numpy as np
import matplotlib.pyplot as plt
n = 50 # for linspace

class twolink():
    """ This class is meant for fk and ik operations around a 2-link manipulator.
        This was updated from problem 10 to allow for starting theta values. """

    def __init__(self, length1, length2, *args):
        """ Class initialization """
        self.a1 = length1
        self.a2 = length2
        self.theta1 = [0.0]
        self.theta2 = [0.0]
        self.x = [0.0]
        self.y = [0.0]

        # Assign theta1/2 if they were passed in
        if len(args) != 2:
            return

        self.theta1 = args[0]
        self.theta2 = args[1]


    def setXY(self, x, y):
        """ Set the class's x and y values """
        self.x = x
        self.y = y

    def setThetas(self, theta1, theta2):
        """ Set the class theta values """
        self.theta1 = theta1
        self.theta2 = theta2


    def fk(self):
        """ Calculate the forward kinematics to determine the x & y values """
        self.x = self.a2 * np.cos(self.theta1 + self.theta2) + self.a1 * np.
↪cos(self.theta1)
        self.y = self.a2 * np.sin(self.theta1 + self.theta2) + self.a1 * np.
↪sin(self.theta1)
```

(continues on next page)

(continued from previous page)

```
        return self.x, self.y

    def ik(self):
        """ Calculates the inverse kinematics to determine the theta1 & theta2␣
↪values """
        D = (self.x*self.x + self.y*self.y - self.a1 * self.a1 - self.a2 * self.
↪a2) / (2 * self.a1 * self.a2)
        self.theta2 = np.arctan2(np.sqrt(1 - D * D), D)
        gamma = np.arctan2((self.a2 * np.sin(self.theta2)), (self.a1 + self.a2 *␣
↪np.cos(self.theta2)))
        self.theta1 = np.arctan2(self.y, self.x) - gamma

        return self.theta1, self.theta2

    def plot(self, funct, show = True):
        """ Adds a function to the plot and shows it or not """
        plt.plot(funct[0], funct[1])
        if show:
            plt.show()


# The following functions could be turned into a polygon class but was unecessary␣
↪for this one problem
def line1():
    y = np.linspace(0, 15, n, endpoint=True)
    x = np.linspace(5, 5, n, endpoint=True)
    return x, y

def line2():
    y = np.linspace(15, 15, n, endpoint=True)
    x = np.linspace(5, 20, n, endpoint=True)
    return x, y

def line3():
    y = np.linspace(15, 0, n, endpoint=True)
    x = np.linspace(20, 20, n, endpoint=True)
    return x, y

def line4():
    y = np.linspace(0, 0, n, endpoint=True)
    x = np.linspace(20, 5, n, endpoint=True)
    return x, y


if __name__ == "__main__":
    t = twolink(15, 15)
    l1 = line1()
    l2 = line2()
    l3 = line3()
    l4 = line4()

    # Add lines to the plot so you can see each individually
    t.plot(l1, False)
    t.plot(l2, False)
    t.plot(l3, False)
    t.plot(l4)
```

(continues on next page)

(continued from previous page)

```
    # Place all points into two np arrays for kinematics and plotting
    x = np.append(l1[0], [l2[0], l3[0], l4[0]])
    y = np.append(l1[1], [l2[1], l3[1], l4[1]])
    t.setXY(x,y)
    t.plot(t.ik())
```

29. Code:

```
import numpy as np
import matplotlib.pyplot as plt
import sys

# constants
D = 10.0
r = 5.0
L = 16.0
circ = 2 * r * np.pi



def xt(w1, w2, t, theta, x0):
    return ((L * (w1 + w2)) / (w1 - w2)) * (np.sin((r / (2.0 * L)) * (w1 - w2) *␣
↪t + theta) - np.sin(theta)) + x0

def yt(w1, w2, t, theta, y0):
    return (L * (w1 + w2) / (w1 - w2)) * (np.cos((r / 2.0 * L) * (w1 - w2) * t +␣
↪theta) - np.cos(theta)) + y0

if __name__ == "__main__":
    # Set initial variables
    positions = np.array([[0.0,0.0,0.0],])
    t=5.0
    x_position = 0.0
    y_position = 0.0

    # From time = 0 to 5
    x_position = (2.0 / (2.0 * np.pi) ) * circ * t
    positions = np.concatenate((positions, [[x_position,0.0,0.0],]))


    # From time = 5 to 6
    w1 = 3.0 / (2.0 * np.pi)
    w2 = 4.0 / (2.0 * np.pi)
    t = 1.0
    theta = (r / 2 * L) * (w1 - w2) * t + 0.0
    x_position = xt(w1, w2, t, theta, x_position)
    y_position = yt(w1, w2, t, theta, y_position)
    ary = np.array([[x_position,y_position,0.0],])
    positions = np.concatenate((positions, ary), axis=0)


    # From time = 6 to 10
    w1 = 1.0 / (2.0 * np.pi)
    w2 = 2.0 / (2.0 * np.pi)
    t = 4.0
    theta = (r / 2.0 * L) * (w1 - w2) * t + theta
    x_position = xt(w1, w2, t, theta, x_position)
```

(continues on next page)

(continued from previous page)

```
    y_position = yt(w1, w2, t, theta, y_position)
    positions = np.concatenate((positions, [[x_position,y_position,0.0],]))

    print('Final position: ', ' '.join(map(str, positions[3:4])))
```

31. **Show that the differential drive kinematic equations are non-holonomic constraints.**

    Position and orientation are supposed to be independent for holomonic functions. With differential drive if you are only able to travel in a straight line then the kinematic equation would be holonomic, but when calculating the position we use an equation of the form: t = d1/v1 = d2/v2, which is not holonomic. The book has a description of a holonomic constraint as follows: "A holonomic constraint only depends on the coordinates and time and does not depend on derivatives".

    If both wheels moved at an equal rate then they could be pulled out as a constant of integration but since a differential drive robot has variable wheel speeds we are not able to claim the equations are holonomic.