Design and Simulation of a Robotic Arm using Computer Programming

Himani Madaan, Devarsh Randeria, Sanjay Kumar Purty

INTRODUCTION:

A robotic arm is a type of mechanical arm, usually programmable, with similar functions to a human arm; the arm may be the sum total of the mechanism or may be part of a more complex robot. The links of such a manipulator are connected by joints allowing either rotational motion (such as in an articulated robot) or translational (linear) displacement.The links of the manipulator can be considered to form a kinematic chain. The terminus of the kinematic chain of the manipulator is called the end effector and it is analogous to the human hand. A robotic arm can be used for varied applications such as assembly operations, workpiece handling, pick and place, application of sealant, spot welding operation etc. This paper mainly talks about a pick and place robotic arm.

Pick and place robotic arm picks a particular object from a given position and drops it to another position from where the work further takes place.Pick and place robots are commonly used in modern manufacturing environments.Pick and place robots handle repetitive tasks while freeing up human workers to focus on more complex work.Automating this process helps to increase production rates.

Pick and place robots are not only used in manufacturing but are also used in other applications some of which are listed below:

- **Assembly** – Pick and place robots used in assembly applications grab incoming parts from one location, such as a conveyor, and place or affix the part on another piece of the item. The two joined parts are then transported to the next assembly area.
- **Packaging** – Pick and place robots used in the packaging process grab items from an incoming source or designated area and place the items in a packaging container.
- **Bin picking** – Pick and place robots used in bin picking applications grab parts or items from bins. These pick and place robots typically have advanced vision systems allowing them to distinguish colour, shape and size to pick the right items even from bins containing randomly mixed items. These parts or items are then sent to another location for assembly or packaging.
- **Inspection** – Pick and place robots used for inspection applications are equipped with advanced vision systems to pick up objects, detect anomalies and remove defective parts or items by placing them in a designated location.

This paper mainly talks about the design and simulation of a 2R link manipulator using computer programming i.e, how the link dimension will vary because of change in payload and how both the links will move when they are given a task of moving from one coordinate to the other for pick and place operation.

DESIGN:

The robotic arm is assumed to be a shaft and the design is done accordingly.
The weights of the individual links are considered to be negligible.

Consider a payload of mass **m** kg that needs to be lifted by a pick and place robot.
Let **L** be the total length of the robotic arm link.
So, **L=L1+L2** where L1 and L2 are lengths of link 1 and 2 respectively where $\frac{L1}{L2} = 1.5$.
Let **d** be the diameter of the robotic arm whose dimension is considered to be 5mm.
Robotic arm is made up from **Aluminium 6061**.
The yield strength of Al 6061 is considered to be150 MPa and a factor of Safety(FOS) of 3 is considered.
Now when the shaft is subjected to pure bending moment, the bending stress is given by,

$$\sigma_b = \frac{M\frac{d}{2}}{\frac{\pi d4}{64}}$$

$$\sigma_b = \frac{32M}{\pi d3},$$

Yield tensile strength of aluminium(also considering FOS) should be greater than the bending stress generated, thus,

$\frac{150}{3} \geq \frac{32M}{\pi d3}$, where M = mgL

$$\frac{150}{3} \geq \frac{32mgL}{\pi d3}$$

$$L \leq \frac{50\pi d3}{32mg}$$

Here the mass value will be taken from the user and accordingly the total length of the arm and the individual length of the links will be calculated.

CODING:
This part of the term paper displays and explains the code used for implementing the above mentioned simulation.

```
In [1]: # import required libraries

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

First of all, required libraries are imported which will be used in the further code to perform certain functions. General use of these libraries is as follows:

NumPy is a Python library used for working with arrays
Python **Math** Library provides us access to some common mathematical functions and constants in Python
pyplot is a plotting library used for 2D graphics

In [3]:
```
# define initial (fixed) position of robotic arm

x0=0
y0=0
```

This defines the initial position of the robotic arm at origin.

In [4]:
```
# to select the best path for robotic arm motion
# (based on minimum area to be swept)

def find_opt(val, initial_pos):
    swept_area=100000
    ans=val[0]
    for v in val:
        area=0
        start = min(v[1], initial_pos[1])
        end = max(v[1], initial_pos[1])
        area+=math.pi*L2*L2*(end-start)
        start = min(v[0], initial_pos[0])
        end = max(v[0], initial_pos[0])
        area+=math.pi*L1*L1*(end-start)
        print("Area for", v, "is", area)
        if(area<swept_area):
            swept_area=area
            ans=v

    return ans
```

Here a function named **find_opt** is created in order to find the optimum path for the robot to travel from one point to another based on the minimum area to be swept. This function needs to be created because there may be one or values of link angles corresponding to a particular location and out of this an optimum needs to be selected.

The function takes two variables as input one is **val** which is a list that stores different link angles to reach that particular position and **initial_pos** takes the angular value of the initial position given. Here a variable swept_area is defined at some particular value and the area obtained is then compared with this swept_area, if the given area is less than the swept_area then that area becomes equal to swept_area and by running the for loop we finally get the optimised value.

This is the function definition only. It will work when a function call is given in further sections to get the best path to be followed by links to reach desired pick and place points. So it will be called twice, one to get the most optimal result for pick position (based on initial default position), and second at the time of place position (based on the initial pick position).

In [5]:
```
x1=float(input('Enter x coordinate of pick position:  '))
y1=float(input('Enter y coordinate of pick position:  '))
```

This code takes the input from the user for the location from which robots need to do the picking operation. So we need to take our arm from [0, L1+L2] position to [x1,y1] coordinates.

```
In [8]: # solve trignometric equations to get inclination angles of links

        from sympy import *

        a1, a2 = symbols('a1,a2') # here a1 is inclination angle of L1 with x axis
                                  # and a2 is inclination angle of L2 with x-axis

        eq1 = Eq((L1*cos(a1)+L2*cos(a2)), x1)
        # print("Equation 1:", eq1)

        eq2 = Eq((L1*sin(a1)+L2*sin(a2)), y1)
        # print(eq2)


        try:
            val=solve([eq1, eq2], a1, a2)

            print("Posible values of inclination angles are:  ")
            print(val)


        except NotImplementedError:
            print("Not possible to reach this position. Please re-enter the coordinates")
```

This code calculates the angle that both the links need to move from origin in order to reach the particular position. These angles are calculated using two equations:

X1= L1cos(a1)+L2cos(a2).
Y1= L1sin(a1)+L2cos(a2).
Here X1,Y1 represents the x-y coordinates.
L1,L2 are the length of the links.
a1,a2 are the link angles.

Here a sympy.solve() function is used to find out values of a1 and a2. This is an inbuilt function in python. All the possible pairs of values of (a1,a2) will be stored in a list named ‘val’. And if there is no solution to this set of equations, an exception will be generated showing that it is not possible to reach this position.

```
In [12]: # movement of L1 link
         def print_L1_movement(theta, fixed_angle):
             global figno
             t2=fixed_angle
             for t1 in theta:
         #     for t2 in theta2:
                 x1=L1*math.cos(t1)
                 y1=L1*math.sin(t1)
                 x2=x1+L2*math.cos(t2)
                 y2=y1+L2*math.sin(t2)

                 filename=str(figno)+'.jpg'
                 figno = figno+1

                 plt.figure()
                 plt.plot([x0,x1],[y0,y1])
                 plt.plot([x1,x2],[y1,y2])
                 plt.xlim([0,2])
                 plt.ylim([0,2])
                 plt.savefig(filename)
```

After receiving the most optimal pair of angles of inclinations of our links, we need to show the movement of those links from their initial position. This role is performed by ‘*print_L1_movement*’ and ‘*print_L2_movement*’ functions for movement of links L1 and L2 respectively, one by one. When L1 movement is shown, L2 is kept fixed and vice-versa for L2 movement. The function takes two

inputs i.e. *theta* (containing step angles to move L1 from initial angle to final angle) and *fixed_angle* which is the angle of second link to be kept fixed.

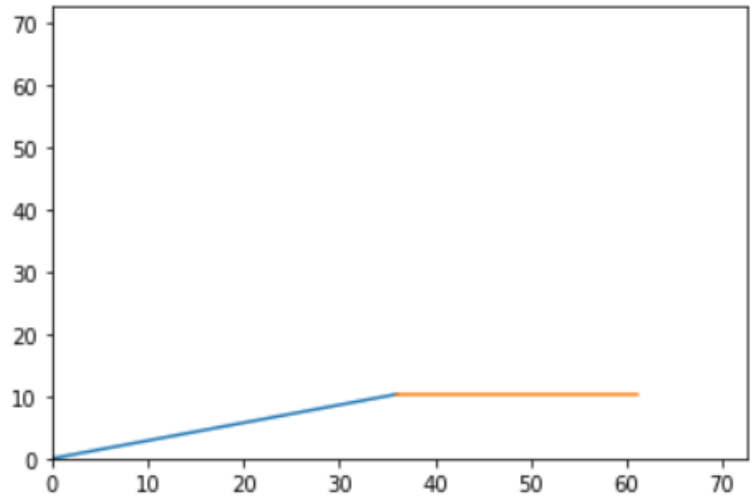
```
In [14]: # Find most optimal pair of inclination angles from the above possible solutions

         pick_pos = find_opt(val,[0,0])
```

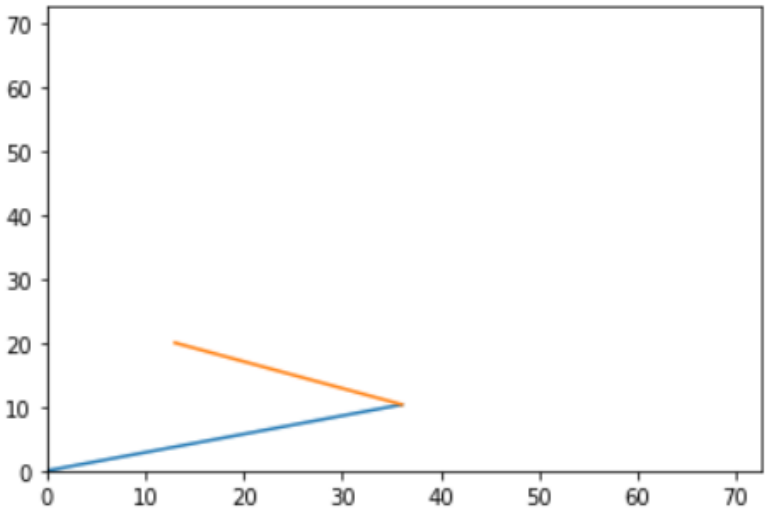
This section of code is calling the find_opt function which was defined earlier by sending arguments val (all possible angles of L1 & L2) with [0,0] initial position and returning optimum pair of angles L1 and L2 which we are storing in pick_pos.

```
In [ ]: theta1=[0,pick_pos[0]/4, pick_pos[0]/2, pick_pos[0]*3/4,pick_pos[0]]
        t2=0

        print_L1_movement(theta1,t2)
```

```
In [ ]: t1=pick_pos[0]
        theta2=[0,pick_pos[1]/4, pick_pos[1]/2, pick_pos[1]*3/4,pick_pos[1]]


        print_L2_movement(theta2,t1)
```

In this code we are calling both the function for movement of L1 arm and movement of L2 arm i.e print_L1_movement & print_L2_movement and also capturing the positions of both, where first argument is array of angles (of size 5 where the final angle is divided into sub angles so that we can obtain enough number of images to get proper simulation) and the second being the constant angle with respect to L1 it is t2=0 and with respect to L2 it is t1. The plots thus generated are as follows: Here payload is 2 kg and the pick coordinates to reach is (13,20).





These plots are then saved in our computer as well to create an animated video showing the movement of arm.

In [ ]:
```
# to find most optimal movement of arm based on previous position

place_pos = find_opt(val1, pick_pos)
```

In [ ]:
```
# movement of L1 from pick position to place position

theta3 = [pick_pos[0], (pick_pos[0]+place_pos[0])/2, place_pos[0]]
t4=pick_pos[1]

print_L1_movement(theta3,t4)
```

In [ ]:
```
# movement of L2 from pick position to place position

theta4 = [pick_pos[1], (pick_pos[1]+place_pos[1])/2, place_pos[1]]
t3=place_pos[0]


print_L2_movement(theta4,t3)
```

In the same manner, input is taken for place position coordinates by the user and the whole process is repeated to move the arm from ‘pick position’ to the final ‘place position’. Code snippets to be used are shown above. Again, the plots are generated and stored in the same manner to finally create a simulation video.

```
import glob
from PIL import Image,ImageDraw

# Create the frames
x = []


for i in range(1,figno):
    new_frame = Image.open(str(i)+'.jpg')
    x.append(new_frame)

for i in range(1,7):
    new_frame = Image.open(str(figno-1)+'.jpg')
    x.append(new_frame)
x[0].save('png_to_gif.gif', format='GIF',append_images=x[1:],save_all=True,duration=200, loop=0)
```

This code converts the images generated into Gif format so that we can obtain a simulation.

Future Work:

The simulation done above does not identify an object, it only performs a pick and place operation. Furthermore object identification can be added to the robotic arm using machine learning i.e, it only picks a specified colour object from a range of colours in a given plane and then picks it and drops it to the required location, here the major problem is that because of large number of objects some object might interrupt the path which robot may choose to travel because of which path optimization needs to be performed using inverse kinematics method which is currently beyond the scope of this paper.

Python Code Link:

https://drive.google.com/file/d/1hMMUGNlNVgi2FVtHHVqPDRASyTXIw1Ww/view?usp=sharing

Sample Simulation Link:

https://drive.google.com/file/d/1DOG2YclE1kHE-RIaaZbFTilJu8zRMDNn/view?usp=sharing