Trajectory Planning for Manipulator Using Genetic Algorithm

Report submitted to the SASTRA Deemed to be University as the requirement for the course

BCSCCS708: MINI PROJECT

Submitted by

ARAVIND B
(Reg.No.120003027,Computer Science and Engineering)
BHARATH KUMAR K C S
(Reg.No.120003051,Computer Science and Engineering)
KRISHNA PRASAD M K R
(Reg.No.120003149,Computer Science and Engineering)

December 2020

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING
THANJAVUR - 613 401

BONAFIDE CERTIFICATE

This is to certify that the report titled "**Trajectory Planning for Manipulator Using Genetic Algorithm**" submitted as a requirement for the course, BCSCCS708: **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr.ARAVIND B** (Reg.No.120003027,Computer Science and Engineering),**Mr.BHARATH KUMAR K C S** (Reg.No.120003051,Computer Science and Engineering), **Mr. KRISHNA PRASAD M K R**(Reg.No.120003149, Computer Science and Engineering) during the academic year 2020-2021, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation: Mr. SURESH K S.,AP-III,SoC,SASTRA Deemed University

Date :

Submitted for Project Viva Voce held on

Examiner - I **Examiner - II**

ACKNOWLEDGEMENT

First and foremost we thank the almighty for helping me to gain support in all forms to finish our Mini Project successfully. We express our sincere thanks to DR.S.VAIDHYASUBRAMANIAM, Vice Chancellor and Dr.R.CHANDRAMOULI, Registrar, SASTRA DEEMED UNIVERSITY, for permitting us to do this Mini Project as a part of our curriculum.

We express our profound gratitude to DR.UMAMAKESHWARI A, Dean, School of Computing, SASTRA DEEMED UNIVERSITY and DR.SHANKAR SRIRAM V S, Associate Dean, School of Computing, SASTRA DEEMED UNIVERSITY for their complete support throughout the Mini Project.

We are fortunate to have, Mr. SURESH K S, Assistant Professor, SoC, as our project guide. His valuable assistance and supervision guided us towards the successful completion of our Mini Project. Also we would like to express our gratitude to Mr. RAJENDIRAN P, the Project Coordinator. Finally we thank all the technical and non-technical staff of Computer Science and Engineering Department and parents and friends for their constant support throughout the completion of the report.

List of Figures

ABSTRACT

The robotic arm manipulators are presently used in a wide number of applications and industries. Obstacle avoidance is a major problem faced by it while working. This is because mostly these manipulators are working in a crowded work space.

So to overcome this kind of problem an algorithm called Genetic Algorithm tries to find a suitable trajectory for the manipulator to move in its work space. This Algorithm uses certain operators like Selection, Crossover and Mutation. Before implementing this algorithm some kinematic analysis is done to generate the equations which will be the input for the algorithm. The coordinates of obstacles are passed as parameter to ensure that trajectory avoids obstacles. The algorithm is implemented for four sample inputs in the coordinate system and results are verified.

KEYWORDS: Robotic Manipulator,Obstacle Avoidance, Kinematic Analysis, Genetic Algorithm, Selection, Mutation, Cross-over.

TABLE OF CONTENTS

Chapter 1

Contents of the Base Paper

1 Paper Details

THE JOURNAL OF ENGINEERING , VOLUME 2018, ISSUE 16 ,PP 1579-1586, NOVEMBER 2018
doi: 10.1049/joe.2018.8266
A. Jiang, X. Yao and J. Zhou , Xiangtan University ,Republic of China

2 Introduction

In this 21st century, mechanical robots and machines are replacing people. As Bill Gates told, “Robotics and other combinations will make the world pretty fantastic compared with today”. Likewise, from the packaging of foods to defence fields robots are playing a vital role. Robotic manipulators are one such application in the real world. It is a device used to manipulate the robotic arm without direct contact by an operation, which can be logically programmed. That works similar to the human arm. For performing rotational motion, number of joints are connected to the manipulator. Rotation motion can be performed through different degree of freedom. Though it has many benefits of reducing the human time it has some problems. One of the major problems is path planning. Path planning for an autonomous mobile robot is simply defined as finding the shortest distance between two points in an environment [1]. It is easy to find the shortest path for a robot with some computations. But, If the environment has some obstacles or let’s say it is a crowded environment then what will be the path of the robot? There is no assurance that the robot will not hit the obstacle with the robotic arm. So, we need to improve the computations and generate the path that the arm should not collide with the obstacle. There are many algorithms for finding an optimal path to avoid obstacles. A* algorithm is one of them which solves using heuristic values [2].

Some other methods are Intelligent control, C-Space method and Artificial Potential method [3]. C-Space incurs spatial alteration whereas, Intelligent

Control performs the manipulation to achieve the objective based on its own intelligence without support of humans. Also one more method based on dual neural networks based on Q-learning provides a solution [4]. Though these methods work fine, the efficient and traditional algorithm for trajectory planning is Genetic Algorithm.It completely avoids obstacles in its solution path. For avoiding complex computations and to make the process easier and simple we are using Genetic Algorithm here.

3 Proposed Solution

3.1 Establishing the Robotic Manipulator

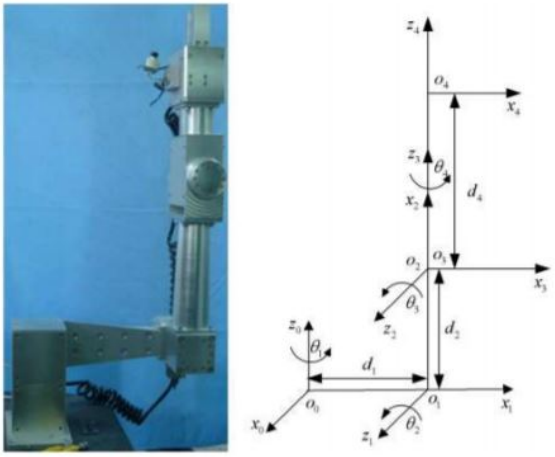

Figure 1.1: Robotic Manipulator

Figure 1.1 represents the robotic manipulator that is used. It has one fixed rod at a particular end and other joints are movable in space. The figure explains the system of coordinates established for the manipulator. It clearly states the joint angle movements for that arm based on the direction specified for each rod.

3.2 Deriving Kinematic Equations

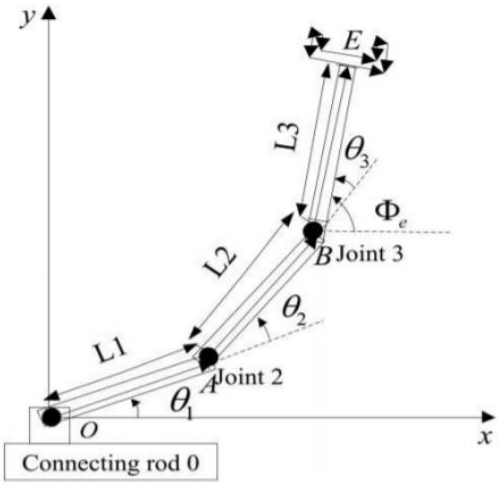

Figure 1.2: Forward Kinematics

This 4 degree of freedom manipulator is confined to only 3 degrees of freedom. This is because one of the rods end is fixed at a point and it can manipulate only the direction. This leads to the case where the remaining rods move freely and the movement of the other end actuator is free. Let L1, L2, L3 be the lengths of each rod.Refer Figure 1.2. The fixed point is O and the ends of rods are specified as A , B and E. The angle subtended by the first ,second and third rod is θ1, θ2 and θ3 respectively.

Based on these the coordinates of point E can be found, which is the end actuator and let it be expressed by (xe,ye).
Now ,

$$x_e = L_1 cos\theta_1 + L_2 cos(\theta_1 + \theta_2) + L_3 cos(\theta_1 + \theta_2 + \theta_3) \quad (1.1)$$

$$y_e = L_1 sin\theta_1 + L_2 sin(\theta_1 + \theta_2) + L_3 sin(\theta_1 + \theta_2 + \theta_3) \quad (1.2)$$

And also let

$$\phi_e = \theta_1 + \theta_2 + \theta_3 \quad (1.3)$$

Now the inverse kinematics to calculate the displacement in each joint based on the actuator position is done. Figure 1.3 is reference to give an idea about inverse kinematics.
Let us assume point B to have coordinates as (xw,yw)
Then ,

$$x_w = x_e - L_3 cos\phi_e \quad (1.4)$$

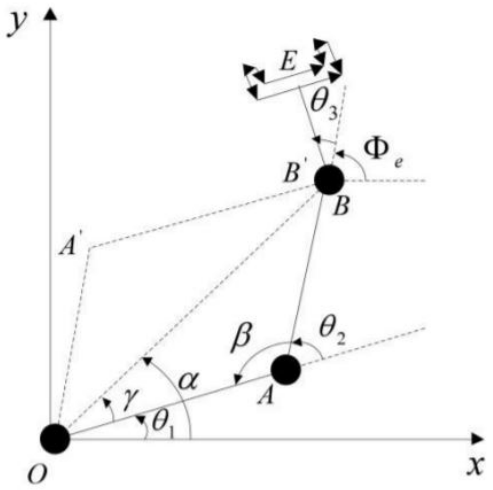

Figure 1.3: Inverse Kinematics

$$y_w = y_e - L_3 sin\phi_e \tag{1.5}$$

Therefore

$$\alpha = tan^{-}1(y_w/x_w) \tag{1.6}$$

Similarly , on solving

$$\beta = cos^{-}1(L_1^2 + L_2^2 - x_w^2 - y_w^2/2L_1L_2) \tag{1.7}$$

$$\gamma = cos^{-}1(x_w^2 + y_w^2 + L_1^2 - L_2^2/2L_1\sqrt{x_w^2 + y_w^2}) \tag{1.8}$$

Using these above equations the values of angle subtended are given as

$$\theta_1 = \alpha - \gamma, \theta_2 = \pi - \beta, \theta_3 = \phi_e - \theta_1 - \theta_2 \tag{1.9}$$

Based on these inverse kinematic equations now we can get one solution of possible angle series for the end actuator.

3.3 Genetic Algorithm

Genetic Algorithm follows a principle where the criteria is such that only the fittest of the given sample survives.The necessary parameters required to solve this problem are got as input and encoded as chromosomes.Then certain methods like selection,mutation and crossover are performed.The chromosomes generally represent data in an array.When a certain number of these chromosomes are collected these are called the population of individuals.

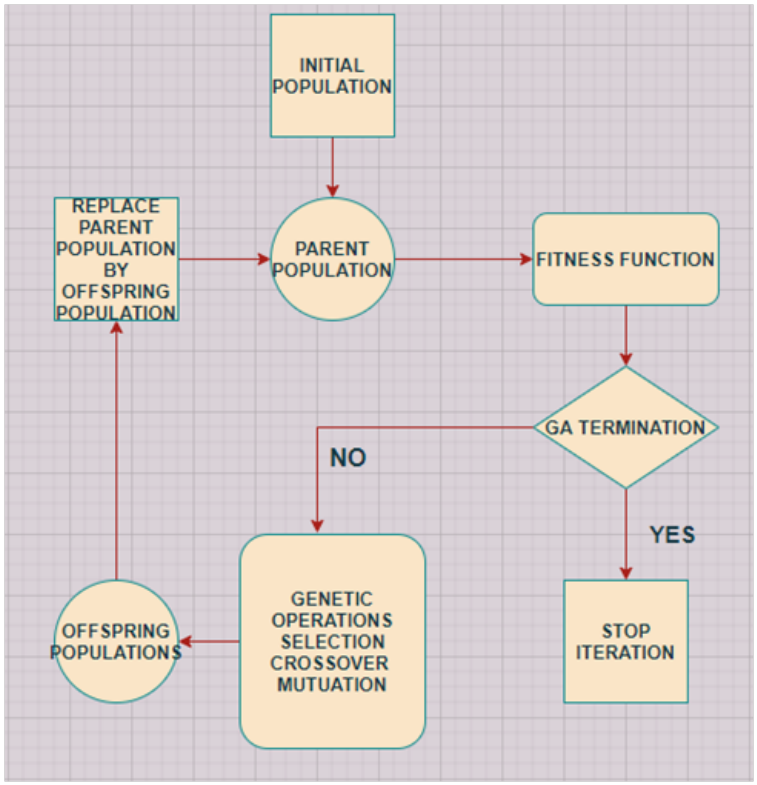

Figure 1.4: Flow of the algorithm

Encoding of the necessary inputs into chromosomes through real number encoding is done and then generate some random individuals. These random individuals must be present in the range of possible coordinates where the actuator can move.The following fitness function is defined which helps us evaluate the fitness of these individuals.

$$F_f = 1/(F_c + F_q) \tag{1.10}$$

Here Fc represents the length of the path and Fq represents the total joint distance. Presently let fitness function be defined as :

$$F = F_o b/F_f \tag{1.11}$$

Here Fo is 1 if it intersects with the obstacle coordinates or else its zero. Based on these kinds of functions only selection operator works. It takes only certain individuals based on value and passes it over to further operators. A roulette wheel is used for selection here. Then crossover operation takes over where there occurs a swap in genes of certain pairs and 2 new child are spawned. A percentage is defined for this to occur among individuals in the solution. Finally mutation operator is also applied where a certain gene of individuals is toggled so as to create a new spark and arrive at a sustained chromosome with possibility of obtaining high fitness value.

This whole process of performing operations repeats for a certain number of times or for the value which we specify as the number of generations. Basically fitness function here checks if it doesn't collide

with the obstacle coordinates and tries to keep minimum path length. The flowchart helps us understand the process.

For carrying out this experiment certain inputs are needed and some parameters are default values. The main parameters required here are basically the start and end point. The other parameters are like the coordinates of the obstacles in the environment ,which we have to avoid in the trajectory. Also the length of all the arms we use in the manipulator is needed.

The origin is the point where one end of arm is attached and immovable. Hence that alone won't move and be fixed. We have to set a value for the population size which is to generate a number of possible candidate solutions and a value around 40 is taken. Next the number of generations to be generated is also needed, which is as 250 for 2 link manipulators and 20 for 3 link manipulators. And also we need crossover percentage and mutation percentage for carrying out crossover and mutation operators. For crossover we take 0.8 percent and for mutation we take 0.05 percent. Basically these form the data set or the data needed for this proposed method. Hence based on these values and inputs we will be simulating the algorithm in a coordinate system and have attached the pictures of simulations with input stage and output stage. Also the input used for different scenarios is specified at last.

Chapter 2

Merits and Demerits of the Base Paper

1 Existing Solution

1.1 Neural Networks based Reinforcement Learning

Intelligent control is automatic control technology where control objective is achieved without human intervention which can drive intelligent machines automatically. The evolution of classic control theory was from modern control theory and it has entered into a theory of intelligence for systems in large scale.
Intelligent control theory mainly focuses on the development of model control theory in depth. Direction of the intelligent control system is developed by the control system. For the obstacle avoidance we need to create a perfect path for manipulator in it's environment without colliding the obstacles.

1.2 Real-time collision avoidance algorithm

This algorithm represents the collision avoidance of the robotic manipulator. This method does not care about the motion of other objects in work space. So this algorithm is efficient for implementation in real time. This can be performed by allowing the instantaneous velocity by constructing limitations on the motion of a manipulator. For linear inequality constraints potential collisions and joint limits are formulated. Interior point method is used to solve the convex optimization which is evolved from selection of the optimal velocity.

2 Merits

2.1 Neural Networks based Reinforcement Learning

With advancement of neural networks we can expect the exact required solution easily in this model. The path planning method for the mechanical arm includes intelligent control method, artificial potential field and c-space method. Main motive behind the c–space method is for Spatial alternation. We can get the complete solution for any hard environment. Optimal path will be determined for the obstacle avoidance after the reinforcement.

2.2 Real-time collision avoidance algorithm

This works on real time which simultaneously avoids obstacles and searches the goal point because it follows A* algorithm. The A* algorithm also finds the lowest path between the start and goal state, where changing from one state to another requires some cost.

3 Demerits

3.1 Neural Networks based Reinforcement Learning

One of the biggest disadvantages in neural networks is calculation and will be very complex to solve. Solving problems in the neural network usually consumes enormous amount of time to get the desired path.
Moreover this uses a huge amount for storing the state space. So this algorithm would not be convenient for the real time control.Due to the coupling relationship between joints and the limit of the structure, this algorithm faces complex calculations for the mobile robot.

3.2 Real-time collision avoidance algorithm

A* algorithm is used to achieve the obstacle avoidance. A* algorithm will be completed if the branching factor is finite.
A* search is highly dependent on the accuracy of the respective heuristic algorithm. This leads to further complexities for the algorithm to solve the problem.

Chapter 3

Source Code

```
#! /usr/bin/python3


import os

import numpy as np

import matplotlib.pyplot as plt

from plotter import Plotter

from genetic_algorithm import GeneticAlgorithm

import trajectory_generation as tg

from invkin import Arm

from three_link import Arm3Link




class ProblemParams:

        """A struct like class to store problem parameters"""

        def __init__(self, description=None, link_lengths=None, start_cood=None, end_cood=None,
obs_coods=None):

                self.description = description

                self.link_lengths = link_lengths

                self.start_cood = start_cood

                self.end_cood = end_cood

                self.obs_coods = obs_coods



preset_params = [

        ProblemParams("Two links, two obstacles", [4,4], [6.5,2.8], [-3.3,5.1], [[-0,5.3],[5.4,3.2]]),
```

```
def select_preset_param(preset_params):
        # Clear terminal on win, linux
        os.system('cls' if os.name == 'nt' else 'clear')
        print("Robotic arm trajectory using Genetic Algorithm\n")
        print("Select a preset problem:")
        for i,param in enumerate(preset_params):
                print("  {0}: {1}".format(i+1, param.description))
        print("\n  q: Quit")

        choice = None
        while choice == None:
                choice = input("Choice: ")
                if choice == 'q':
                        return 'q'
                elif choice.isdigit() and 1 <= int(choice) <= len(preset_params):
                        choice = int(choice)
                else:
                        print("Invalid input!")
                        choice = None
        return choice-1


def select_link_lengths():
```

```
# Clear terminal on win, linux
os.system('cls' if os.name == 'nt' else 'clear')
print("Robotic arm trajectory using Genetic Algorithm\n")

links_num = None
link_lengths = None

while links_num == None:
        links_num = input("Number of links:  ")
        if links_num == 'q':
                return 'q'
        elif links_num.isdigit():
                links_num = int(links_num)
                if links_num < 2 or links_num > 3:
                        print("Invalid input! Enter two or three")
                        links_num = None
        else:
                print("Invalid input!")
                links_num = None

link_lengths = []
for i in range(links_num):
        link_length = None

        while link_length == None:
```

```
                link_length = input("Length of link {0}: ".format(i+1))
                if link_length == 'q':
                        return 'q'
                elif link_length.isdigit():
                        link_length = int(link_length)
                        link_lengths.append(link_length)
                else:
                        print("Invalid input!")
                        link_length = None

        return link_lengths


while True:


        plotter = Plotter()

        link_lengths = None
        start_cood = None
        end_cood = None
        obs_coods = None

        ga_genr_2 = 250
        ga_genr_3 = 20
```

```
ga_genr = 10

ga_pop_sz = 40

ga_mut_ratio = 0.05

ga_xov_ratio = 0.30

ga_mu_2 = [0.5,0.5]

ga_mu_3 = [0.4,0.3,0.3]

ga_mu = None

ga_eps = 0.1


param_method = select_param_method()


if param_method == 'q':
        break


elif param_method == 1:
        param_idx = select_preset_param(preset_params)


        if param_idx == 'q':
                break


        else:
                link_lengths = preset_params[param_idx].link_lengths

                start_cood = preset_params[param_idx].start_cood

                end_cood = preset_params[param_idx].end_cood

                obs_coods = preset_params[param_idx].obs_coods
```

```
                plotter.link_lengths = link_lengths
                plotter.start_cood = start_cood
                plotter.end_cood = end_cood
                plotter.obs_coods = obs_coods

                plotter.static_show()


elif param_method == 2:
        link_lengths = select_link_lengths()

        if link_lengths == 'q':
                break

        plotter.link_lengths = link_lengths

        plotter.picker_show()

        start_cood = plotter.start_cood
        end_cood = plotter.end_cood
        obs_coods = plotter.obs_coods

        if len(obs_coods) < 2:
                print("Select atleast two obstacles!")
```

```
                continue

        os.system('cls' if os.name == 'nt' else 'clear')
        print("Robotic arm trajectory using Genetic Algorithm\n")
        print("Running genetic algorithm... ")

        ga_mu = ga_mu_2 if len(link_lengths) == 2 else ga_mu_3
        ga_genr = ga_genr_2 if len(link_lengths) == 2 else ga_genr_3

        ga = GeneticAlgorithm(link_lengths, start_cood, end_cood, obs_coods, tg.fitness_population,
ga_mu, ga_eps, ga_pop_sz, ga_mut_ratio, ga_xov_ratio, ga_genr)
        output_chr = ga.run()
        print("Done")

        output_path = tg.chrome_traj(output_chr, start_cood, end_cood)

        arm = Arm(link_lengths) if len(link_lengths) == 2 else Arm3Link(np.array(link_lengths))
        link_angles_series = np.degrees(arm.time_series(output_path))

        plotter.transition_show(link_angles_series)

        usr_input = input("\nTry again? [y/n] ")
        if usr_input == 'y':
                continue
        else:
                break
```

1 Genetic Algorithm Implementation

```
import numpy as np

np.random.seed(1)



class GeneticAlgorithm:


    def __init__(self, link_lengths, start_cood, end_cood, obs_coods, fitness, mu=[0.4,0.2], epsilon=0.1,
population_size=120, mutation_percent=0.05, crossover_percent=0.30, generations=500):

        self.L1 = link_lengths[0]

        self.L2 = link_lengths[1]


        self.start_cood = start_cood

        self.end_cood = end_cood

        self.obs_coods = obs_coods

        self.fitness = fitness # fitenss function, takes population as input

        self.mu = mu

        self.epsilon = epsilon


        self.fitness_params = (link_lengths, start_cood, end_cood, obs_coods, epsilon, mu)


        self.L = 12

        self.x_min = 0

        self.x_max = pow(2,self.L)-1

        self.y_min = 0

        self.y_max = pow(2,self.L)-1
```

```
    def chromosome_to_points(self, chromosome):
        return (chromosome)*(2*self.R1)/(2**self.L-1) - self.R1


    def chromosome_init(self):
        chromosome = np.zeros((self.population_size,2*self.k))

        centre_cood = [2**(self.L-1)-1, 2**(self.L-1)-1]
        distance_max = (self.x_max+1)/2
        distance_min = self.R2/self.R1*distance_max

        for i,chrom in enumerate(chromosome):
            chrom_valid = False

            while chrom_valid == False:
                random_chrom_x = np.random.randint(2**self.L,size=[self.k])
                random_chrom_y = np.random.randint(2**(self.L-1),size=[self.k]) + 2**(self.L-1)
                random_chrom = np.column_stack((random_chrom_x,random_chrom_y))

                distance = np.sqrt((random_chrom[:,0]-centre_cood[0])**2+(random_chrom[:,1]-
centre_cood[1])**2)

                if np.all(distance_min < distance) and np.all(distance_max > distance):
                    chrom_valid = True
                    chromosome[i] = np.ravel(random_chrom)
```

```
    return chromosome



def fitness_mod(self,chromosome):
    fitness_row, _ = self.fitness(self.chromosome_to_points(chromosome), *self.fitness_params)
    for i,v in enumerate(fitness_row):
        if np.isnan(v):
            fitness_row[i] = 0
        else:
            fitness_row[i] = abs(v)
    return fitness_row



def run(self):
    chromosome = self.chromosome_init()
    fitness_row = self.fitness_mod(chromosome)
    r=1
    for genr in range(self.generations):


        print("New generation: "+str(r), end="\n", flush=True)
        r=r+1
        roulette_wheel_cdf = np.cumsum(fitness_row/np.sum(fitness_row))
        crossover_point = np.random.randint(self.k-1) if self.k != 1 else 0
        index = np.zeros((2))
```

```
        new_chromosome = np.zeros((self.population_size,2*self.k))

        for i in range(int(self.population_size/2)):

            a = np.random.rand(2)
            index = np.searchsorted(roulette_wheel_cdf, a)
            parent = np.array([chromosome[index[0]], chromosome[index[1]]])

            if np.random.rand() < self.crossover_percent:
                new_chromosome[2*i+0,0:2*crossover_point+1] = parent[0,0:2*crossover_point+1]
                new_chromosome[2*i+0,2*crossover_point+1:2*self.k] =
parent[1,2*crossover_point+1:2*self.k]
                new_chromosome[2*i+1,0:2*crossover_point+1] = parent[1,0:2*crossover_point+1]
                new_chromosome[2*i+1,2*crossover_point+1:2*self.k] =
parent[0,2*crossover_point+1:2*self.k]
            else:
                new_chromosome[2*i+0] = parent[0]
                new_chromosome[2*i+1] = parent[1]

        for i in range(self.population_size):
            if (np.random.rand() < self.mutation_percent):
                p = np.random.randint(12*2*self.k)
                q = int(np.floor(p/12))
                p = int(p - 12*q)
```

```
            binary = list(np.binary_repr(int(new_chromosome[i,q]),12))
            if (binary[p] == '0'):
               binary[p] = '1'
            elif (binary[p] == '1'):
               binary[p] = '0'
            binary_string = "".join(binary)
            new_chromosome[i,q] = int(binary_string,2)

      chromosome = new_chromosome
      fitness_row = self.fitness_mod(chromosome)
      self.fitness_stats.append(max(fitness_row))

print()

fitness_row = self.fitness_mod(chromosome)
max_idx = np.argmax(fitness_row)
return (self.chromosome_to_points(chromosome))[max_idx]
```

```
import numpy as np
import scipy.interpolate as sc
import matplotlib.pyplot as plt
import three_link
import invkin
import timeit



def generate_trajectories(formatted_population, start, end, fitness_calculated):
    shape = np.shape(formatted_population)
    left_end, right_end = start, start
    if start[0] < end[0]:
        right_end = end
    else:
        left_end = end

    population_trajectories = [False for g in range(shape[0])]
    trajectory_points = np.zeros([shape[0], shape[1] + 2, shape[2]])
    for i in range(shape[0]):
        if fitness_calculated[i]:
            continue
        ch_with_start = np.insert(formatted_population[i, :, :], 0, left_end, axis=0)
        chrome_all_pts = np.insert(ch_with_start, (shape[1] + 1), right_end, axis=0)
```

```
        population_trajectories[i] = sc.PchipInterpolator(chrome_all_pts[:, 0], chrome_all_pts[:, 1])
        trajectory_points[i, :, :] = chrome_all_pts
    return trajectory_points, population_trajectories


def chrome_traj(chrome, start, end):
    sorted_chrome = format(chrome)
    sh = np.shape(sorted_chrome)
    left_end, right_end = start, start
    if start[0] < end[0]:
        right_end = end
    else:
        left_end = end
    K = sh[1]
    ch_with_start = np.insert(sorted_chrome, 0, left_end, axis=1)
    chrome_all_pts = np.insert(ch_with_start, (K + 1), right_end, axis=1)
    ch_x, ch_y = chrome_all_pts[:, :, 0][0], chrome_all_pts[:, :, 1][0]
    trajectory = sc.PchipInterpolator(ch_x, ch_y)
    traj_points = path_points(trajectory, 0.1, start, end)
    return traj_points


def format(population) -> object:
    shape = np.shape(population)
    if len(shape) == 1:
```

```
        P = 1
        K = int(shape[0]/2)
    elif len(shape) == 2:
        P = shape[0]
        K = int(shape[1]/2)
    formatted_population = np.zeros([P, K, 2])
    for i in range(P):
        if P == 1:
            chrome = np.reshape(population, [K, 2])
        else:
            chrome = np.reshape(population[i, :], [K, 2])
        chrome_sorted = chrome[chrome[:, 0].argsort()].transpose()
        formatted_population[i, :, :] = chrome_sorted.transpose()
    return formatted_population


def check_point_validity(formatted_population, link_len, start, end) -> list:
    shape = np.shape(formatted_population)
    left_end, right_end = start, start
    if start[0] < end[0]:
        right_end = end
    else:
        left_end = end
    validity = []
    for i in range(shape[0]):
```

```
        r = np.linalg.norm(formatted_population[i, :, :], axis=1)
        if np.any(formatted_population[i, :, 0] < left_end[0]):
            validity.append(False)
        elif np.any(formatted_population[i, :, 0] > right_end[0]):
            validity.append(False)
        elif np.all(r > link_len[0]):
            if np.all(r < (sum(link_len))):
                if np.all(formatted_population[i,:,1] > 0):
                    validity.append(True)
                else:
                    validity.append(False)
            else:
                validity.append(False)
        else:
            validity.append(False)

    return validity



def check_trajectory_validity(trajectory, obstacles):
    obstacles = np.array(obstacles)

    if np.any(trajectory(obstacles[:,0]) > obstacles[:,1]):
        validity = False
    else:
```

```
        validity = True
    return validity



def path_points(y, epsilon, start, end):
    pt_x = [start[0]]
    pt_y = [start[1]]
    der = y.derivative()

    x = start[0]

    if start[0] < end[0]:
        while x < end[0]:
            del_x = epsilon / np.sqrt(der(x) ** 2 + 1)
            if (x + del_x) < end[0]:
                pt_x.append(x + del_x)
                pt_y.append(y(x + del_x))
                x += del_x
            else:
                pt_x.append(end[0])
                pt_y.append(end[1])
                break
    else:  # end point on left side
        while x > end[0]:
            del_x = epsilon / np.sqrt(der(x) ** 2 + 1)
```

```
        if (x - del_x) > end[0]:
            pt_x.append(x - del_x)
            pt_y.append(y(x - del_x))
            x -= del_x
        else:
            pt_x.append(end[0])
            pt_y.append(end[1])
            break

    points = np.zeros([2, len(pt_x)])
    points[0, :] = np.array(pt_x)
    points[1, :] = np.array(pt_y)

    return points.transpose()


def fitness_population(population, link_len, start_pt, end_pt, obstacles, epsilon, mu, Single=False):
    """
    Envelope function for complete fitness calculation
    Order of operations:
    1. point checking       (set fitness to zero for invalid)
    2. path interpolation
    3. path discretization
    4. reverse kinematics on path
    5. Path checking        (check order here)
```

```
5. fitness calculation
"""
if len(link_len) == 3:
    arm1 = three_link.Arm3Link(link_len)
elif len(link_len) == 2:
    arm1 = invkin.Arm(link_len)

if Single == True:
    pop_size = 1
else:
    pop_size = np.shape(population)[0]

cost_pop = [np.inf for i in range(pop_size)]
fitness_calculated = [False for i in range(pop_size)]

formatted_pop = format(population)
pt_validity = check_point_validity(formatted_pop, link_len, start_pt, end_pt)
for i in range(pop_size):
    if pt_validity[i] == False:
        cost_pop[i] = np.inf
        fitness_calculated[i] = True

points, trajectories = generate_trajectories(formatted_pop, start_pt, end_pt, fitness_calculated)
traj_points = None
for i in range(pop_size):
```

```
        if fitness_calculated[i] == False:
            traj_points = path_points(trajectories[i], epsilon, start_pt, end_pt)
            theta = np.array(arm1.time_series(traj_points))
            validity = check_trajectory_validity(trajectories[i], obstacles)
            if validity == False:
                cost_pop[i] = np.inf
            else:
                cost_pop[i] = fitness_chrome(theta, mu)
            fitness_calculated[i] = True

    fitness_pop = 1/np.array(cost_pop)
    return np.array(fitness_pop), traj_points

def fitness_chrome(theta, mu):

    theta = theta.T
    div = np.shape(theta)[1]
    theta_i = theta[:, 0:div - 2]
    theta_j = theta[:, 1:div - 1]
    del_theta = abs(theta_j - theta_i)
    fitness = 0
    for i in range(div - 2):
        for j in range(len(mu)):
            fitness += mu[j] * theta[j, i]
    return fitness
```

```
#! /usr/bin/python3




import math


import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button, RadioButtons




class Plotter():


        def __init__(self):
                self.link_lengths = None
                self.link_angles = None
                self.start_cood = None
                self.end_cood = None
                self.obs_coods = None



        def plot_grids(self, ax):
                # X and Y axis
                ax.axhline(c="0.9")
```

```
                ax.axvline(c="0.9")


                # Two circles
                circle1 = plt.Circle((0,0), self.link_lengths[0], color='0.9', fill=False)
                circle2 = plt.Circle((0,0), sum(self.link_lengths), color='0.9', fill=False)
                ax.add_artist(circle1)
                ax.add_artist(circle2)



        def plot_terminals(self, ax):
                ax.plot(*self.start_cood, 'bo')
                ax.plot(*self.end_cood, 'go')

                ax.annotate(r'$P_s$', xy=self.start_cood, xytext=(5,-10), textcoords='offset points')
                ax.annotate(r'$P_t$', xy=self.end_cood, xytext=(5,-10), textcoords='offset points')



        def plot_obstacles(self, ax):
                for i,cood in enumerate(self.obs_coods):
                        ax.plot(*cood, 'r^')
                        ax.annotate(r'$P_{o'+str(i)+r'}$', xy=cood, xytext=(5,-10), textcoords='offset
points')


        def plot_start_point(self, ax):
                point = ax.plot(*self.start_cood, 'bo')
```

```
            label = ax.annotate(r'$P_s$', xy=self.start_cood, xytext=(5,-10), textcoords='offset
points')
            return point, label


    def plot_end_point(self, ax):
            point = ax.plot(*self.end_cood, 'go')
            label = ax.annotate(r'$P_t$', xy=self.end_cood, xytext=(5,-10), textcoords='offset
points')
            return point, label


    def plot_obs_point(self, ax, cood=None, label_idx=""):
            if cood == None:
                    cood = self.obs_coods[-1]
            point = ax.plot(*cood, 'r^')
            label = ax.annotate(r'$P_{o'+str(label_idx)+r'}$', xy=cood, xytext=(5,-10),
textcoords='offset points')
            return point, label


    def plot_links(self, ax, *args):
            if args:
                    coods_x, coods_y = args[0]
            else:
                    coods_x, coods_y = self.get_coods_from_link_angles()
```

```
        [link_object] = ax.plot(coods_x, coods_y, '-mo')
        return link_object


def plot_links_by_time(self, ax, coods_series, t):
        steps = len(coods_series[0])
        coods_x = coods_series[0][int(t*(steps-1))]
        coods_y = coods_series[1][int(t*(steps-1))]

        return self.plot_links(ax, [coods_x, coods_y])


def plot_update_links_by_time(self, link, coods_series, t):
        steps = len(coods_series[0])
        coods_x = coods_series[0][int(t*(steps-1))]
        coods_y = coods_series[1][int(t*(steps-1))]

        link.set_xdata(coods_x)
        link.set_ydata(coods_y)


def plot_set_lims(self, ax):
        ax.axis('scaled')
        axis_limit = 1.2*sum(self.link_lengths)
        ax.set_xlim(-axis_limit, axis_limit)
```

```
        ax.set_ylim(-axis_limit, axis_limit)


def plot_end_path(self, ax, xs, ys):
        ax.plot(xs, ys, '--', c="y")


def plot_joint_path(self, ax, xs, ys):
        ax.plot(xs, ys, '--', c="0.8")


def static_plot(self, ax):
        self.plot_grids(ax)
        self.plot_terminals(ax)
        self.plot_obstacles(ax)
        self.plot_set_lims(ax)
        self.plot_set_lims(ax)


def static_show(self):
        fig,ax = plt.subplots()
        self.static_plot(ax)
        plt.show()

def static_show(self):
```

```
        fig,ax = plt.subplots()
        self.static_plot(ax)
        plt.show()


def transition_plot_base(self, ax, coods_series):
        self.plot_grids(ax)
        self.plot_terminals(ax)
        self.plot_obstacles(ax)
        self.plot_set_lims(ax)

        coods_x_series, coods_y_series = coods_series
        self.plot_end_path(ax, coods_x_series.T[-1], coods_y_series.T[-1])
        for xs, ys in zip(coods_x_series.T[:-1], coods_y_series.T[:-1]):
                self.plot_joint_path(ax, xs, ys)


def transition_show(self, link_angles_series):
        fig = plt.figure()

        ax_main = fig.add_axes([0.1,0.2,0.8,0.7])
        ax_slider = fig.add_axes([0.1,0.03,0.8,0.03])

        coods_series = self.get_coods_series_from_link_angles_series(link_angles_series)
```

```
        self.transition_plot_base(ax_main, coods_series)
        link = self.plot_links_by_time(ax_main, coods_series, 0)


        slider = Slider(ax_slider, "Time", 0, 1, valinit=0)
        def on_slider_upd(val):
            self.plot_update_links_by_time(link, coods_series, val)
            fig.canvas.draw()
        slider.on_changed(on_slider_upd)

        plt.show()


def picker_plot_base(self, ax):
        self.plot_grids(ax)
        self.plot_set_lims(ax)


def picker_show(self):
        fig,ax = plt.subplots()
        self.picker_plot_base(ax)

        self.start_cood = None
        self.end_cood = None
        self.obs_coods = []
```

```
ax_title = ax.set_title("Pick start point")


start_point = None

end_point = None

obs_points = []


start_label = None

end_label = None

obs_labels = []


def on_click(event):


        if event.button != 1 or event.xdata == None or event.ydata == None:

                return


        nonlocal ax_title


        nonlocal start_point

        nonlocal end_point

        nonlocal obs_points


        nonlocal start_label

        nonlocal end_label

        nonlocal obs_labels
```

```
                cood = [event.xdata, event.ydata]

                if self.start_cood == None:
                        self.start_cood = cood
                        start_point, start_label = self.plot_start_point(ax)
                        ax_title.set_text("Pick end point")

                elif self.end_cood == None:
                        self.end_cood = cood
                        end_point, end_label = self.plot_end_point(ax)
                        ax_title.set_text("Pick obstacle points")

                else:
                        self.obs_coods.append(cood)
                        obs_point, obs_label = self.plot_obs_point(ax,
label_idx=len(self.obs_coods)-1)
                        obs_points.append(obs_point)
                        obs_labels.append(obs_labels)

                fig.canvas.draw()

        cid = fig.canvas.mpl_connect("button_release_event", on_click)
        plt.show()
        fig.canvas.mpl_disconnect(cid)
```

```
def get_coods_from_link_angles(self, *args):
        coods_y = [0]
        coods_x = [0]

        if args:
                link_angles = args[0]
        else:
                link_angles = self.link_angles

        angle = 0
        for l,a in zip(self.link_lengths, link_angles):
                angle += a
                cood_y = coods_y[-1] + l*math.sin(math.radians(angle))
                cood_x = coods_x[-1] + l*math.cos(math.radians(angle))

                coods_y.append(cood_y)
                coods_x.append(cood_x)

        return coods_x, coods_y


def get_coods_series_from_link_angles_series(self, link_angles_series):
        coods_y_series = []
```

```
coods_x_series = []

for link_angles in link_angles_series:
        coods_x, coods_y = self.get_coods_from_link_angles(link_angles)
        coods_x_series.append(coods_x)
        coods_y_series.append(coods_y)

coods_x_series = np.array(coods_x_series)
coods_y_series = np.array(coods_y_series)

return coods_x_series, coods_y_series
```

```
import numpy as np
import scipy.optimize


class Arm3Link:

    def __init__(self,Len=None):

        self.angles = [.3, .3, 0]
        self.default = np.array([np.pi/4, np.pi/4, np.pi/4])
        self.Len = np.array([1, 1, 1]) if Len is None else Len

    def inv_kin(self, xy):

        def distance_to_default(q, *args):
            weight = [1, 1, 1.3]
            return np.sqrt(np.sum([(qi - q0i)**2 * wi for qi, q0i, wi in zip(q, self.default, weight)]))


        def x_constraint(q, xy):
            x = (self.Len[0]*np.cos(q[0]) + self.Len[1]*np.cos(q[0]+q[1]) +self.Len[2]*np.cos(np.sum(q))) -
xy[0]
            return x

        def y_constraint(q, xy):
            y = (self.Len[0]*np.sin(q[0]) + self.Len[1]*np.sin(q[0]+q[1]) +self.Len[2]*np.sin(np.sum(q))) - xy[1]
```

```
            return y


        return
scipy.optimize.fmin_slsqp(func=distance_to_default,x0=self.angles,eqcons=[x_constraint,y_constraint],a
rgs=(xy,),iprint=0)


    def time_series(self,coordinate_series):
        angle_series=[]
        self.angles = self.inv_kin(coordinate_series[0])

        for i in range(len(coordinate_series)):
            angle_series.append(self.inv_kin(coordinate_series[i]))
            self.angles = self.inv_kin(coordinate_series[i])

        return angle_series
```

```
import math
import numpy as np

def div(a,b):
    return a/b if b is not 0 else math.inf if a>0 else -math.inf

class Arm :
  def __init__(self,arm_lens):
    self.arm_lens = arm_lens

  def get_position(self,angles):
    x = self.arm_lens[0]*np.cos(angles[0]) + self.arm_lens[1]* np.cos(angles[0]+angles[1])
    y = self.arm_lens[0]*np.sin(angles[0]) + self.arm_lens[1]* np.sin(angles[0]+angles[1])
    return x,y

  def inv_kin(self,final_coords):
    D = div( (final_coords[0]**2 + final_coords[1]**2 - self.arm_lens[0]**2 - self.arm_lens[1]**2),
2*self.arm_lens[0]*self.arm_lens[1] )
    angles = [0,0]
    temp = math.atan2( (1-(D**2))**(1/2),D  )
    angles[1] = temp if temp>=0 else -temp
    angles[0] = math.atan2(final_coords[1],final_coords[0]) - math.atan2(
self.arm_lens[1]*np.sin(angles[1]), self.arm_lens[0]+ self.arm_lens[1]*np.cos(angles[1]) )
    return np.array(angles)

  def time_series(self,coordinate_series):
```

```
        angle_series=[]
        for i in range(len(coordinate_series)):
            angle_series.append(self.inv_kin(coordinate_series[i]))
        return angle_series
def test():

    initialangles = [.349 , .349]
    lengths = [4,4]
    initialPos = [0,0]
    Arm1 = Arm(lengths)

    series = [[2,2],[3,3],[4,4]]
    print(Arm1.time_series(series))
```

Chapter 4

Snapshots

1 Experiment 1

For the first experiment 2 degrees of freedom mechanical arm with 2 obstacles is taken. The input is as follows.
Length of arms - L1 = 4 , L2 = 4
Starting coordinates : (6.5,2.8)
Ending coordinates : (-3.3,5.1)
Obstacle coordinates : (0,5.3) , (5.4,3.2)
Number of generations : 250
Population size : 40
Mutation percent : 0.05
Crossover percent : 0.8
The Figure 4.1 shows the sample environment with inputs.

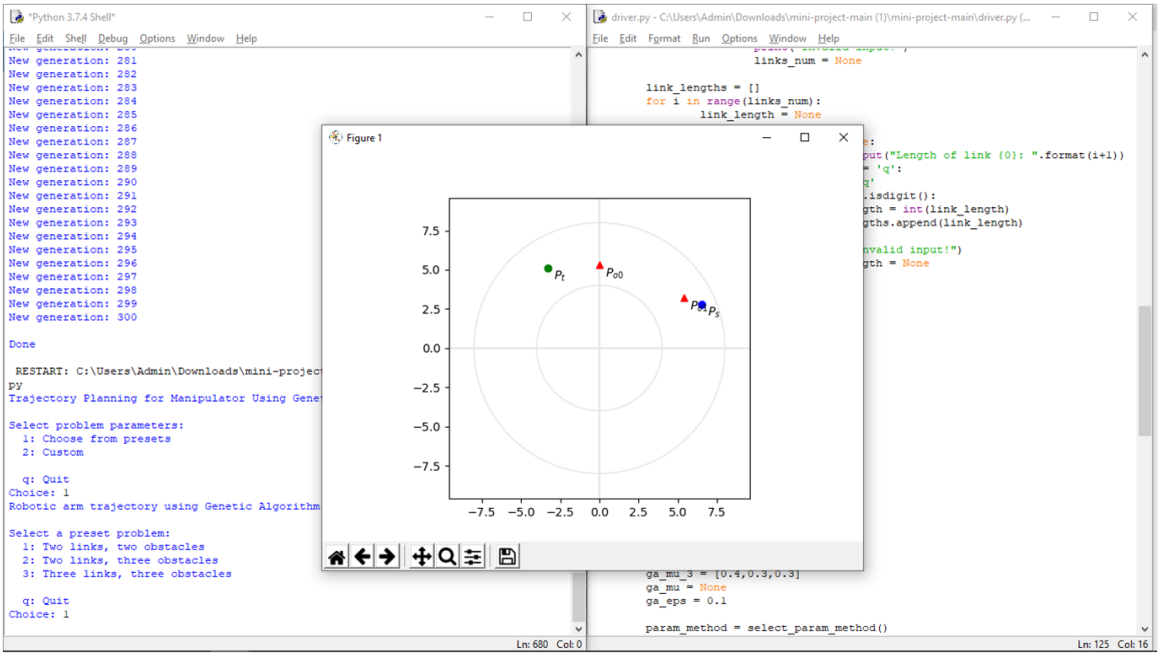

Figure 4.1: Input 1

The Figure 4.2 shows the output produced by the Genetic Algorithm. This is the output obtained after generating 250 generations and also we can see that the arm avoids collision in the given trajectory.
The trajectory for both arms actuators are produced, the yellow dotted line represents the trajectory and the blue dotted lines represent the path for the first arm. Points in red are the obstacles and the points in green are starting point and ending point.

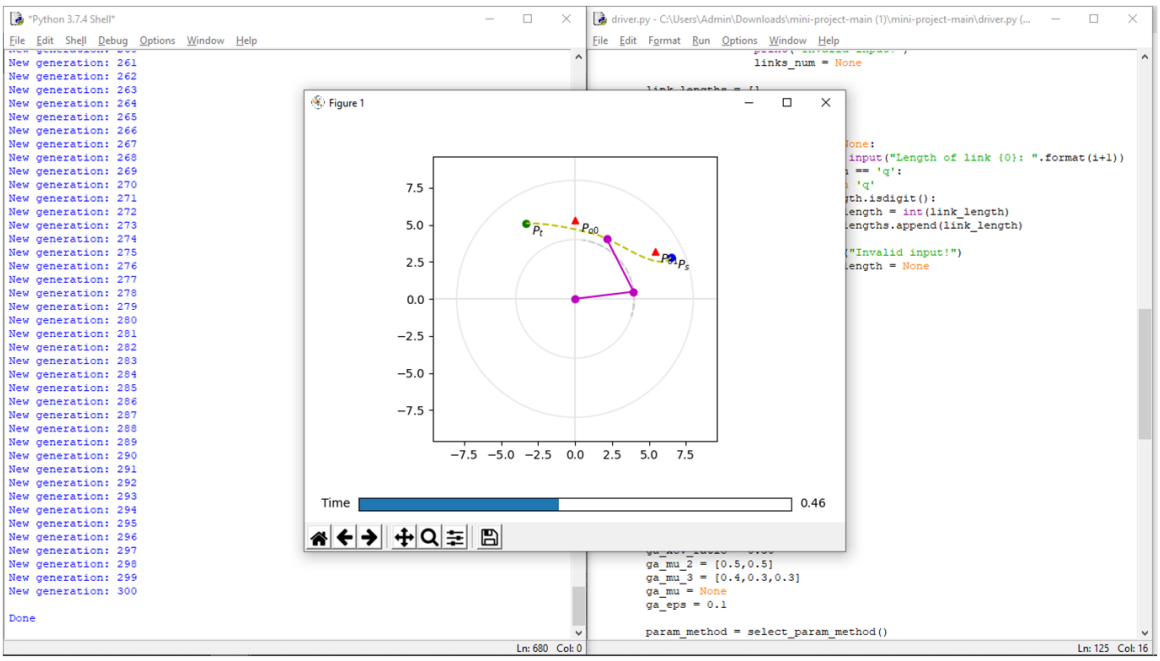

Figure 4.2: Output 1

2 Experiment 2

For the second experiment 2 degrees of freedom mechanical arm with 3 obstacles is taken. The input is as follows.
Length of arms - L1 = 4 , L2 = 4
Starting coordinates : (7,2.6)
Ending coordinates : (-5,2.8)
Obstacle coordinates : (5,3.8) , (1.7,5.8) , (-2.5,5.8)
Number of generations : 250
Population size : 40
Mutation percent : 0.05
Crossover percent : 0.8
The Figure 4.3 shows the sample environment with inputs.

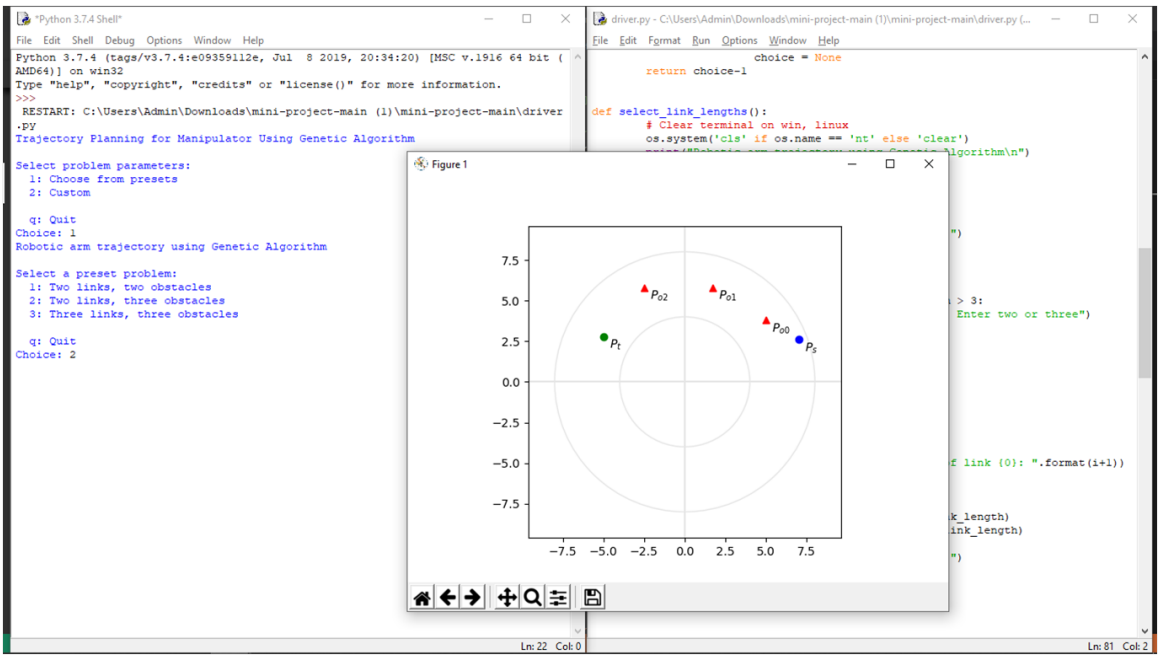

Figure 4.3: Input 2

The Figure 4.4 shows the output produced by the Genetic Algorithm. This also shows a path without colliding with obstacles.

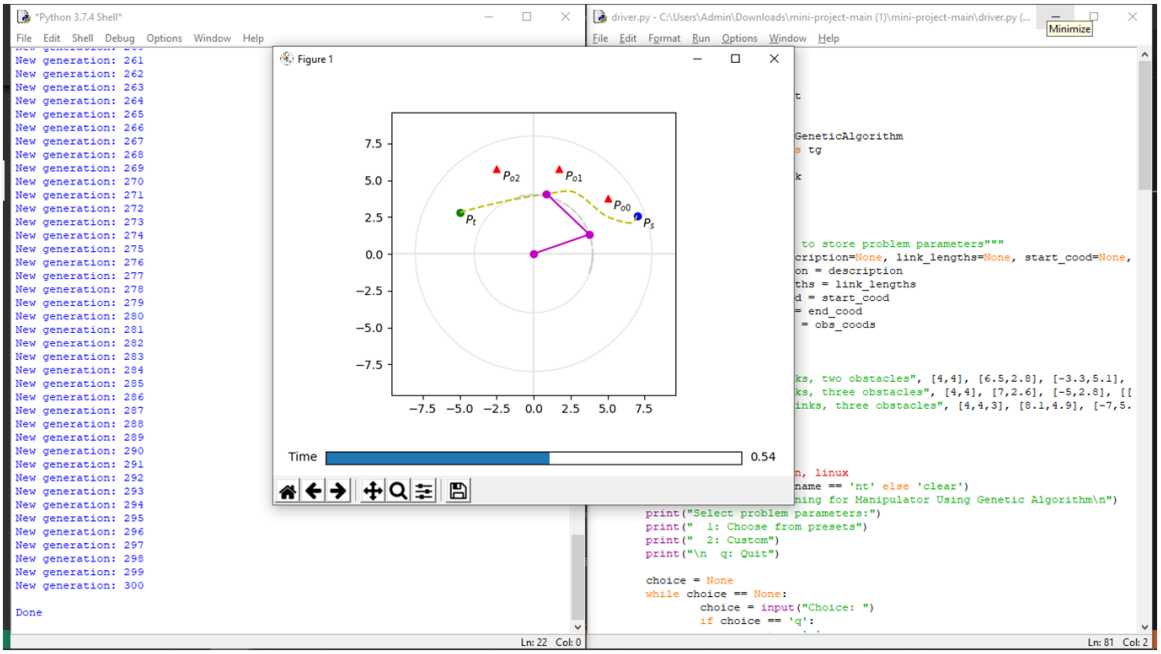

Figure 4.4: Output 2

3 Experiment 3

For the third experiment 3 degrees of freedom mechanical arm with 2 obstacles is taken.
Number of generations : 20
Population size : 40
Mutation percent : 0.05
Crossover percent : 0.3
The Figure 4.5 shows the sample environment with inputs.

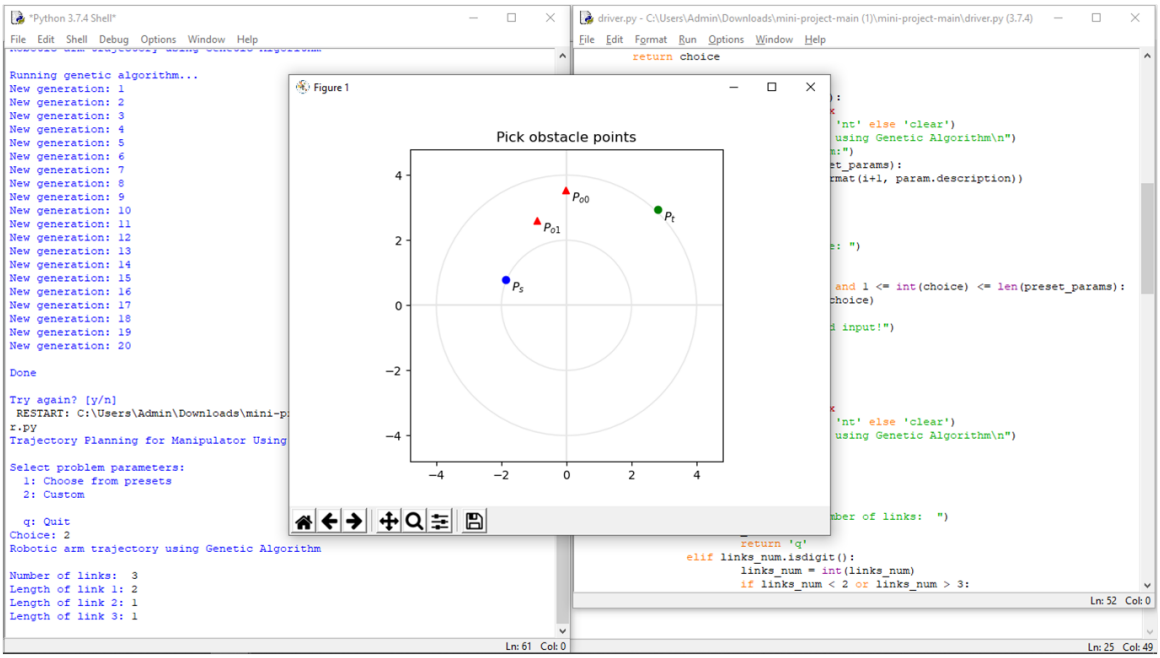

Figure 4.5: Input 3

The Figure 4.6 shows the output produced by the Genetic Algorithm. This also shows a path without colliding with obstacles.

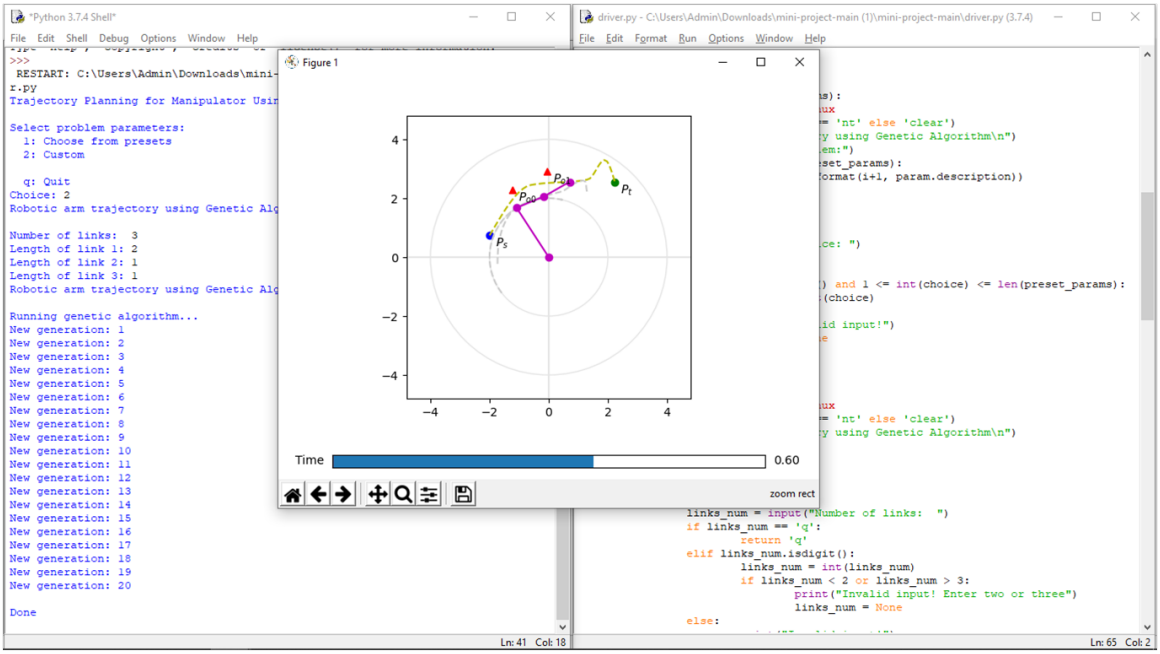

Figure 4.6: Output 3

4 Experiment 4

For the final experiment 3 degrees of freedom mechanical arm with 4 obstacles is taken. The following is the details regarding the input we provide.

Length of arms - L1 = 4 , L2 = 4 , L3 = 3

Starting coordinates : (8.1,4.9)

Ending coordinates : (-7,5.8)

Obstacle coordinates : (0,9) , (-1,8) , (1,8) , (0,7)

Number of generations : 20

Population size : 40

Mutation percent : 0.05

Crossover percent : 0.3

The Figure 4.7 shows the sample environment with inputs.

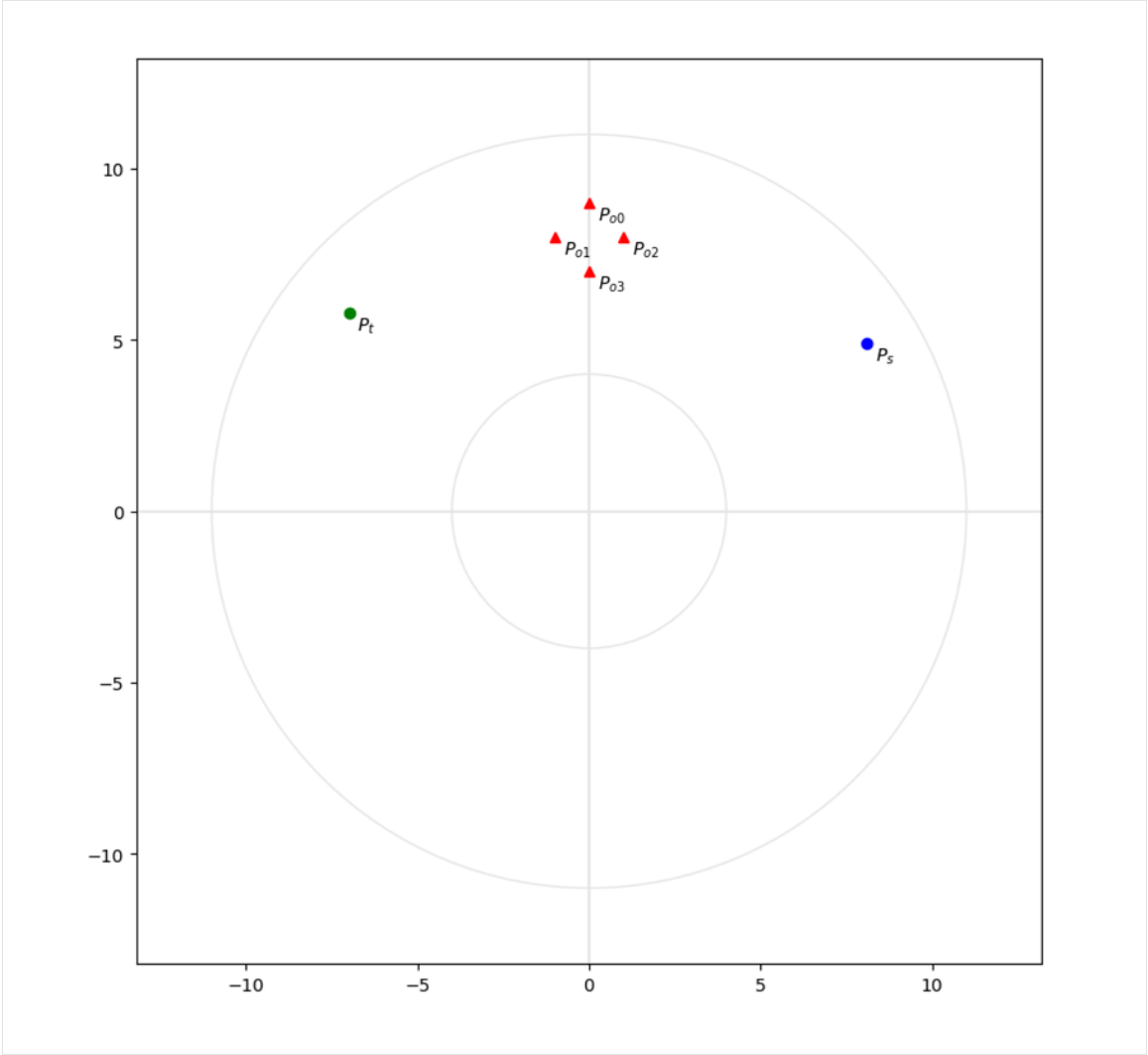

Figure 4.7: Input 4

The Figure 4.8 shows the output produced by the Genetic Algorithm. This also shows a path without colliding with obstacles.
The trajectory for all 3 arms actuators are produced, the yellow dotted line represents the trajectory and the blue dotted lines represent the path for the first arm and the white dotted line shows path of second arm. Points in red are the obstacles and the points in green are starting point and ending point.

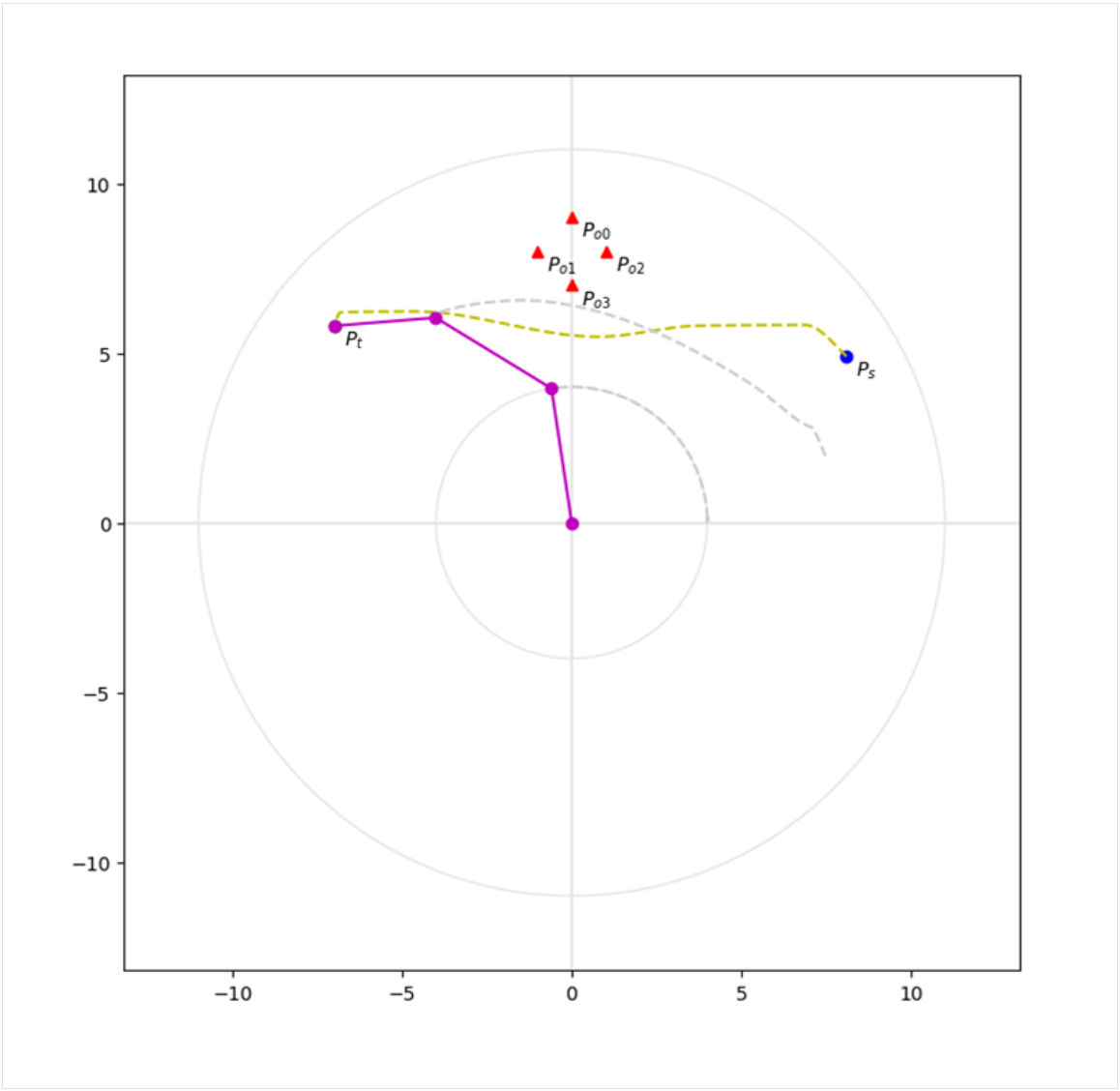

Figure 4.8: Output 4

Chapter 5

Conclusion and Future Plans

In this paper, the usefulness and effectiveness of Genetic Algorithm for obstacle avoidance is explored. With the help of Genetic Algorithm, output was generated and it was an optimal solution.

A mechanical arm is set up and explained the coordinate axes of how the orientation of the arm is present was shown. Then Kinematic Analysis on the arm was done and had got the equations of the actuators. Next the genetic algorithm was applied on it and had gotten the result of trajectory. So now from the above output screens we have verified the working of Genetic Algorithm.

Further this present solution is only for static based inputs where the obstacles are stationary and present in the same place. So maybe the problem statement can be extended where the number of obstacles are also increased and the obstacles are in moving state. Also some more constraints like including more manipulators in the system can also be taken.

So basically a combination of Genetic Algorithm with Reinforcement Learning could be used to solve the above mentioned problem statement. By doing these we have to possibly alter the percentage of crossover, mutation and increase the number of generations to get more optimal and suitable path.

References

[1] Duguleana, M., Barbuceanu, F.G., Teirelbar, A., et al.: 'Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning', Robot. Comput.-Integr. Manuf., 2012, 28, (2), pp. 132–146

[2] Bosscher, P., Hedman, D.: 'Real-time collision avoidance algorithm for robotic manipulators', Ind. Robot, 2011, 38, (2), pp. 186–197

[3] Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., et al.: 'Wireless sensor networks: a survey', Comput. Netw., 2002, 38, (4), pp. 393–422

[4] Yick, J., Mukherjee, B., Ghosal, D.: 'Wireless sensor network survey', Comput. Netw., 2008, 52, (12), pp. 2292–2330

[5] Li, H., Zhang, X.G., Liu, Y.: 'Energy efficient routing based on ordered topology of coal mine equipment', J. China Univ. Min. Technol., 2011, 40, (5), pp. 774–780