Meta-heuristic Nature Inspired Evolutionary controller for Quad-copter Target Tracking

Optimization Techniques ME-435

Priyam Gupta 2K17/ME/174 Prakrit Tyagi 2K17/ME/163

SUBMITTED TO: Dr Girish Kumar , Department of Mechanical Engineering
Delhi Technological University, Delhi, 110092

Contents

Metaheuristic Nature Inspired Evolutionary controller for Quadcopter Target Tracking

Prakrit Tyagi *2K17/ME/163* and Priyam Gupta *2K17/ME/174*

Delhi Technological University,Delhi-110092

Abstract

Target tracking by unmanned aerial vehicles (UAVs) in urban areas is an important application. However, due to the kinematic constraints of the UAV coupled with the visibility obstruction due to environmental factors impose hard constraints on the UAV for persistent tracking. In this paper, we propose a Metaheuristic Nature Inspired Evolutionary controller to persistently track an object on the ground. The controller determines the control commands for the UAV by minimising cost functions. The framework is simulated for 3D as well as 2D cases to validate the proposed approach.

1 Introduction

The autopilot systems are composed of an *inner loop* which controls the attitude of the drone and the *outer loop* is responsible for mission objectives. These control systems are popularly based on Proportional-Integral-Derivative (PID) control systems which have shown unprecedented success in stable environments. However the restraints of these classical control methods and their lack of robustness in unpredictable and harsh environments and difficulty in solving non-linear complexities in real time demands more adaptive and intelligent control systems. Genetic Algorithm offers an ideal candidate for controller designs due to its real time adaptability and the compatibility of its individual structures and control rules.

Quad-copter Target tracking is a non linear control problem which requires real time flexibility and robust control algorithms for successful operation. It is also one of the important applications of UAV which fall in both civilian and military domain. These characteristics make target tracking a suitable candidate for testing the novel control algorithm.

The controller design principle is fundamentally an optimization task with the objective of converging on such a controller structure and its parameters, which minimizes the chosen performance index. Genetic Algorithms are in principle, optimization algorithms which are problem independent and stochastic in nature. Most of the optimizer used in control systems are gradient based optimizer but they fail in situations where the system is discontinuous or non differentiable. The meta-heuristic nature provides flexibility and adaptability to the genetic algorithms which can be utilized in target tracking problems. The fundamental operations for reproduction in genetic programming are selection, crossover and mutation.

Crossover takes more computation time than other operations which is undesirable in a control problem in order to get minimum latency. IWO is an asexual genetic algorithm which replaces crossover with spatial dispersion wherein the seeds are produced by randomly selecting seeds from the normal distribution around the parent plant. This can significantly reduce the communication latency and allow for efficient and adaptive trajectory planning.

This work proposes the development of a target tracking system for a quad-copter in unpredictable physical environment. The framework is to perform autonomous tracking of a moving target and works as an Outer Loop Controller(OLC) maintaining the altitude and trajectory .There are a lot of trajectory planning algorithms using deterministic methods that sufficiently solves the problem under assumptions and fails to take in account the disturbances and non-linearity of the systems. The proposed OLC is based on a meta-heuristic evolutionary programming which works by searching a design space for various points and generating cost for each points, it then selects a point as a solution with minimum cost.

2 Literature Review

Genetic Algorithm is a popular and widely applied algorithm because of its robust and adaptive nature and ability to converge to global optima.

Dracopoulos et al[1] studied the basic understanding of application of genetic algorithm in controller design. In one research [2], Genetic Algorithm is also implemented for tuning the weights of the PID controller for a real time path planning. [3] proposes Genetic Algorithm (GA) to determine the shortest path that the quadrotor must travel given one target point to save energy and time without hitting an obstacle.

In this project we build upon the work of Galvez et al[3] and propose a trajectory planning framework based on Invasive Weed Optimization which is asexual in nature and we corrobortae it to be more robust and faster than the conventional Genetic Algorothm optimizer used by Galvez.

3 Methodology

3.1 Quad-copter Kinematics

In this section the non-linear kinematic model for Quad-copter Unmanned Aerial Vehicle is presented [4].

$$\begin{pmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{p_z} \end{pmatrix} = \begin{pmatrix} cos(\theta) & sin(\phi)sin(\theta) & cos(\phi)sin(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ -sin(\theta) & sin(\phi)cos(\theta) & cos(\phi)cos(\theta) \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{1}$$

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} p \\ q \\ r \end{pmatrix} \tag{2}$$

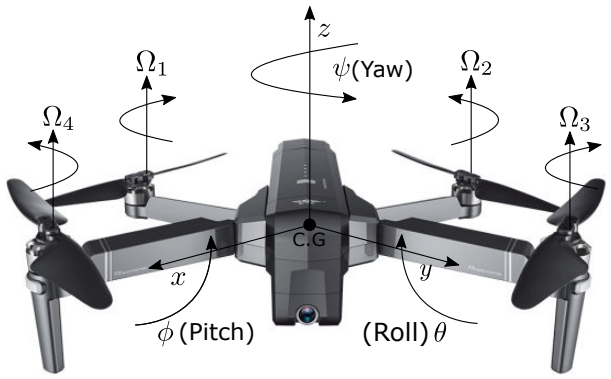


Figure 1: Quad-copter Model

where the state variables ϕ, θ, ψ, x, y, z represent pitch, roll, yaw, x-position, y-position, and altitude, respectively measure with respect to inertial frame and control variables u, v, w, p, q, r represent body frame velocities and angular rotations about these velocity vectors in body frame. With the state vector and control input vector defined as $X = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}$ and $U = \begin{bmatrix} u & v & w & p & q & r \end{bmatrix}$, the state equations represented can be written as:

$$\dot{X} = f(X) + \Phi(X)U(t) \tag{3}$$

where $f(X) \in \Re^{6\times 1}$ and $\Phi(X) \in \Re^{6\times 6}$ is the input coefficient matrix , assuming that ϕ and θ are small. This is used to calculate next state when current state and control inputs are given. The system inputs are related to rotor speeds as:

3.2 Genetic Algorithm

Genetic Algorithm is a meta-heuristic nature inspired evolutionary algorithm based on the principles of Darwinian theory of "survival of the fittest". It is an iterative process wherein an initial population of individuals is generated randomly. The fitness of these individuals is calculated using a cost function. This step is followed by reproduction where a new population of programs is created using the parent population. The programs with higher fitness value are prioritized for their contribution in the next generation of programs. An optimum solution to the objective is attained by repeating this process of evaluation and evolution. A schematic for the Genetic Algorithm process is depicted in Figure 2.

In this project we have used Invasive Weed Optimization[5] (a specific class of Genetic Algorithm) which takes inspiration from the capability of the weeds to survive in hostile conditions and efficiently utilizing the resources to invade the whole field. Before going into details of the algorithm, we will introduce certain fundamental terms associated with IWO algorithms. A seed is an individual candidate that is produced from the parent candidate. After evaluating the fitness of the seed it flowers into a plant which can be compared with other flowered plants. Maximum population size is the maximum number plants that can

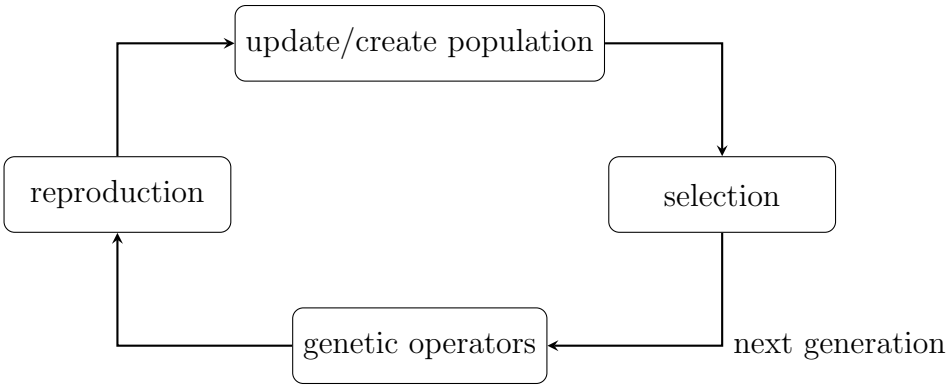


Figure 2: IWO Optimizer Overview Flowchart

exist in a particular iteration. The fundamental operations for reproduction in genetic programming are selection, crossover and mutation. Crossover takes more computation time than other operations which is undesirable in a control problem in order to get minimum latency. IWO is an asexual genetic algorithm which replaces crossover with spatial dispersion wherein the seeds are produced by randomly selecting seeds from the normal distribution around parent plant. The standard deviation of the normal distribution goes on decreasing every iteration. The produced plants along with the parents are ranked together and a top few elements are allowed to reproduce further. Over successive generations the desired qualities from each individual gets accumulated into the optimum candidate.

3.3 Genetic Algorithm Control Architecture

The control architecture consists of a IWO algorithm loop and a fitness calculation loop which utilizes prior known information i.e quad-copter model, states of quad-copter and of target. It is specified that this is an asexual genetic algorithm so the only process it uses are Generation, Dispersion and Selection. Fitness for each generation is calculated using cost functions which require future state of qua-copter and target. Therefore it can also be asserted that the proposed GA is predictive in nature. To get the future state of quad-copter we input the parameters in state equation (3) together with current state of quad-copter and solve to get new state. The new state of target is similarly generated. The cost function used are given below.

$$cost = \sqrt{(x_U - x_T)^2 + (y_U - y_T)^2} + W * (z_U - 50)^2 \tag{4}$$

where x_U,y_U are coordinates of UAV and x_T,y_T are coordinates of Target and W is weight used to normalize both cost functions

The fitness of each generation is ranked and then all the set of parameters are passed to production step. In this step according to the rank of set of parameters they are given limit to number seeds they can reproduce through asexual reproduction. In the the Spatial dispersion step the parameters produce seeds through Gaussian distribution using standard deviation assigned for the current iteration, they produce seeds within the limit assigned to them. These steps are run until the maximum number of iterations are reached and with

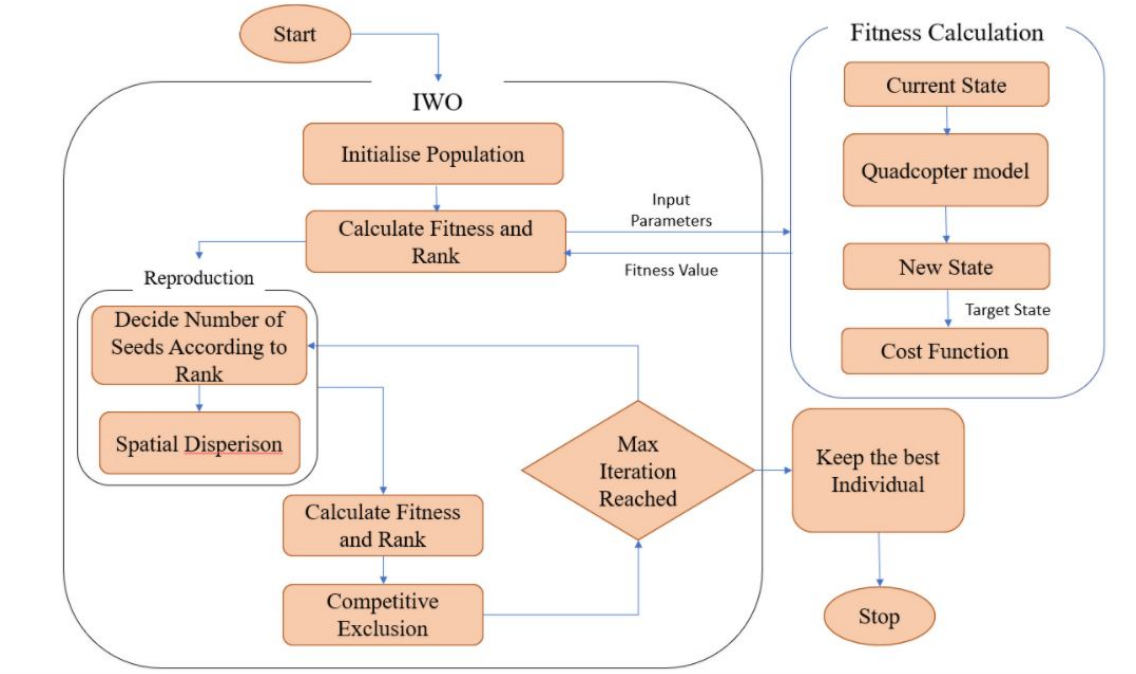


Figure 3: IWO Optimizer Detailed

each forward iteration the standard deviation of each iteration is decreased exponentially. The solution is output at max iteration.

This finishes solving for solution for the current time step and then when time step changes the above steps are run agains.

The Simulation Framework is schematically shown in the Figure 4.

4 Results

The proposed framework was tested in a 2D using a car model as well as in 3D using a quadcopter model and the resulting trajectories are given below. The simulation environment was created using python3.6.

In the First case which is 2D a ground vehicle will chase a ground target moving or fixed at a location. We have applied a Ackerman steering model for both the chaser and target with angular rate ω being the only control input and the velocity is kept constant for the chaser.

$$\dot{x} = V * \cos\theta \tag{5}$$

$$\dot{y} = V * \sin\theta \tag{6}$$

$$\dot{\theta} = \omega \tag{7}$$

Then we tested our controller for three different cases and the trajectory results for which are shown in figure 5, 6 and 7.

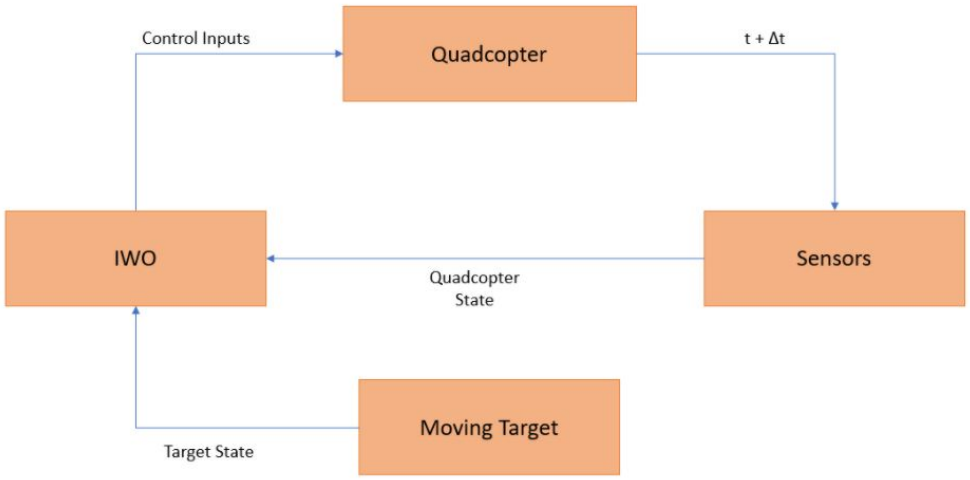


Figure 4: Simulation Framework

- Fixed Target , figure 5
- Moving Target with $V_{target} < V_{Chaser}$, figure 6
- Moving Target with $V_{target} > V_{Chaser}$, figure 7

In the Second case which is 3D case a quad-copter is used to chase a ground moving target. We generated two results for different target trajectories.

- Target in Straight path
- Target moving in circular path

The parameters for the Genetic Algorithm which were used in the implementation are delineated in the Table 1. The trajectory followed by the quad-copter in 3D environment is demonstrated in Figure 8

Table 1: IWO Parameters

Parameter	Value/Range
Maximum Population Size	20
Max Seed	8
Min Seed	3
Initial Std Deviation	35
Final Std Deviation	0.01
Modulation Index	3
Max Iteration Number	3000

In the 2D simulation for the fixed target the objective of the bot was to reach the target position through shortest path while its starting position was facing 120° to the x-axis. In

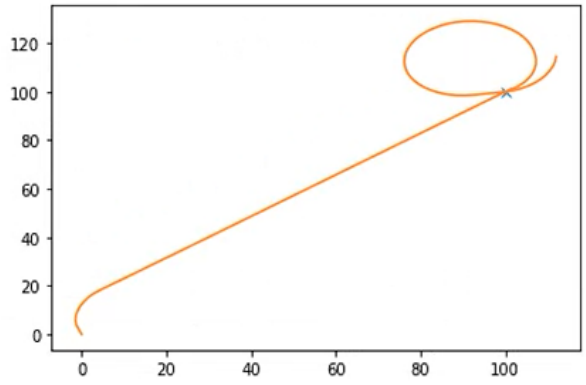


Figure 5: Trajectory for Fixed Target Tracking (X represents the fixed target)

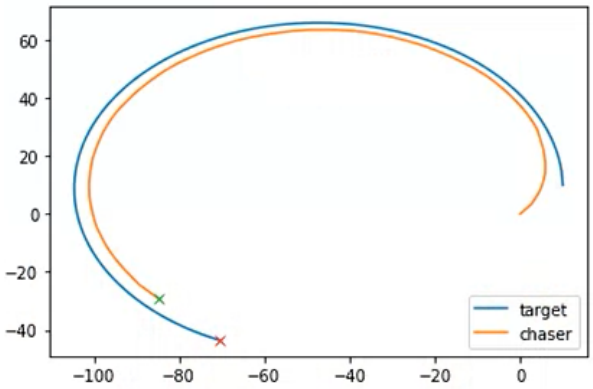
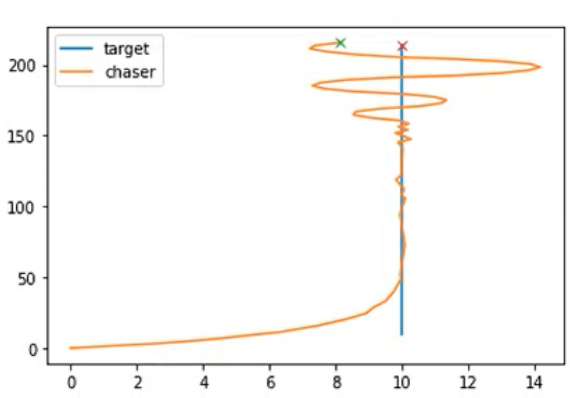
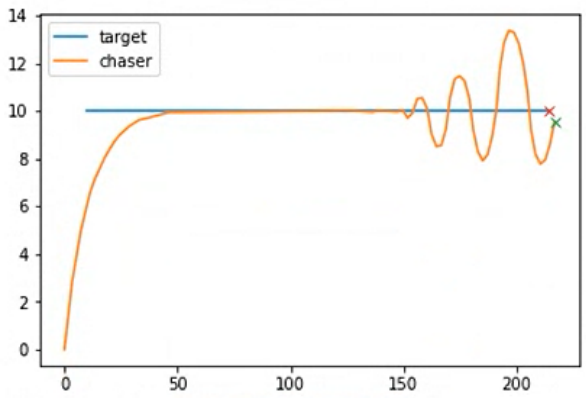


Figure 6: Trajectory for moving target with $V_{target} > V_{Chaser}$



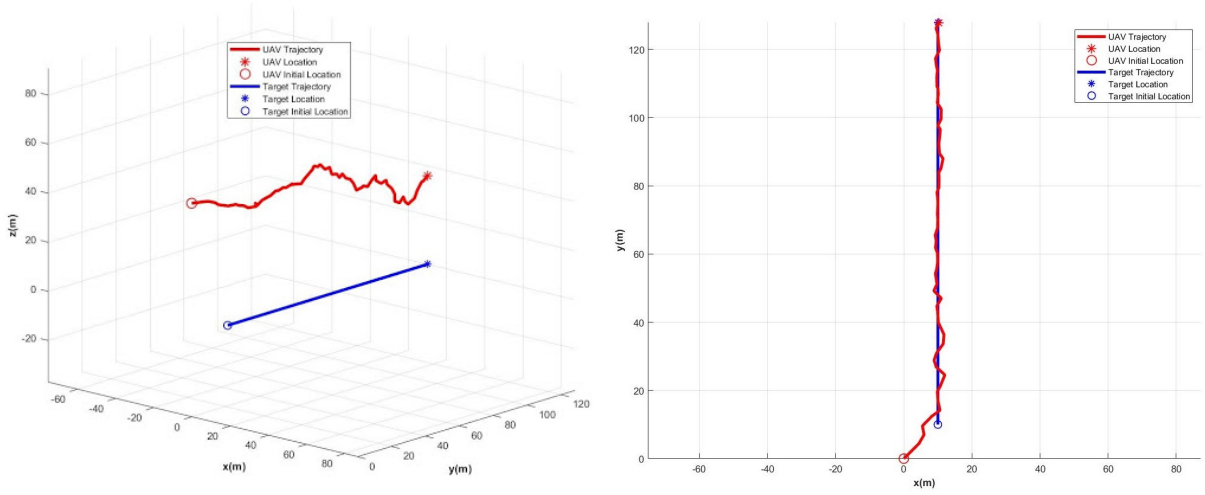
(a) Target following a vertical path with constant velocity



(b) Target following a horizontal path with constant velocity

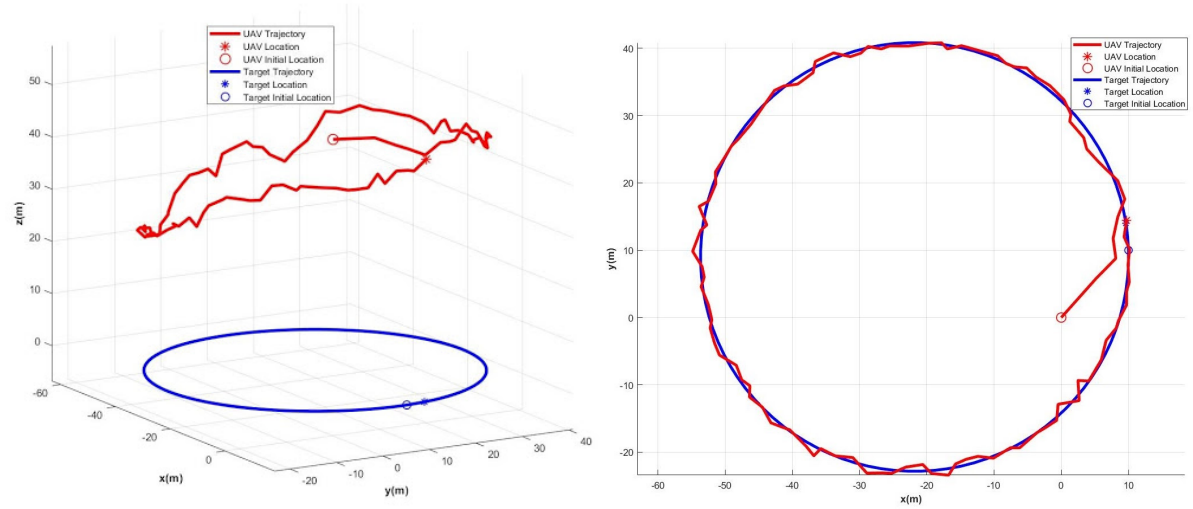
Figure 7: Trajectory for moving target with $V_{target} < V_{Chaser}$

figure 5 it can be seen that the bot was able to reach the target successfully. Even after running for extra time steps it goes around to reach back to the target position. In the moving target case the bot had to follow the target as it moves along its path. We tested



(a) UAV trajectory for target moving in a straight line

(b) Top view for Target following a straight line



(c) UAV trajectory for Target following a circular path

(d) Top view for Target following a circular path

Figure 8: Trajectory for 3D implementation of the framework

moving target case for both when the target velocity is greater than chaser velocity and vice versa. For the case where the target velocity is greater than the chaser velocity, chaser tried to maintain minimum possible constant distance between them as shown in figure 6. In the other moving target case the velocity of target was more than chaser therefore once catching up with the target it overshoots and tries to catch it again as seen in figure 7. This happens repeatedly thus a trajectory similar to exponential sinusoidal pattern can be seen.
In the 3D simulation for the target moving in straight line the quad-copter objective was to reach the target and follow it. Initial position of UAV was (0,0) and initial position of Target was (10,10). It can be seen in the figure 8a and 8b that the UAV was able to follow

the target through the simulation time. Similarly for the case of Target moving in circular trajectory as evident in figures 8c and 8d the quad-copter followed the target.

5 Conclusion

We Successfully implemented Meta-heuristic Optimization Algorithm for Target Tracking and showed results for 2D and 3D case. The Chaser (quad-copter) was able to track the target in all the cases but the smoothness of the trajectory can be improved in 3-D case which can be achieved by using dynamic model of a quad-copter.Future Work may involve adding Obstacle Avoidance and incorporating Kalman Filter for prediction of states since we assumed no noise.

References

[1] D. C. Dracopoulos, "Genetic algorithms and genetic programming for control," in *Evolutionary algorithms in engineering applications*, pp. 329–343, Springer, 1997.

[2] H. K. Tran and T. N. Nguyen, "Flight motion controller design using genetic algorithm for a quadcopter," *Measurement and Control*, vol. 51, no. 3-4, pp. 59–64, 2018.

[3] R. L. Galvez, E. P. Dadios, and A. A. Bandala, "Path planning for quadrotor uav using genetic algorithm," in *2014 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pp. 1–6, IEEE, 2014.

[4] J. McCormack, *Quadcopter Attitude Control Optimization and Multi-Agent Coordination*. PhD thesis, University of New Hampshire, 2019.

[5] A. R. Mehrabian and C. Lucas, "A novel numerical optimization algorithm inspired from weed colonization," *Ecological informatics*, vol. 1, no. 4, pp. 355–366, 2006.

6 Code

6.1 Invasive Weed Optimization Code

```
import random,copy,math
import numpy as np
import matplotlib.pyplot as plt
import time
pi = round(math.pi,6)
sin,cos,tan = math.sin,math.cos,math.tan

delta_t = 0.2
car_vel = 10
car_omega = 0
target_pos = np.array([10,10,0])

MAX_POPULATION_SIZE=20 #maximum number of plants in a colony(or population)
sigma_fin=0.01 #final standard deviation
sigma_ini=35 #initial standard deviation
Smin=2       #min seeds produced
Smax=10      #max seeds produced
n_mi= 3      #modulation index
iter_max=1500 #Maximum number of iterations to be done
CHROMOSOME_SIZE=6 #v_x,v_y,v_Z,omega_x,omega_y,omega_z

#Input Limits
v_x_min,v_x_max = -15,15
v_y_min,v_y_max = -15,15
v_z_min,v_z_max = -11,11
omega_x_min,omega_x_max = -(pi/5),pi/5
omega_y_min,omega_y_max = -(pi/5),pi/5
omega_z_min,omega_z_max = -(pi/5),pi/5

#class that generates chromosomes
class Chromosome:
    def __init__(self,state,tar_curr_state,mode =" "):
        self._genes=np.zeros((CHROMOSOME_SIZE+1),dtype=float)
        self._state = state #current state of the chaser
        self._tar_curr_state = tar_curr_state #current state of the target
        self.new_tar_state = 0

        #initializing the control parameters
        if mode=="initialise":

            self._genes[0]= v_x_min+(v_x_max-v_x_min)*random.random()
            self._genes[1]= v_y_min+(v_y_max-v_y_min)*random.random()
            self._genes[2]= v_z_min+(v_z_max-v_z_min)*random.random()
```

```
            self._genes[3]= omega_x_min+(omega_x_max-omega_x_min)*random.random()
            self._genes[4]= omega_y_min+(omega_y_max-omega_y_min)*random.random()
            self._genes[5]= omega_z_min+(omega_z_max-omega_z_min)*random.random()

        self._cost = self.get_cost() #calculating cost
        self._genes[-1] = self._cost #adding cost to the genes


    def get_genes(self):
        return self._genes

    def get_cost(self):

        vel = self._genes[:3]
        omega = self._genes[3:-1]

        pos = self._state[0]
        angle = self._state[1]
        phi,theta,psi = angle


        rotmat_velocity =
            np.array([[cos(theta),sin(phi)*sin(theta),cos(phi)*sin(theta)],\
                                  [0,cos(phi),-sin(phi)],\
                                  [-sin(theta),sin(phi)*cos(theta),cos(phi)*cos(theta)]])

        pos = pos + np.matmul(rotmat_velocity,vel)*delta_t
        angle = angle + omega*delta_t


        curr_pos = np.array([pos,angle])

        #getting target position
        self.new_target_state = self.cal_target_state() #updating new target state
        self._state = curr_pos #updating new chaser state
        cost = np.sum((self.new_target_state[:2]-curr_pos[0][:2])**2) #+
            abs(curr_pos[0][2]-50))/51 #- curr_pos[0][2]*1000 #calculating cost

        return cost

    def get_state(self):
        return self._state

    def cal_target_state(self):
        theta = self._tar_curr_state[-1] + (car_omega*delta_t)
        x = self._tar_curr_state[0] + car_vel * math.cos(theta) * delta_t
        y = self._tar_curr_state[1] + car_vel * math.sin(theta) * delta_t
```

```
        return np.array([x,y,0,theta])

    def get_target_state(self):
      return self.new_target_state

    def __str__(self):
        return self._genes.__str__()

#class that create one set of generations
class Population:
    def __init__(self,size,curr_state,tar_curr_state,mode=" "):
        self._chromosomes=[]
        i=0
        while i<size:
            self.add_chromosomes(Chromosome(curr_state,tar_curr_state,mode))
            i+=1
    def add_chromosomes(self,chromosome):
        self._chromosomes.append(chromosome)
    def get_chromosomes(self):
        return self._chromosomes

#class that helps in evolving and mutating the genes of the chromosomes
class GeneticAlgorithm:

    @staticmethod
    def reproduce(pop,iter,curr_state,tar_curr_state):
        new_pop=copy.deepcopy(pop)
        worst_cost=pop.get_chromosomes()[MAX_POPULATION_SIZE-1].get_genes()[CHROMOSOME_SIZE]
        best_cost=pop.get_chromosomes()[0].get_genes()[CHROMOSOME_SIZE]
        sigma_iter=GeneticAlgorithm.std_deviation(iter,iter_max)
        if(best_cost!=worst_cost):

            for i in range(MAX_POPULATION_SIZE):
                ratio=(pop.get_chromosomes()[i].get_genes()[CHROMOSOME_SIZE]-worst_cost)/(best_c
                S=Smin+(Smax-Smin)*ratio #number of seeds chromosome can produce
                    on the basis of rank
                for j in range(int(S)):
                    seed=Chromosome(curr_state,tar_curr_state)
                    for k in range(CHROMOSOME_SIZE):
                        seed._genes[k]=np.random.normal(pop._chromosomes[i].get_genes()[k],sigma_

                    new_pop.add_chromosomes(seed)
            GeneticAlgorithm.sort(new_pop)
            pop._chromosomes=new_pop._chromosomes[:MAX_POPULATION_SIZE]

        else:
            # print("best and worst cost equal")
```

```
            return pop, False
        # print("REPRODUCED")
        return pop, True
    @staticmethod
    def std_deviation(iter,iter_max):
        sigma_iter=(((iter_max-iter)**n_mi)/iter_max**n_mi)*(sigma_ini-sigma_fin)+sigma_fin
        # print ("sigma",sigma_iter,'\n')
        return sigma_iter
    @staticmethod
    def sort(pop):
        pop_chroms_2d_array=np.array([pop.get_chromosomes()[i].get_genes() for i
            in range(len(pop._chromosomes))])
        sindices=np.argsort(pop_chroms_2d_array[:,CHROMOSOME_SIZE],axis=0)

        sorted_chroms=pop.get_chromosomes()
        for i in range(0,len(pop._chromosomes)):
            sorted_chroms[i]._genes=pop_chroms_2d_array[sindices[i]]
            sorted_chroms[i]._state = pop._chromosomes[sindices[i]]._state

        pop._chromosomes=sorted_chroms
        # print("SORTED")
#-------------------------------------------------------------------------------------------
def _print_population(pop,gen_number,fitness):
    print("\n-----------------------------------------------------------")
    print("Generation#",gen_number,"|Fittest chromosome
        fitness:",pop.get_chromosomes()[0].get_genes()[CHROMOSOME_SIZE])
    #print("Target Chromosome:",TARGET_CHROMOSOME)
    print("-----------------------------------------------------------")
    fitness.append(pop.get_chromosomes()[0].get_genes()[CHROMOSOME_SIZE])
    i=0
    for x in pop.get_chromosomes():
        print("Chromosome #",i+1,":",x.get_genes(),"|State:
            ",x.get_state(),"|Fitness:",x.get_genes()[CHROMOSOME_SIZE])
        k=x.get_genes()[CHROMOSOME_SIZE]
        i+=1


def optimizer(curr_state,tar_curr_state):
  # print("INITIATING PROGRAMME")
  population=Population(MAX_POPULATION_SIZE,curr_state,tar_curr_state,"initialise")#initialisin
      population
  GeneticAlgorithm.sort(population)
  fitness = []
  iter=1

  while iter<iter_max:
```

```
        population,check=GeneticAlgorithm.reproduce(population,iter,curr_state,tar_curr_state)

        if(check==False):
            #iter+=1
            break;

        iter+=1

    new_chaser_state,new_target_state =
        population.get_chromosomes()[0].get_state(),population.get_chromosomes()[0].get_target_st
    return(new_chaser_state,new_target_state)
```

6.2 Overall Framework Code

```
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from mpl_toolkits.mplot3d import axes3d
%matplotlib inline


initial_state = np.array([[0,0,50],[0,0,0]])
target_initial_state = np.array([10,10,0,pi/2])

curr_state,target_curr_state = optimizer(initial_state,target_initial_state)

num_steps = 100
j=0
chaser_states = [initial_state]
target_states = [target_initial_state]
while j <= num_steps:

    curr_state,target_curr_state = optimizer(curr_state,target_curr_state)
    print("\nstep: ",j,"chaser state: ", curr_state, "target state:
        ",target_curr_state)
    chaser_states.append(curr_state.tolist())
    target_states.append(target_curr_state.tolist())

    #plotting
    chaser_states_l = np.array(chaser_states)
    target_states_l = np.array(target_states)
    fig = plt.figure()
    ax = fig.gca(projection="3d")
    ax.plot(target_states_l.T[0], target_states_l.T[1], target_states_l.T[2],label
        = "Target");
    ax.plot(chaser_states_l[:,0].T[0], chaser_states_l[:,0].T[1],
        chaser_states_l[:,0].T[2],label = "Chaser");
```

```
  ax.legend()
  plt.show()

  j+=1

chaser_states = np.array(chaser_states)
target_states = np.array(target_states)
# print(states)
```