T.C. KOCAELİ ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ

PWM OUTPUT OPTIMIZATION USING THE GREY WOLF ALGORITHM

MÜHENDİSLİK TASARIMI-3

ARA RAPORU

JORDI HAKO NINTCHEU
170207095

Bölümü: Elektronik ve Haberleşme Mühendisliği

Danışman: Dr. SERHAT YILMAZ

KOCAELİ, 2021/2022

CONTENT

FIGURES INDEX

ICONS AND ABBREVIATIONS

GWO: Grey Wolf Optimizer
IGWO: Improved Grey Wolf Optimizer
DE: Differential Evolution
DGWO: Differential Grey Wolf Optimizer
SGWO: Survival Grey Wolf Optimizer
SOF: Survival Of The Fittest
PSO: Particle Swarm Optimizer
ABC: Artificial Bee Colony
CS: Cuckoo Search
SFLA: Shuffled Frog Leaping Algorithm
ACO: Ant Colony Optimization
BBO: Biogeography Based Optimization

PWM OUTPUT OPTIMIZATION USING THE GREY WOLF ALGORITHM

Jordi Hako Nintcheu

SUMMARY: The grey wolf optimizer (GWO) is a novel type of swarm intelligence optimization algorithm. An improved grey wolf optimizer (IGWO) with evolution and elimination mechanism was proposed so as to achieve the proper compromise between exploration and exploitation, further accelerate the convergence and increase the optimization accuracy of GWO. The biological evolution and the “survival of the fittest " (SOF) principle of biological updating of nature are added to the basic wolf algorithm. The differential evolution (DE) is adopted as the evolutionary pattern of wolves. The wolf pack is updated according to the SOF principle so as to make the algorithm not fall into the local optimum. That is, after each iteration of the algorithm, sort the fitness value that corresponds to each wolf by ascending order, and then eliminate R wolves with the worst fitness value, meanwhile randomly generate wolves equal to the number of eliminated wolves. Finally, 12 typical benchmark functions are used to carry out simulation experiments with GWO with differential evolution (DGWO), GWO algorithm with SOF mechanism (SGWO), IGWO, DE algorithm, particle swarm algorithm (PSO), artificial bee colony (ABC) algorithm and cuckoo search (CS) algorithm. Experimental results show that IGWO obtains the better convergence velocity and optimization accuracy.

In this article, we propose a model to optimize the output of a PWM using the grey wolf optimizer algorithm, to be used in underwater machines and vehicles. This paper and the work attached to it, could be done thanks to the research and thesis of A.A OLUOGLU of Kocaeli university on the subject OBJECT TRACKING WITH ANFIS METHOD IN UNDERWATER VEHICLES. His work is referenced below.

Key words: GWA, GWO, Grey Wolf Algorithm, Grey Wolf Optimizer, Underwater vehicles, IOT robot, artificial intelligence, machine learning, deep learning.

1. INTRODUCTION

The swarm intelligence algorithms are proposed to mimic the swarm intelligence behavior of biology in nature, which has become a hot topic of cross-discipline and research in recent years. The appearance of swarm intelligent optimization algorithms provides fast and reliable methods for finding solutions on many complex problems. Because the swarm intelligence algorithm has characteristics of self-organization, parallel, distributive, flexibility and robustness, now they have been very widespread used in many cases, such as electric power system, communication network, system identification and parameter estimation, robot control, transportation and other practical engineering problems. Therefore, the research on the swarm intelligence optimization algorithms has an important academic value and practical significance.

At present, a variety of swarm intelligence optimization algorithms have been proposed by simulating the biotic population and evolution process in nature, such as particle swarm optimization (PSO) algorithm, shuffled frog leaping algorithm (SFLA), artificial bee colony (ABC) algorithm, ant colony optimization (ACO) algorithm, biogeography-based optimization (BBO) algorithm, and cuckoo search (CS) algorithm. Particle Swarm Optimization (PSO) algorithm put forward by Kennedy and Eberhart to mimic the foraging behavior of birds and fish flock, but the convergence velocity and searching accuracy of PSO algorithm are unsatisfactory to some extent. Shuffled Frog-leaping Algorithm (SFLA) put forward by Eusuff in 2003 is a novel swarm intelligent cooperative searching strategy based on the natural memetics. On the one hand, individuals exchange information in the global searching process, and its search precision is high. On the other hand, SFLA has the disadvantage of slow convergence velocity and easy to fall into the local optimum. Artificial Bee Colony (ABC) Algorithm put forward by Karaboga in 2005 to mimic the finding food source behavior of bees. In order to mimic the social behavior of the ant colony, Dorigo et al. Proposed the novel Ant Colony Optimization (ACO) Algorithm in 2006. But their disadvantages are the slow convergence speed and easy to premature convergence. The Biogeography-Based Optimization (BBO) algorithm was put forward by Simon in 2008, whose idea is based on the geographical distribution principle in biogeography. Cuckoo Search (CS) Algorithm was proposed by Yang and Deb in 2009 based on the cuckoo's parasitic reproduction mechanism and Levy flights searching strategy, whose advantage is that CS algorithm is not easy to fall into the local optimum compared with other intelligent algorithms and has less parameters, and whose disadvantage is that the adding of Levy Flight search mechanism leads to strong leap in the process of search, thus, its local search is not careful.

The common shortcoming of these algorithms is that each swarm intelligence algorithm has problems in different degrees such that the convergence velocity is

slow, the optimization precision is low, and easy to fall into the local optimum. The key reason for these shortcomings is whether an algorithm is able to achieve the proper compromise between exploration and exploitation in its each searching phase or not. Exploration and exploitation are contradictory. Exploration reflects the ability of the algorithm to search for new space, while exploitation reflects the refining ability of the algorithm. These two criteria are generally used to evaluate stochastic optimization algorithms. Exploration refers to when a particle leaves the original search path to a certain extent and searches towards a new direction, which reflects the ability of exploiting unknown regions. Exploitation refers to when a particle continues to search more carefully on the original trajectory to a certain extent, which can insure the wolf makes a detailed search to the region that has been explored. Too small exploration can cause premature convergence and falling into a local optimum, however, too small exploitation will make the algorithm converge too slowly.

The grey wolf optimizer (GWO) as a novel swarm intelligent optimization algorithm was put forward by Seyedali Mirjalili etc in 2014, which mainly mimics wolf leadership hierarchy and hunting mechanism in nature. Seyedali and Mirjalili have proved that the optimization performance of standard GWO is superior to that of PSO, GSA, DE and FEP algorithms. Due to the wolves algorithm with the advantages of simple in principle, fast seeking speed, high search precision, and easy to realize, it is more easily combined with the practical engineering problems. Therefore, GWO has high theoretical research value. But GWO is a new biological intelligence algorithm, the research about it is just at the initial phase, so research and development of the theory are still not perfect. In order to make the algorithm perform better, further exploration and research is needed.

Many swarm intelligence algorithms mimic the hunting and searching behaviors of some animals. However, GWO simulates the internal leadership hierarchy of wolves, thus, in the searching process the position of best solution can be comprehensively assessed by three solutions. But for other swarm intelligence algorithms, the best solution is searched only led by a single solution. So GWO can greatly decrease the probability of premature and falling into the local optimum. So as to achieve the proper compromise between exploration and exploitation, an improved GWO with evolution and elimination mechanism is proposed. The biological evolution and the SOF principle of biological updating of nature are added to the basic wolf algorithm. In order to verify the performance of the improved GWO, typical benchmark functions are adopted to carry out simulation experiments, meanwhile, experimental results are compared with PSO algorithm, ABC algorithm and CS algorithm. The experimental results show that the improved grey wolf optimizer (IGWO) obtains the better convergence velocity and optimization accuracy.

The paper is organized as follows. In section 3, the Greywolf algorithm is explained and subsequently used to optimize the output of a PWM. The algorithm designed for the purpose is included and final code is attached as a pdf to this paper.

2. RELATED WORKS

A lot of work has previously been done relating GWO to up to date problems in the science and technology fields. Some of these works include, the traveling salesman problem (TSP), the minimum spanning tree problem, nonlinear equation systems, power flow problem and others.

2.1. Travelling Salesman Problem (TSP)

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Note the difference between Hamiltonian Cycle and TSP. The Hamiltonian cycle problem is to find if there exists a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.

This problem has been successfully solved using the grey wolf optimizer which we discuss in this paper.

2.2. The minimum labeling spanning tree problem

The minimum labeling spanning tree problem is to find a spanning tree with least types of labels if each edge in a graph is associated with a label from a finite label set instead of a weight.

A bottleneck edge is the highest weighted edge in a spanning tree. A spanning tree is a minimum bottleneck spanning tree (or MBST) if the graph does not contain a spanning tree with a smaller bottleneck edge weight. A MST is necessarily a MBST (provable by the cut property), but a MBST is not necessarily a MST

This also, is a successful application of the GWO.

2.3. Collision-Free Navigation in Crowded Scenarios

The power flow problem is to find a unique solution of a power system (given certain input variables) and, therefore, it is central to power grid analysis. Amongst the many problems which can be solved using the Grey wolf optimizer, this occupies a renowned position as the GWO has effectively been used to resolve this problem.

3. BACKGROUND AND OVERVIEW

The grey wolf optimizer is a novel heuristic swarm intelligent optimization algorithm proposed by Seyedali Mirjalili et al. in 2014. The wolf, as top predators in the food chain, has a strong ability to capture prey. Wolves generally like social life and in the interior of the wolves exists a rigid social hierarchy.

In order to mimic wolves internal leadership hierarchy, the wolves is divided into four types of wolf: **alpha**, **beta**, **delta** and **omega**, where the best individual, second best individual and third best individual are recorded as alpha, beta, and delta, and the rest of the individuals are considered as omega. In the GWO, the hunting (optimization) is guided by alpha, beta, and delta8. They guide other wolves (W) tend to the best area in searching space. In the iterative searching process, the possible position of prey is assessed by three wolves alpha, beta, and delta. In the optimization process, the locations of wolves are updated based on Eqs (1) and (2).

$$D = |C \cdot XP(t) - X(t)| \quad (1)$$

$$X(t+1) = XP(t) - A \cdot D \quad (2)$$

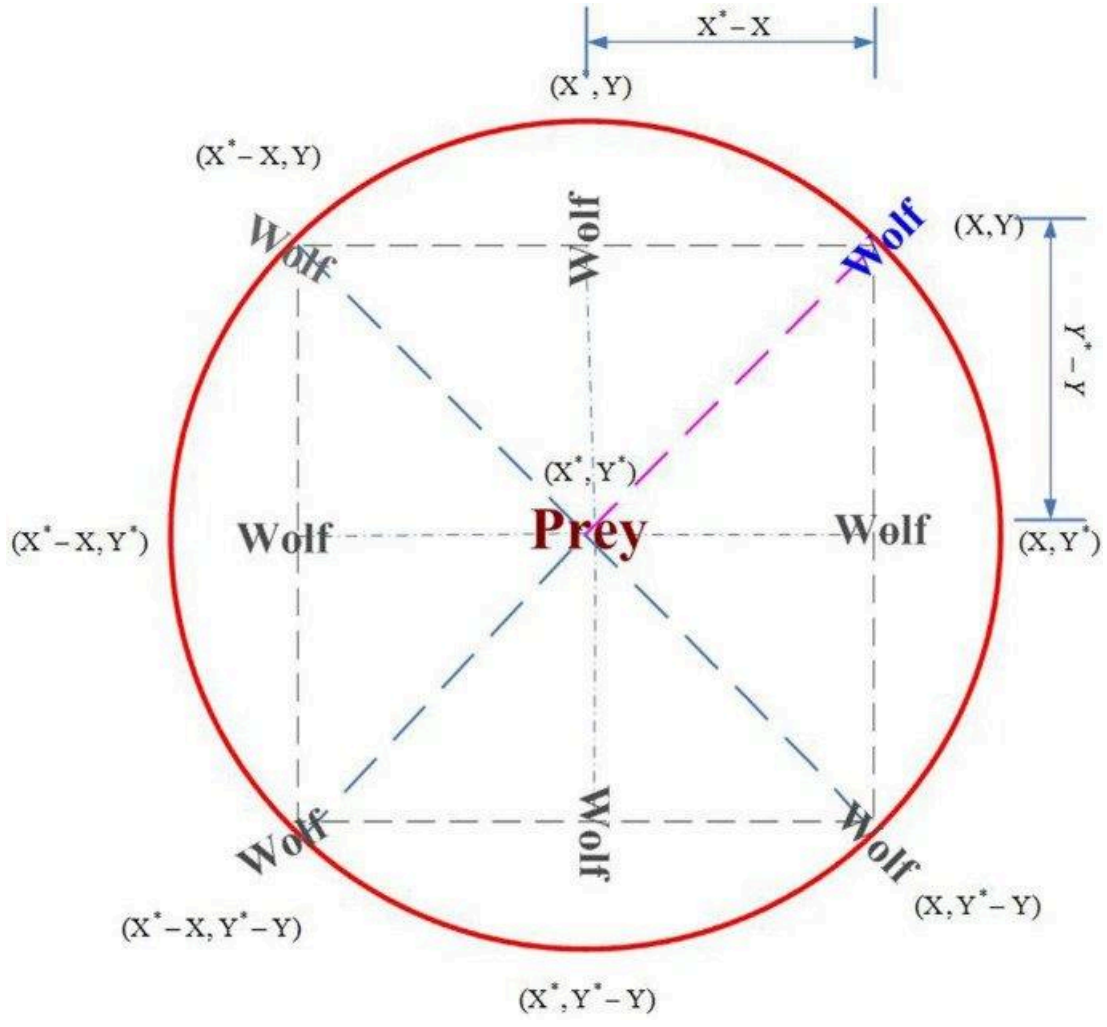


Fig 3.1: Possible Wolf's Positions

where, t represents the t-th iteration, A and C are coefficient vectors, XP is the position vector of prey, X represents the wolf position. The vector A and C can be expressed by:

A = 2a · r1 − a (3)
C = 2 · r2 (4)

where, the coefficient linearly decreases from 2 to 0 with the increasing iteration number, r1 and r2 are random vectors located in the scope [0, 1].

The principle of the position updating rules described in Eqs (1) and (2) are shown in Fig. 3. 1. It can be seen from Fig. 1 the wolf at the position (X, Y) can relocate itself around the prey according to above updating formulas. Although Fig. 3.1 only shows 7 positions that the wolf could possibly move to, by adjusting the random parameters C and A it can make the wolf relocate itself to any position in the continuous space near prey. In the GWO, it always assumes that the position of alpha, beta and delta is likely to be the prey (optimum) position. In the iteration searching process, the best individual, second best individual and third best individual obtained so far are respectively recorded as alpha, beta, and delta. However, other wolves who are regarded as omega relocate their locations according to the locations of alpha, beta, and delta. The following mathematical formulas are used to re-adjust positions of the wolf omega. The conceptual model in which a wolf updates its position is shown in Fig. 2.

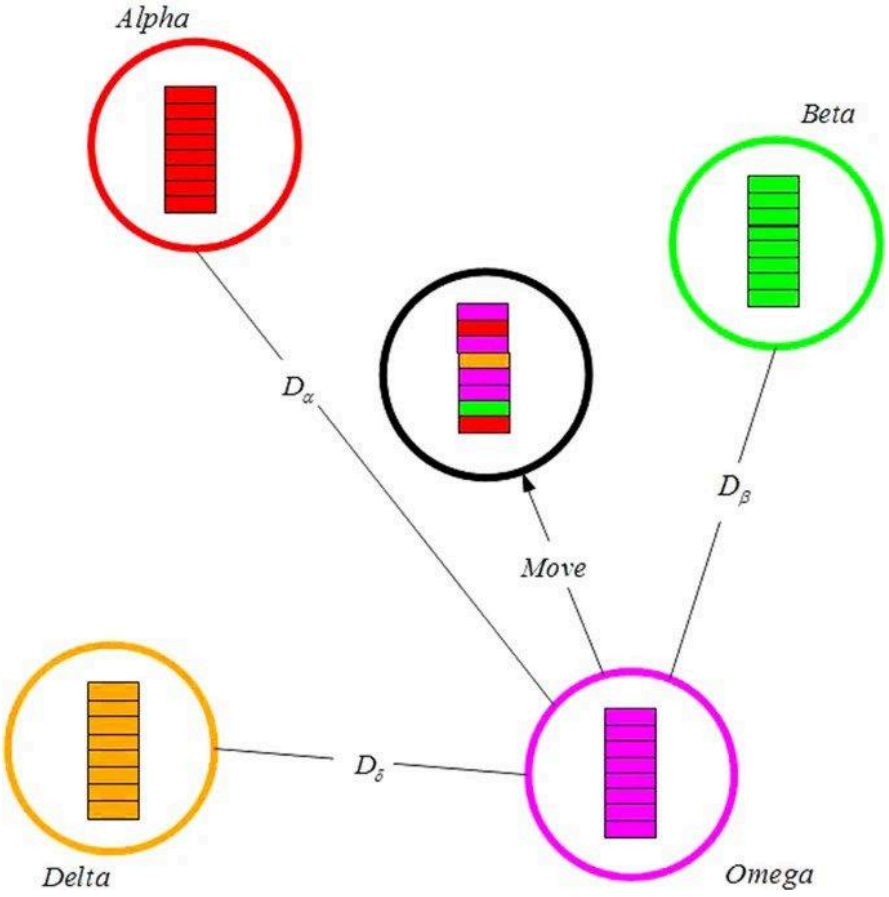


Fig 3.2: The conceptual model in which a wolf updates its position

$D\alpha = |C1 \cdot X\alpha - X|$ ...(5)
$D\beta = |C2 \cdot X\beta - X|$ …(6)
$D\delta = |C3 \cdot X\delta - X|$ …(7)

where, Xα, Xβ and Xδ are the position vector of alpha, beta, and delta, respectively. C1, C2, C3 are randomly generated vectors, X represents the position vector of the current individual. The Eqs (5), (6) and (7) respectively calculate the distances between the position of the current individual and that of individual alpha, beta, and delta. So the final position vectors of the current individual are calculated by:

$X1 = X\alpha - A1.(D\alpha)$ (8)
$X2 = X\beta - A2 \cdot (D\beta)$ (9)
$X3 = X\delta - A3 \cdot (D\delta)$ (10)
$X(t+1) = X1 + X2 + X3 /3$ (11)

where, A1, A2, A3 are randomly generated vectors, and t represents the number of iterations.

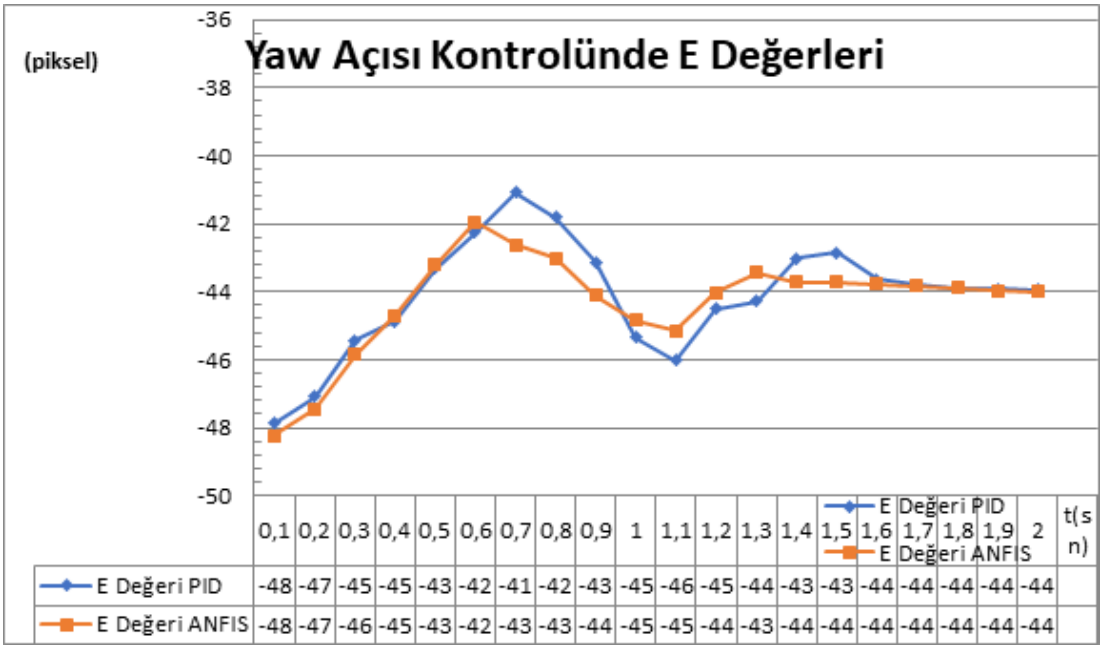
4. IMPLEMENTATION OF THE GWO TO IMPROVE THE OUTPUT OF THE PWM

Below are the steps used in the implementation of the GWO for optimizing the output of a PWM.

4.1. Problem Identification.

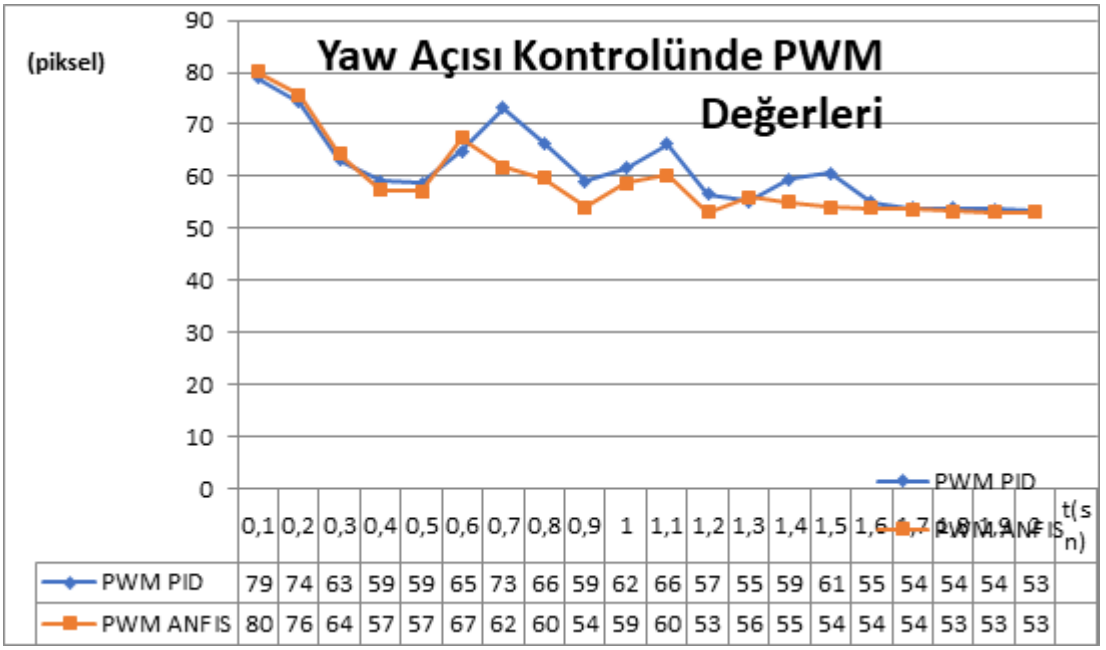
One of the biggest challenges of the implementation of the GWO on a project is the problem identification. This is very important as it will go a long way to facilitate the obtention of wanted output.

For this project we were presented with a series of error values output by the PWM and the wanted correct values as shown in the figures below;



	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	2	t(sn)
E Değeri PID	-48	-47	-45	-45	-43	-42	-41	-42	-43	-45	-46	-45	-44	-43	-43	-44	-44	-44	-44	-44	
E Değeri ANFIS	-48	-47	-46	-45	-43	-42	-43	-43	-44	-45	-45	-44	-43	-44	-44	-44	-44	-44	-44	-44	

Fig 4.1: This Yaw Error Angles Of the PWM



	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	2	
PWM PID	79	74	63	59	59	65	73	66	59	62	66	57	55	59	61	55	54	54	54	53	
PWM ANFIS	80	76	64	57	57	67	62	60	54	59	60	53	56	55	54	54	54	53	53	53	

Fig 4.2: This Yaw Expected Angles Of the PWM

Our main goal was to use the GWO to always get the right(the wanted) PWM value for a given error value.

4.2. Methodology

For this project, we took a series of error values and expected values as pairs. From the formula

P - E = S

Where P is the expected value and E is the error value, our goal is to find an S for which P - E is 0 or close to 0.

This implies that we need to minimize the value of the function P - E or of the function S.

Knowing that the PWM works according the value of the duty cycle, we opted to use the GWO to find a duty cycle value between 0 and 255 for which S+E is closed to P

Let S be a function of the output voltage Vo of the PWM. If S varies proportionally to a change in Vo, then reducing the change in Vo will reduce S

Where change(Vo) = change(d)/256 X Vt and Where Vt is 12V for the motor used in this experiment. Hence Vo will vary from 0 to 12V.

change(d) = Xa where Xa is our best solution

Our goal is to calculate the change in duty cycle that will result in a maximum error reduction. This can be done by using the function pwm_optimize which is attached with the code below.

Our expected voltage will vary with the expected displacement angle as follows:

Açı Değeri	**Beklenen Sonuç**	**Denk Geldiği Gerilim**	
		Sol Motor	**Sağ Motor**
90°	1	0v/12v	
85°	1	0v/12v	
80°	1	0v/12v	
75°	1	0v/12v	
70°	1	0v/10v	
65°	1	0v/10v	
60°	1	0v/10v	
55°	1	0v/10v	
50°	1	0v/8v	

45°	1	0v/8v
40°	1	0v/8v
35°	1	0v/8v
30°	1	0v/6v
25°	1	0v/6v
20°	1	0v/6v
15°	1	0v/6v
10°	1	0v/6v
5°	0.5	6v/6v
0°	0.5	6v/6v
-5°	0.5	6v/6v
-10°	0	6v/0v
-15°	0	6v/0v

-20°	0	6v/0v
-25°	0	6v/0v
-30°	0	6v/0v
-35°	0	8v/0v
-40°	0	8v/0v
-45°	0	8v/0v
-50°	0	8v/0v
-55°	0	10v/0v
-60°	0	10v/0v
-65°	0	10v/0v
-70°	0	10v/0v
-75°	0	12v/0v
-80°	0	12v/0v

-85°	0	12v/0v
-90°	0	12v/0v

Table 4.2.1: Expected and Actual Values of the PWM

4.3. Algorithm

For this project to be carried out successfully, we developed the following algorithm based on the GWO algorithm, and decided to write our code using the python language as we are more familiar with this language.

Below is a step by step explanation of our algorithm. The complete code for this algorithm is attached to this paper as a separate pdf

Step1: Randomly initialize Grey wolf population of N particles Xi (i=1, 2, …, n)

Step2: Calculate the fitness value of each individuals
sort grey wolf population based on fitness values
alpha_wolf = wolf with least fitness value
beta_wolf = wolf with second least fitness value
gamma_wolf = wolf with third least fitness value

Step 3: For Iter in range(max_iter): # loop max_iter times
calculate the value of a
a = 2*(1 - Iter/max_iter)
For i in range(N): # for each wolf
a. Compute the value of A1, A2, A3 and C1, C2, C3
A1 = a*(2*r1 -1), A2 = a*(2*r2 -1), A3 = a*(2*r3 -1)
C1 = 2*r1, C2 = 2*r2, C3 = 2*r3

b. Computer X1, X2, X3
X1 = alpha_wolf.position -
A1*abs(C1*alpha_wolf_position - ith_wolf.position)
X2 = beta_wolf.position -
A2*abs(C2*beta_wolf_position - ith_wolf.position)
X3 = gamma_wolf.position -
A3*abs(C3*gamma_wolf_position - ith_wolf.position)

c. Compute new solution and it's fitness
Xnew = (X1 + X2 + X3) / 3

```
            fnew = fitness( Xnew)

        d. Update the ith_wolf greedily
            if( fnew < ith_wolf.fitness)
                ith_wolf.position = Xnew
                ith_wolf.fitness = fnew
    End-for

    # compute new alpha, beta and gamma
        sort grey wolf population based on fitness values
        alpha_wolf = wolf with least fitness value
        beta_wolf  = wolf with second least fitness value
        gamma_wolf = wolf with third least fitness value
End -for
```

Step 4: Return best wolf in the population

4.4. Code Explanation

The fist part of our code consists of a function which will serve as our fitness function. Amongst a multitude of fitness functions such as the sphere function, etc, we opted to use the *rastrigin* function as our fitness function for this experiment.

```
import random
import math    # cos() for Rastrigin
import copy    # array-copying convenience
import sys     # max float

def fitness_rastrigin(position):
 fitness_value = 0.0
 for i in range(len(position)):
   xi = position[i]
   fitness_value += (xi * xi) - (10 * math.cos(2 * math.pi * xi))
 return fitness_value
```

The next part of our code is a class to initialize the wolf structure. Each and every wolf of our population will inherit from the attributes of our wolf class.

```
# wolf class
class wolf:
  def __init__(self, fitness, dim, minx, maxx, seed):
    self.rnd = random.Random(seed)
    self.position = [0.0 for i in range(dim)]
```

```
    for i in range(dim):
      self.position[i] = ((maxx - minx) * self.rnd.random() + minx)

    self.fitness = fitness(self.position) # curr fitness
```

Next, we initialize the GWO function which is the actual grey wolf optimizer. Each Time we want to optimize a certain system, we will call on this function to get the fittest value, that is the alpha wolf's position.

```
def gwo(fitness, max_iter, n, dim, minx, maxx):
    rnd = random.Random(0)

    # create n random wolves
    population = [ wolf(fitness, dim, minx, maxx, i) for i in range(n)]

    # On the basis of fitness values of wolves
    # sort the population in asc order
    population = sorted(population, key = lambda temp: temp.fitness)

    # best 3 solutions will be called as
    # alpha, beta and gamma
    alpha_wolf, beta_wolf, gamma_wolf = copy.copy(population[: 3])


    # main loop of gwo
    Iter = 0
    while Iter < max_iter:

        # after every 10 iterations
        # print iteration number and best fitness value so far
        if Iter % 10 == 0 and Iter > 1:
            print("Iter = " + str(Iter) + " best fitness = %.3f" % alpha_wolf.fitness)

        # linearly decreased from 2 to 0
        a = 2*(1 - Iter/max_iter)

        # updating each population member with the help of best three members
        for i in range(n):
            A1, A2, A3 = a * (2 * rnd.random() - 1), a * (
              2 * rnd.random() - 1), a * (2 * rnd.random() - 1)
            C1, C2, C3 = 2 * rnd.random(), 2*rnd.random(), 2*rnd.random()

            X1 = [0.0 for i in range(dim)]
            X2 = [0.0 for i in range(dim)]
            X3 = [0.0 for i in range(dim)]
```

```
            Xnew = [0.0 for i in range(dim)]
            for j in range(dim):
                X1[j] = alpha_wolf.position[j] - A1 * abs(
                  C1 - alpha_wolf.position[j] - population[i].position[j])
                X2[j] = beta_wolf.position[j] - A2 * abs(
                  C2 -  beta_wolf.position[j] - population[i].position[j])
                X3[j] = gamma_wolf.position[j] - A3 * abs(
                  C3 - gamma_wolf.position[j] - population[i].position[j])
                Xnew[j]+= X1[j] + X2[j] + X3[j]

            for j in range(dim):
                Xnew[j]/=3.0

            # fitness calculation of new solution
            fnew = fitness(Xnew)

            # greedy selection
            if fnew < population[i].fitness:
                population[i].position = Xnew
                population[i].fitness = fnew

        # On the basis of fitness values of wolves
        # sort the population in asc order
        population = sorted(population, key = lambda temp: temp.fitness)

        # best 3 solutions will be called as
        # alpha, beta and gaama
        alpha_wolf, beta_wolf, gamma_wolf = copy.copy(population[: 3])

        Iter+= 1
    # end-while

    # returning the best solution
    return alpha_wolf.position
```

Our next block is a test driver code for the rastrigin function;

```
# Driver code for rastrigin function

print("\nBegin grey wolf optimization on rastrigin function\n")
dim = 1
fitness = fitness_rastrigin


print("Goal is to minimize Rastrigin's function in " + str(dim) + " variables")
print("Function has known min = 0.0 at (", end="")
```

```
for i in range(dim-1):
  print("0, ", end="")
print("0)")

num_particles = 50
max_iter = 100

print("Setting num_particles = " + str(num_particles))
print("Setting max_iter    = " + str(max_iter))
print("\nStarting GWO algorithm\n")



best_position = gwo(fitness, max_iter, num_particles, dim, -255, 255)

print("\nGWO completed\n")
print("\nBest solution found:")
print(["%.6f"%best_position[k] for k in range(dim)])
err = fitness(best_position)
print("fitness of best solution = %.6f" % err)

print("\nEnd GWO for rastrigin\n")


print()
print()
```

And finally, the last block of code is a function which optimizes the output of the PWM and returns 3 distinct values, which are the duty cycle optimization value of the PWM, the motor left and motor right optimization value.

```
def pwm_optimize(expected_val, err_val, angle):

    best_position = gwo(fitness, max_iter, num_particles, dim, -10.0, 10.0)

    err_reducer = best_position[0]

    diff = expected_val - err_val
    d = ((256 * diff) / 90) + (err_val * err_reducer)

    if angle >= 10:
        right_motor = (d / 256) * 12
        left_motor = 0
    elif angle >= -5:
        right_motor = (d / 256) * 12
        left_motor = (d / 256) * 12
```

```
else:
    right_motor = 0
    left_motor = (d / 256) * 12

return d, right_motor, left_motor
```

5. EXPERIMENTAL RESULTS

At the end of this experiment, we were able to create a simple GWO(Grey wolf optimizer) which for every expected value and error value entered, reduced the error and returned the corresponding voltage reduction or addition that should be implemented by the PWM in order to minimize the error.

Example results:

Expected Angle	Error Angle	Duty cycle optimization output by optimizer	Left Motor output optimization	Right Motor output optimization
-50°	-45	-14.222108957172221	0 V	-0.6666613573674478 V

Table 5.1: Sample Results of The GWO algorithm

```
In [42]: a, b, c= pwm_optimize(-50, -45, -50)

         print('The PWM duty cycle will be increase/reduced by: ', a)
         print('The left motor\'s output voltage will be increase/reduced by: ', b, 'V')
         print('The right motor\'s output voltage will be increase/reduced by: ', c, 'V')

         Iter = 10 best fitness = -9.999
         Iter = 20 best fitness = -9.999
         Iter = 30 best fitness = -9.999
         Iter = 40 best fitness = -9.999
         Iter = 50 best fitness = -9.999
         Iter = 60 best fitness = -10.000
         Iter = 70 best fitness = -10.000
         Iter = 80 best fitness = -10.000
         Iter = 90 best fitness = -10.000
         The PWM duty cycle will be increase/reduced by:  -14.222108957172221
         The left motor's output voltage will be increase/reduced by:  0 V
         The right motor's output voltage will be increase/reduced by:  -0.6666613573674478 V
```

Fig 5.1: Results Screen Shot

6. CONCLUSIONS, LIMITATIONS AND FUTURE WORK

The GWO algorithm is great when it comes to optimizing the output of a system. During our experiment, we were able to optimize the output of a PWM using the concepts behind this great algorithm.

One of the biggest problems we faced in this study, and hence in the implementation of this algorithm, was the problem identification. ¾ of the time allocated to this project was spent in an effort to identify the problem and hence propose an adequate solution using the GWO algorithm.

Our future work in this field would be the implementation of this algorithm as well as other SWAN algorithms underwater vehicles to optimize their performances.

ACKNOWLEDGEMENT

This work was done and set up thanks to the help and request of Dr. SERHAT YILMAZ, to whom we say thank you for the support.

REFERENCES

SUALTI ARAÇLARINDA ANFIS YÖNTEMİ İLE NESNE TAKİBİ:
Grey Wolf Optimization Algorithm:
https://www.baeldung.com/cs/grey-wolf-optimization
Science direct:
https://www.sciencedirect.com/science/article/abs/pii/S0149197021001360
Paralax.com:
https://www.parallax.com/go/PBASICHelp/Content/LanguageTopics/Commands/PWM.htm#:~:text=To%20determine%20the%20proportional%20PWM,average%20voltage%20is%201.953%20V.
Mathematical models for the Grey Wolf Optimizer:
https://www.youtube.com/watch?v=wA5FPPOYOKk
Implementation of the Grey Wolf Optimization Algorithm
https://www.geeksforgeeks.org/implementation-of-grey-wolf-optimization-gwo-algorithm/?ref=lbp

ATTACHMENTS

PWM OUTPUT OPTIMIZATION USING THE GREY WOLF ALGORITHM full python code