A PROJECT REPORT ON

PATH PLANNING IN ROBOTICS

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING (CST ML and AI)

Submitted by:

Puru Singh	**2017528**
Ashish Upadhyay	**2017474**
Nikita Kumari	**2017657**
Parth Sarthi	**2017520**

Under the Guidance of

Dr. Upma Jain

Assistant Professor

Project Team ID: MP23CST006

Department of Computer Science and Engineering
Graphic Era (Deemed to be University)
Dehradun, Uttarakhand

June-2024

CANDIDATE'S DECLARATION

I/We hereby certify that the work is being presented in the Project Report entitled **"Path Planning in Robotics"** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering and submitted to the Department of Computer Science and Engineering of the Graphic Era (Deemed to be University), Dehradun is an authentic record of my work carried out during a period from **August-2023 to May-2024** under the supervision of **Dr. Upma Jain**, **Assistant Professor**, Department of Computer Science and Engineering, Graphic Era (Deemed to be University). The matter presented in this dissertation has not been submitted by me/us for the award of any other degree of this or any other Institute/University.

Puru Singh	2017528	signature
Ashish Upadhyay	2017474	signature
Nikita Kumari	2017657	signature
Parth Sarthi	2017520	signature

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Signature **Supervisor**

Signature **Head of the Department**

External Viva

Name of the Examiners: **Signature with Date**

1.

2.

Abstract

One of the most significant processes in robotics is path planning, which is significant for the development of vehicles that can run on their own in various fields, from cars to vacuum cleaners and drones. Effective navigation capabilities allow robots to travel from one point to another without collisions or long distances or trips. A* and Dijkstra are the traditional algorithms which have set the foundation for this field but often fail in dynamic and complex environments and hence a lot of advanced techniques are needed here.

This report discusses a new hybrid method with the PSO and GWO algorithms to improve path planning. Integrating PSO and GWO compensates for PSO's slow convergence rate with GWO's ability to escape local optima and vice versa. PSO first searches and improves the possible paths by updating particle position and velocity, but GWO approaches tune such paths to account for obstacles and find the shortest path. Simulation experiments show that this hybrid algorithm is superior to the classic methods and can be used as an effective algorithm for autonomous robotic navigation.

Keywords: Particle Swarm Optimization (PSO), Cubic Spline, Robotic Path Planning, Grey Wolf Algorithm (GWO)

Acknowledgement

Any achievement, be it scholastic or otherwise does not depend solely on individual effort but on the guidance, encouragement, and cooperation of intellectuals, elders, and friends. Several personalities in their capacity have helped me in carrying out this project work.

Our sincere thanks to project guide **Dr. Upma Jain, Assistant Professor,** Department of Computer Science and Engineering, Graphic Era (Deemed to be University), for her valuable guidance and support throughout the course of project work and for being a constant source of inspiration.

We extend our thanks to **Prof. (Dr.) Guru Prasad M.S.**, Project Coordinator, Department of Computer Science and Engineering, Graphic Era (Deemed to be University), for his valuable suggestions throughout all the phases of the Project Work.

We are extremely grateful to **Prof. (Dr.) D. P. Singh**, HOD of the Computer Science and Engineering Department, Graphic Era (Deemed to be University), for his moral support and encouragement.

We thank the **management of Graphic Era (Deemed to be University)** for the support throughout the course of our bachelor's degree and for all the facilities they have provided.

Last, but certainly not least we thank all teaching and non-teaching staff of Graphic Era (Deemed to be University) for guiding us on the right path. Most importantly we wish to thank our parents for their support and encouragement.

Puru Singh	2017528
Ashish Upadhyay	2017474
Nikita Kumari	2017657
Parth Sarthi	2017520

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Project Introduction

Path Planning is an important subprocess in the overall robot's movement, which is a key part of functional and optimized autonomous robotics schemes of many applications means, such as automotive cars, house cleaning robots, unmanned aerial vehicles, and industrial robots. It is the way of finding an efficient path for a robot in a defined environment through certain points starting from the initial point and leading to a preset terminal point with corresponding objectives – to avoid obstacles, to minimize distance, and time and to give the best path possible. Originally, path planning algorithms like A* *[1]* and Dijkstra's have been there and now considered as solid tools in work with static environment conditions. All these conventional algorithms employ structured directional techniques to look for shortest pathways while maintaining a certain minimal distance from the obstacles. Nevertheless, they have some disadvantages which are easily recognizable in the case of unsteady or uncertain circumstances when the contour of the ground or the position of the obstacle may change frequently and requires more innovative and flexible approaches.

Based on these central issues, we have observed a great use of research approaches that try to integrate the strong attributes of various algorithms to boost path planning in dynamic surroundings. An example of such creativity is a combination of particle swarm optimization (PSO) and Grey Wolf Optimizations (GWO) *[8][9]* addressing the usefulness of optimizing both the algorithms and the path planning. Thanks to this algorithm that tries to emulate swarming particles to get to the solution of a problem as soon as possible, *PSO [7] is* one of the most efficient algorithms for initial route and local optimization. On the other hand, GWO is far more effective in fine-tuning these solutions since the algorithm mimics hierarchical leadership and hunting patterns of the grey biological wolves, which helps to steer clear away from obstacles and come to terms with complicated, multi-modal terrains.

Based on solving our method combines the PSO's fast exploitation ability during preliminary optimization phase and GWO's precise and flexible updating manner during the second phase of optimization. This two-step optimization strategy is especially useful for avoiding limitations discovered when using standard path-planners in environments that

require real-time operations. PSO, certainly, yields faster convergence while GWO proves to offer meticulous refinement which definitively makes for a strong and versatile path planning solution that outperforms the minimum path length and efficiently avoids obstacles. The experimental outcomes affirm this hybrid approach, proving it to be a significantly enhanced approach over the universally traditional one for the reflexive estimates such as path length minimization and obstacle detection to justify the future potential in improving the performance of the self-governing robots in a manifold and dynamic establishment.

1.2 Problem Statement

Though efficient in static environments, the representation and path planning algorithms like A* and Dijkstra are not adept at handling the vagaries of changing terrains. Therefore, there is a lack of more suitable and effective methods for enabling the path planning of autonomous robots in such situations.

1.3 Objectives

- **Optimal Path Discovery:** To enhance robotic navigation by determining the most efficient route in various environments. This involves employing advanced path planning algorithms that calculate paths focusing on minimizing both travel time and distance.
- **Collision avoidance:** To steer clear of any obstacles present in the environment between the initial and target coordinates. This is crucial for ensuring the robot's safe and unobstructed movement through its path.
- **Effective trajectory of avoidance:** Navigating around obstacles using a trajectory that not only ensures avoidance but does so with efficiency. The goal is to minimize the overall distance covered, consequently reducing the time taken for navigation.

Chapter 2

Literature Survey/ Background

Path Planning has been applied in guiding the mobile robots to reach a particular objective from very simple trajectory planning to the selection of a suitable sequence of action.

Path Planning for mobile robots is divided into two categories: *[1]* Global Path Planning and Local Path Planning.

The Global Path Planning requires a map of environment to calculate the best route. Depending on the analysis of the map, some methods are based on Roadmaps are proposed.

To transform the Global Path into suitable waypoints, the Local Path Planning creates new waypoints taking into consideration the dynamic obstacles and the vehicles constraints.

2.1 The Spectrum of Global Path Planning Algorithms

Global path planning is the core of robot navigation, with the goal of charting a collision-free trajectory while traveling the lowest distance possible. Various techniques have been developed and used to optimize global pathways. This spectrum includes:

2.1.1 **A∗ Algorithm** *[2]*: A heuristic approach capable of finding the optimal path by analysing the graph in a more informed way compared to classic search algorithms.

2.1.2 **Ant Colony Optimization (ACO)** *[3]*: A probabilistic method of tackling computational problems but with disadvantages such as delayed convergence and a proclivity to converge to local optima.

2.1.3 **Genetic Algorithm (GA)** *[4]*: A genetic algorithm is used to find the optimal path for a mobile robot to move in a static environment expressed by a map with nodes and links.

2.1.4 **Slime Mould Optimization Algorithm (SMOA)** *[5]*: Slime mould optimization algorithm (SMA) is a recently presented metaheuristic technique that is inspired by the behaviour of slime mould. Slow convergence speed is a fundamental problem in SMA as in other metaheuristic optimization methods.

2.1.5 **Salp Swarm Algorithm (SSA)** *[6]*: Inspired by the swarming behaviour of salps when navigating and foraging in oceans, SSA is tested on several mathematical optimization functions to observe and confirm their effective behaviours in finding the optimal solutions for optimization problems.

2.1.6 Particle Swarm Optimization (PSO) *[7]*: An evolutionary algorithm that has been applied to many different engineering and technological problems with considerable success. It has been continually modified trying to improve its convergence properties.

2.1.7 Improved Particle Swarm Optimisation *[8]*: In this paper, an improved version of the traditional PSO was implemented. The inertia of the particles is varied throughout the iterations, the values of c_1 and c_2 are also changed based upon this modified value of the inertial weight. The formula used for the modification of inertia with each iteration is given as equations 2.1 and 2.2:

$$A = it\frac{\ln \omega_{\max} - \ln \omega_{\min}}{MaxIt} - \ln \omega_{\max}, \qquad \text{.....(2.1)}$$

$$\omega = \exp(-A), \qquad \text{.....(2.2)}$$

Where $\boldsymbol{\omega_{max}}$ and $\boldsymbol{\omega_{min}}$ are the upper and lower limits of the inertia, ***it*** is the current iteration and ***MaxIt*** is the maximum number of iterations, $\boldsymbol{\omega}$ is the modified inertia. The value of $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ is calculated by equations 2.3 and 2.4:

$$c_1 = \cos(\omega), \qquad \text{.....(2.3)}$$

$$c_2 = \sin(\omega), \qquad \text{.....(2.4)}$$

2.1.8 Improved PSO-GWO Algorithm *[9]*: In this paper, Grey Wolf optimisation was used in combination with improved PSO. The modification of inertial weight after each iteration is adaptive based on the current $\boldsymbol{g_{best}}$ and the $\boldsymbol{g_{best}}$ of previous iteration. The formula for this inertia is given by:

$$\omega = (\omega_{max} + \omega_{min}) * a - \frac{\omega_{max} * k}{MaxIter} \qquad \text{.....(2.5)}$$

$$a = \frac{p^k{}_{gbest}}{p^{k-1}{}_{gbest}} \qquad \text{.....(2.6)}$$

where $\boldsymbol{\omega_{max}}$ and $\boldsymbol{\omega_{min}}$ are the upper and lower limits of inertia respectively, *k* is the current iteration and ***MaxIter*** is the maximum number of iterations, $\boldsymbol{p^k{}_{gbest}}$ is the global best for current iteration, and $\boldsymbol{p^{k-1}{}_{gbest}}$ is the global best of previous iteration.

Chapter 3

Software Design

3.1 Introduction

This project requires creating a simulation with a single particle robot that moves in an environment containing only static circular obstacles and the language to use is Python. In this context, the problem of interest is formulated as follows: to determine the best possible route between two points, a starting point and end point, with the help of an algorithm that combines PSO and GWO algorithms. Environment is a 25x25 cell grid on which there are three randomly sized circular regions marked as obstacles. The start and end coordinates are declared in the global section and the values are assigned by the user input before the execution of the program.

3.2 Environment Setup

In this evaluation, the simulation environment is characterized by a 25×25 grid. Within this grid obstacles are represented as circles with arbitrary positions and radii. In this case, three circular obstacles are inserted in the grid, though the architecture can accommodate as many obstacles as may be desired. The starting and ending places are set by the user to give the ease of testing different conditions on the robot.

3.3 Grid and Obstacles Representation

3.3.1 Grid Setup:

The grid consists of a 25x25 search space that has the start coordinates, the end coordinates, and the obstacles enclosed within it.

3.3.2 Obstacle Placement:

Obstacles are circular and are drawn as circles with two coordinates representing the center and a radius.

The obstacles are generated arbitrarily to blend into the grid system to avoid the situation where they block each other.

3.4 Path Planning Algorithms

The simulation employs a hybrid path planning approach using PSO and GWO. The parameters for both algorithms are fine-tuned to suit the size of the environment and the complexity of the obstacle configuration.

3.4.1 Particle Swarm Optimization (PSO)

PSO is a meta heuristic technique derived from simulating the natural world which specifically looks at the social behavior of a flock of birds. In this project, PSO is introduced to first provide a fast initial sweeping through the search area and to obtain a near-optimal path from the starting point to the destination.

Initialization: Strategies (potential paths) are randomly placed at the start of each search space.

Update: Particles move around to adopt a position which would be most beneficial in the light of their own experience as well as the experience of other particles of their neighborhood.

Termination: The algorithm continues the search until a precondition is satisfied like iteration counts or the attainment of convergence.

3.4.2 Grey Wolf Optimization (GWO)

GWO is an optimization algorithm based on the social leadership hierarchy and hunting strategy of Grey Wolves. It is used in this project to fine tune the paths that are produced by PSO and for eliminating any obstacles and boundaries which are present in the environments.

Initialization: These particles of PSO act as initial solution to GWO.

Hierarchy Emulation: Grey wolves are divided into four distinct groups: alpha, beta, delta and omega which are solutions available on the menu.

Position Update: Wolves will change their positions in reference to the alpha, beta, and delta wolves.

Convergence: This goes on until either a better solution path is developed, or the set number of nodes is traversed through.

3.5 Code Execution Workflow

The code follows a structured workflow to simulate the robot's movement and optimize the path:

3.5.1 Input Handling

The start and end coordinates are provided by the user. The positions and radii of the obstacles are initialized.

3.5.2 PSO Execution

- Initialize the particle swarm.
- Run the PSO algorithm to find a near-optimal path.
- Calculate and store the length of the best path found by PSO.

3.5.3 GWO Execution

- Use the particles processed by PSO as the initial population for GWO.
- Run the GWO algorithm to refine the path.
- Calculate and store the length of the optimized path.

3.5.4 Output Generation

- Generate graphs showing the paths taken by the robot as determined by both PSO and GWO.
- Display the path lengths for comparison.

3.6 Visualization and Analysis

The performance of the approaches is then analyzed and evaluated through graphs that illustrate the trajectories of the robot as determined by both PSO and GWO. From the graphs, one gets a clearer view of the efficacy of the two algorithms in traversing through the terrain impeded with obstacles.

3.6.1 Graph Plotting

- Plot the 25x25 grid and the positions of the circular obstacles.
- Overlay the paths generated by PSO and GWO on the grid.
- Differentiate the paths using distinct colors or line styles for clarity.

3.6.2 Path Length Comparison

- Display the calculated path lengths for the best solutions found by PSO and GWO.
- Highlight any improvements in path length achieved through the GWO refinement process.

Chapter 4

Requirements and Methodology

4.1 Requirements

The code for this project requires the following:

4.1.1 Hardware Requirements

4.1.1.1 A CPU of 6 cores and 3.3 GHz speed was used to run the code of this project. A minimum RAM of 4GB is required to execute the code for smooth and fast output generation.

4.1.2 Software Requirements

4.1.2.1 Python version above 3.10.0 is recommended for up-to-date libraries and dependency support.

4.1.2.2 Visual Studio Code was used to contrast the python program and simulate the path planning. Jupyter Notebook and Google Colab can also be used as alternatives.

4.1.2.3 Python libraries for matplotlib, scipy, numpy, time, and math must be installed on the system being used.

4.2 Methodology

The methodology of the project follows the following steps from the initialization of obstacles to the generation of optimal path using PSO and GWO hybrid approach. The execution of PSO and GWO is serial and the path length for both is calculated after each of the algorithms reach their maximum number of iterations.

The flow diagram of the code is given below:

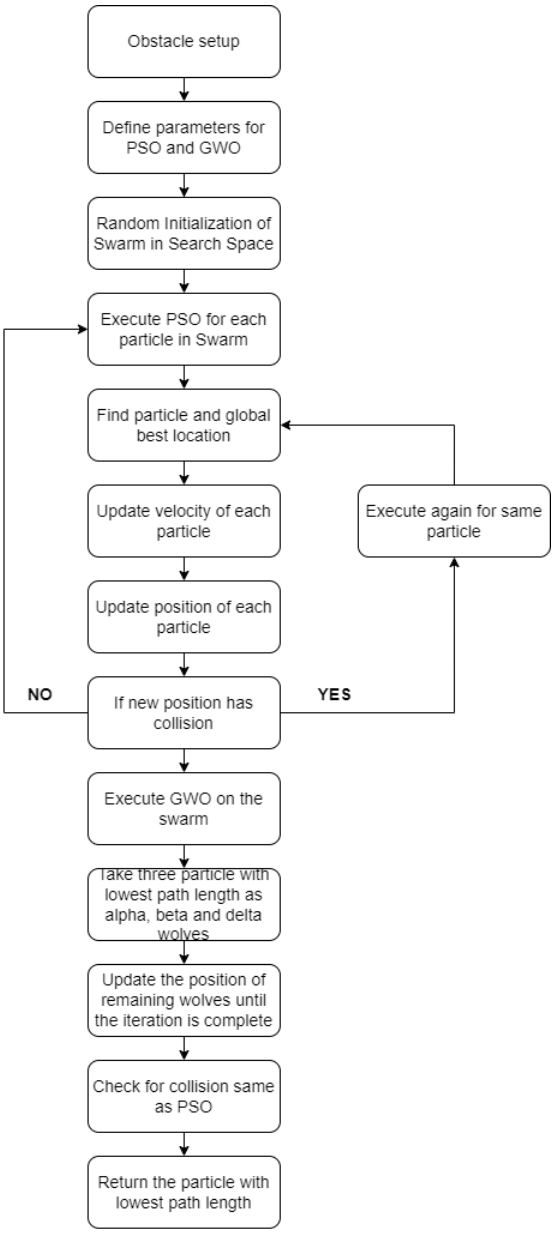

Fig 4.1 Flow diagram of the PSO+GWO hybrid approach

4.2.1 Obstacle setup and defining parameters for PSO and GWO

The obstacles are defined by the coordinates of their center and the radii. The obstacles used in this simulation are circular for the sake of simplicity. The obstacles are given arbitrary positions and radii, such that they do not overlap with each other and have enough space among them such that the movement of the robot is possible without collision.

After obstacle setup, the parameters for the two algorithms, Particle Swarm Optimization (PSO) and Grey Wolf Optimization (GWO) are defined. These parameters include:

4.2.1.1 PSO Parameters: Swarm size, maximum velocity of the particles in the swarm, cognitive coefficient, social coefficient, upper and lower limits of inertia, and the maximum number of iterations.

4.2.1.2 GWO Parameters: Swarm size, maximum number of iterations, and convergence criteria.

Fine tuning of these parameters is important to achieve optimal results for a given problem.

4.2.2 Random Initialization of Swarm in Search Space

A function is used to randomly initialize the particles in the search space. Each particle is given a particular set of x and y coordinates that define its position. The initial personal best ($\boldsymbol{p_{Best}}$) and global best positions ($\boldsymbol{g_{Best}}$) are determined by the help of a loss function.

4.2.3 Execute PSO for Each Particle in Swarm

The PSO algorithm is executed to explore the search space. For each particle in the swarm:

1. Evaluate the fitness of the particle by calculating the length of the path it represents.
2. Check for collisions with obstacles along the path.
3. Update the value of inertial weight.

4.2.4 Find Particle and Global Best Location

After evaluating all particles, the algorithm identifies:

Particle Best ($\boldsymbol{p_{Best}}$): The best position (path) each particle has achieved so far.

Global Best ($\boldsymbol{g_{Best}}$): The best position (path) achieved by any particle in the swarm.

4.2.5 Update Velocity and Position of Each Particle

Each particle has its own velocity, which is updated based on the following equation (4.1):

$$\boldsymbol{P_{vel} = w.P_{vel} + R_1.C_1.\left(P_{best} - P_{pos}\right) + R_2.C_2.(S_{best} - P_{pos})} \quad \ldots (4.1)$$

$\boldsymbol{P_{vel}}$ is the velocity of the particle, $\boldsymbol{w}$ is the inertia, $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ are two random numbers in the range [0,1], $\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ are cognitive and social components for the particle and are taken as 2.05 in this project, $\boldsymbol{P_{best}}$ is the global best position, $\boldsymbol{P_{pos}}$ is the position of the current paeticle, ans $\boldsymbol{S_{best}}$ is the personal best of the current particle.

Each particle's position is updated by adding its new velocity to its current position. This new position represents the next potential path from the start point to the end point.

It is calculated by the following equation (4.2):

$$\boldsymbol{P_{pos} = P_{pos} + P_{vel}} \quad \ldots(4.2)$$

$\boldsymbol{P_{pos}}$ is the position of the current particle and $\boldsymbol{P_{vel}}$ is its velocity.

4.2.6 Collision Check and Re-Evaluation

After updating positions Check if the new position collides with any obstacles.

- No Collision: If there is no collision, proceed to execute PSO again for further iterations.
- Collision Detected: If a collision is detected, the algorithm re-evaluates the same particle to find a valid position, and then updates the particle's best positions accordingly.

4.2.7 Execute GWO on the Swarm

Once PSO has converged or reached the maximum number of iterations, the GWO algorithm is executed to further refine the paths. The paths provided by PSO serve as the initial population for GWO.

4.2.8 Update Position of Wolves

In GWO, the three particles with the lowest path lengths are designated as the alpha, beta, and delta wolves at the beginning of each iteration. These wolves guide the search process, emulating the social hierarchy of grey wolves. After each iteration, alpha, beta, and delta wolves are reassigned based on the new lowest length particles.

The positions of the remaining wolves (particles) are updated based on their relative positions to the alpha, beta, and delta wolves. This process continues iteratively, allowing the algorithm to converge towards more optimal paths. The position is updated based on the following equations:

$$\vec{D}_\alpha = |\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}| \quad \ldots(4.3)$$

$$\vec{D}_\beta = |\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}| \quad \ldots(4.4)$$

$$\vec{D}_\delta = |\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}| \quad \ldots(4.5)$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \quad \ldots(4.6)$$

$$\vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \quad \ldots(4.7)$$

$$\vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \quad \ldots(4.8)$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \quad \ldots(4.9)$$

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a} \quad \ldots(4.10)$$

$$\vec{C} = 2\vec{r}_2 \quad \ldots(4.11)$$

$\boldsymbol{D_\alpha, D_\beta, D_\delta}$ are the distance vectors for $\boldsymbol{\alpha, \beta,}$ **and** $\boldsymbol{\delta}$ wolves respectively. $\boldsymbol{C_1, C_2,}$ **and** $\boldsymbol{C_3}$ have the values 2*r_2 where $\boldsymbol{r_2}$ is a random number in the range [0,1]. $\boldsymbol{X}$ is the position vector of the current wolf. a is initialized with 2 but decreases over each iteration. $\boldsymbol{r_1}$ is a random number between [0,1]. $\boldsymbol{X_1, X_2, X_3}$ are position vectors calculated by $\boldsymbol{\alpha, \beta,}$ **and** $\boldsymbol{\delta}$ wolves, the average of which is the new position of the current wolf.

4.2.9 Collision Check for GWO

Like PSO, each new position is checked for collisions with obstacles:

- No Collision: Continue the optimization process until the maximum number of iterations is reached.
- Collision Detected: Re-evaluate and update the position of the wolves as needed, ensuring valid paths are maintained.

4.2.10 Return the Path with Lowest Path Length

Finally, after completing the GWO iterations, the path with the lowest path length is returned as the optimal solution. This path represents the most efficient route from the start point to the end point, avoiding all obstacles.

Chapter 5

Coding / Code Templates

5.1 Importing libraries

```
import math
import obstacles as ob
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline
import time
```

5.2 Initializing the obstacles in the search space

```
obstacle1 = ob.Obstacle_Circle(5.5, ob.Point(6.5, 7.5))
obstacle2 = ob.Obstacle_Circle(3.33, ob.Point(17, 13))
obstacle3 = ob.Obstacle_Circle(3, ob.Point(11, 18))
obstacles = [obstacle1, obstacle2, obstacle3]
```

5.3 Obstacle class used to initialize the obstacles

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
#class to make circular obstacles
class Obstacle_Circle:
    def __init__(self, r, C):
        self.r = r #radius
        self.C = C #centre
    def inside_circle(self, p):
        return (p.x - self.C.x) ** 2 + (p.y - self.C.y) ** 2 <= self.r**2 + 1
    def how_to_exit_y(self, y):
        return y - self.C.y
    def how_to_exit_x(self, x):
        return x - self.C.x
```

5.4 Defining parameters for PSO

```
DIM = 13
START = ob.Point(0, 0)
END = ob.Point(15, 5)
max_iterations=150
swarm_size=150
max_vel=4,
step_size=1,
inertia=0.9,
c1=2.05,
c2=2.05,
```

5.5 Random Initialization of the particles in the swarm

```
def random_initialization(swarm_size):
    particles_loc = np.random.rand(swarm_size, DIM, 2)*25
    particles_vel = np.random.rand(swarm_size, DIM, 2)

    particles_lowest_loss = [
        loss_function(particles_loc[i, :, :]) for i in range(0, len(particles_loc))
    ]
    particles_best_location = np.copy(particles_loc)

    global_lowest_loss = np.min(particles_lowest_loss)
    global_best_location = particles_loc[np.argmin(particles_lowest_loss)].copy()

    return (
        particles_loc,
        particles_vel,
        particles_lowest_loss,
        particles_best_location,
        global_lowest_loss,
        global_best_location,
    )
```

5.6 Loss function used in random initialization

```
def loss_function(x):
    z = (x[0, 0] - START.x) ** 2 + (x[0, 1] - START.y) ** 2
    for i in range(DIM - 1):
        z = z + ((x[i, 0] - x[i + 1, 0]) ** 2 + (x[i, 1] - x[i + 1, 1]) ** 2)
    z = z + (x[DIM - 1, 0] - END.x) ** 2 + (x[DIM - 1, 1] - END.y) ** 2
    return math.sqrt(z)
```

5.7 Execution of the PSO algorithm

```
for iteration_i in range(max_iterations):
    if iteration_i % 30 == 0:
        print("%i%%" % int(iteration_i / max_iterations * 100))
        print_iteration += 1

    a=gb/gb_past
    b=(w_u+w_l)*a
    c=(w_u*iteration_i)/max_iterations
    w=b-c
    gb_past=gb
    for particle_i in range(swarm_size):
        dim_i = 0
        while dim_i < DIM:
            error_particle_best = (particles_best_location[particle_i, dim_i]- particles_loc[particle_i, dim_i])
            error_global_best = (global_best_location[dim_i] - particles_loc[particle_i, dim_i])
            new_vel = (
                w * min(1, (dim_i ** (0.5)) / 4)
                * particles_vel[particle_i, dim_i]
                + c1 * np.random.rand(1) * error_particle_best
                + c2 * np.random.rand(1) * error_global_best
            )
            if new_vel[0] < -max_vel:
                new_vel[0] = -max_vel
            elif new_vel[0] > max_vel:
                new_vel[0] = max_vel
            if new_vel[1] < -max_vel:
                new_vel[1] = -max_vel
            elif new_vel[1] > max_vel:
                new_vel[1] = max_vel
```

```
            particles_loc[particle_i, dim_i] = (particles_loc[particle_i, dim_i] + new_vel[:] * step_size)
            particles_vel[particle_i, dim_i] = new_vel[:]

            particle_help = is_valid(circles, particles_loc[particle_i, dim_i, :])
            particles_loc[particle_i, dim_i, 0] += particle_help.x
            particles_loc[particle_i, dim_i, 1] += particle_help.y

            if abs(particle_help.x) > 0.1 or abs(particle_help.y) > 0.1:
                dim_i -= 1
            dim_i += 1

        # for the new location, check if this is a new local or global best (if it's valid)
        particle_error = loss_function(particles_loc[particle_i, :])
        if particle_error < particles_lowest_loss[particle_i]:  # local best
            particles_lowest_loss[particle_i] = particle_error
            particles_best_location[particle_i, :] = particles_loc[particle_i, :].copy()
        if particle_error < global_lowest_loss:  # global best
            global_lowest_loss = particle_error
            global_best_location = particles_loc[particle_i, :].copy()

    best_location = global_best_location.copy()
    gb=global_lowest_loss
```

```
particle_path_lengths = [loss_function(particles_loc[i]) for i in range(swarm_size)]
sorted_particles_loc = [loc for _, loc in sorted(zip(particle_path_lengths, particles_loc))]
print("PSO executed")
print("Executing GWO...")
best_particle_gwo=gwo_optimization_adaptive(circles, sorted_particles_loc, max_iterations, step_size=1, start=START, end=END)

return best_location, best_particle_gwo
```

5.8 Checking the updated position for collision

```
def is_valid(circles, p):
    to_add = ob.Point(0, 0)
    point_p = ob.Point(p[0], p[1])
    for i in range(len(circles)):
        if circles[i].inside_circle(point_p):
            to_add = ob.Point(
                circles[i].how_to_exit_x(point_p.x) + to_add.x,
                circles[i].how_to_exit_y(point_p.y) + to_add.y,
            )
    return to_add
```

5.9 Execution of the GWO algorithm

```
for iteration in range(max_iterations):
    a=2*(1-iteration/max_iterations)
    if iteration % 30 == 0:
        print("%i%%" % int(iteration / max_iterations * 100))
    alpha_wolf = updated_par[0]
    beta_wolf = updated_par[1]
    gamma_wolf = updated_par[2]
    for i in range(3, num_particles):
        particle = updated_par[i]
        j=0
        while j < dims:
            for k in range(2):
                r1 = np.random.rand()
                r2 = np.random.rand()
                A = 2 * a * r1 - a  # Parameter A
                C = 2 * r2  # Parameter C
                # Alpha Wolf
                D_alpha = np.abs(C * alpha_wolf[j][k] - particle[j][k])
                X1 = (alpha_wolf[j][k] - A * D_alpha)
                # Beta Wolf
                D_beta = np.abs(C * beta_wolf[j][k] - particle[j][k])
                X2 = (beta_wolf[j][k] - A * D_beta)
                # Gamma Wolf
                D_gamma = np.abs(C * gamma_wolf[j][k] - particle[j][k])
                X3 = (gamma_wolf[j][k] - A * D_gamma)
                # Update the position
                particle[j][k] = ((X1 + X2 + X3) / 3)*step_size

                # Ensure the new position is valid (doesn't collide with obstacles)
                obstacle_correction = is_valid(obstacles, particle[j])
                particle[j][0] += obstacle_correction.x
                particle[j][1] += obstacle_correction.y
                if abs(obstacle_correction.x) > 0.1 or abs(obstacle_correction.y) > 0.1:
                    k -= 1
            j=j+1
```

```
            fitness = fitness_function(particle, start, end)
            if fitness < best_fitness:
                best_fitness = fitness
                best_particle = particle.copy()
        particle_path_lengths = [loss_function(updated_par[i]) for i in range(swarm_size)]
        updated_par = [loc for _, loc in sorted(zip(particle_path_lengths, updated_par))]
```

```
total_path_length = math.sqrt((best_particle[0][0] - start.x) ** 2 + (best_particle[0][1] - start.y) ** 2)
for i in range(len(best_particle) - 1):
    x1, y1 = best_particle[i]
    x2, y2 = best_particle[i + 1]
    total_path_length += math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
total_path_length += math.sqrt((best_particle[-1][0] - end.x) ** 2 + (best_particle[-1][1] - end.y) ** 2)
print("Total Path Length with PSO+GWO (fixed inertial weight):", round(total_path_length, 2))

plot_graph(best_particle, "PSO+GWO (fixed inertia)", str(round(total_path_length,2)), "red")
for obstacle in obstacles:
    plot_circle(obstacle)
plt.savefig("result_pso_gwo", dpi=300)
return best_particle
```

5.10 Plotting the graphs

```
def plot_graph(best_location, graph, len, color):
    plt.close("all")
    plt.figure(figsize=(5, 5))
    plt.grid("on")
    plt.rc("axes", axisbelow=True)
    plt.scatter(END.x, END.y, 100, marker="*", facecolors="k", edgecolors="k")
    plt.scatter(START.x, START.y, 100, marker="o", facecolors="k", edgecolors="k")

    plot_smooth_path(best_location, color)

    for i in range(DIM):
        plt.scatter(
            best_location[i][0],
            best_location[i][1],
            25,
            marker="o",
            facecolors="blue",
            edgecolors="face",
        )

    plt.xlim(-1, 25)
    plt.ylim(-1, 25)
    plt.title(graph+": "+len+" units")
```

5.11 Plotting the obstacles

```
def plot_circle(circle):
    plt.gca().add_patch(
        plt.Circle(
            (circle.C.x, circle.C.y), circle.r - 0.2, edgecolor="black", facecolor="grey"
        )
    )
```

5.12 Plotting the splines

```
def plot_smooth_path(best_location, color):
    X = [point[0] for point in best_location]
    Y = [point[1] for point in best_location]

    X = [START.x] + X + [END.x]
    Y = [START.y] + Y + [END.y]

    spline = CubicSpline(np.arange(len(X)), X)
    smooth_X = spline(np.linspace(0, len(X) - 1, 500))
    spline = CubicSpline(np.arange(len(Y)), Y)
    smooth_Y = spline(np.linspace(0, len(Y) - 1, 500))

    plt.plot(smooth_X, smooth_Y, color, label='Smooth Path')
```

Chapter 6

Testing

A total of five different destinations were checked in the presence of three circular obstacles. Four different approaches were tested on the same parameters for their suitability. The start coordinates were taken as (0,0) for all the simulations. The path lengths for each of the approaches is given in the below table:

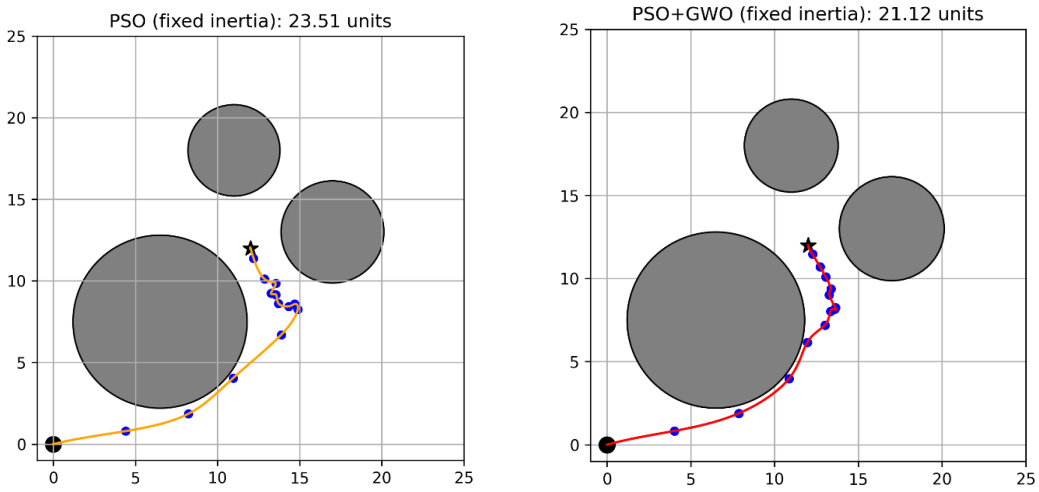

Fig 6.1 Path to (12,12) of PSO and PSO+GWO with fixed inertia respectively

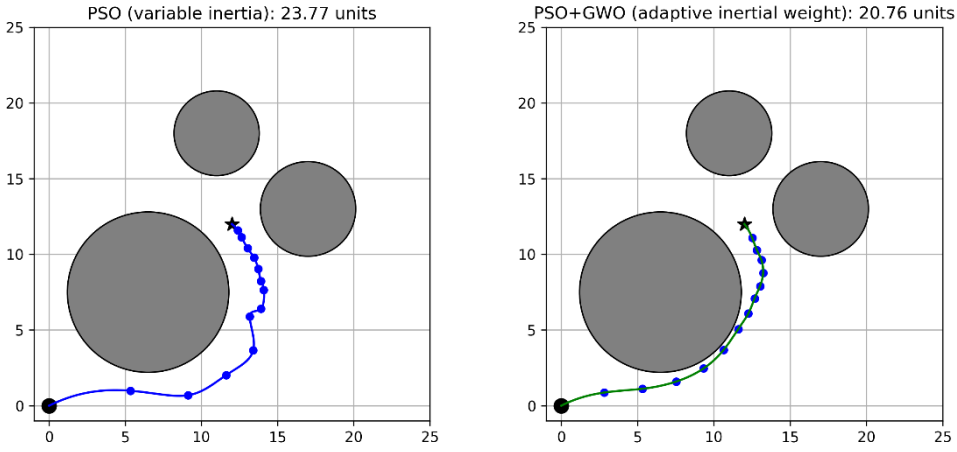

Fig 6.2 Path to (12,12) of PSO with variable inertia and PSO+GWO with adaptive inertia respectively

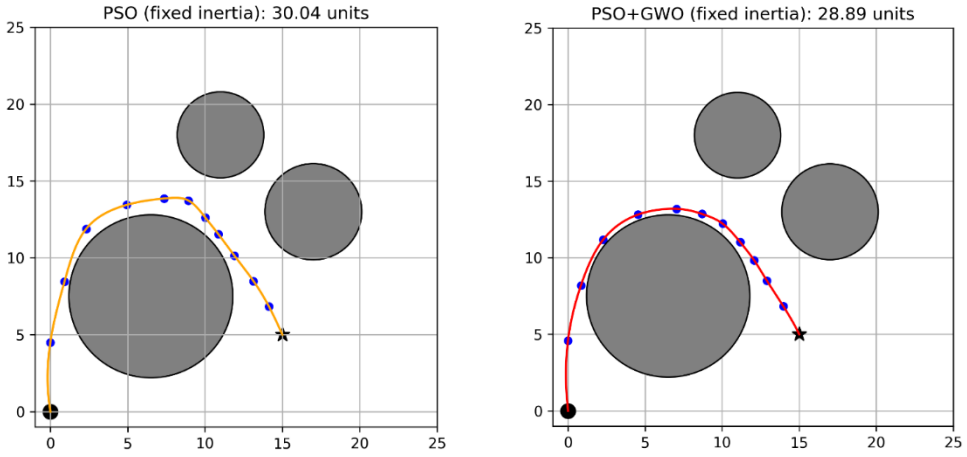

Fig 6.3 Path to (15,5) of PSO and PSO+GWO with fixed inertia respectively

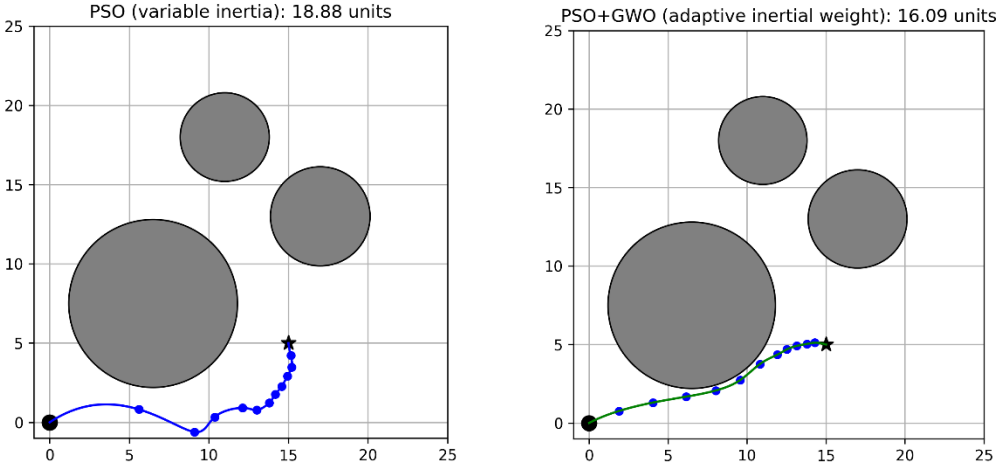

Fig 6.4 Path to (15,5) of PSO with variable inertia and PSO+GWO with adaptive inertia respectively

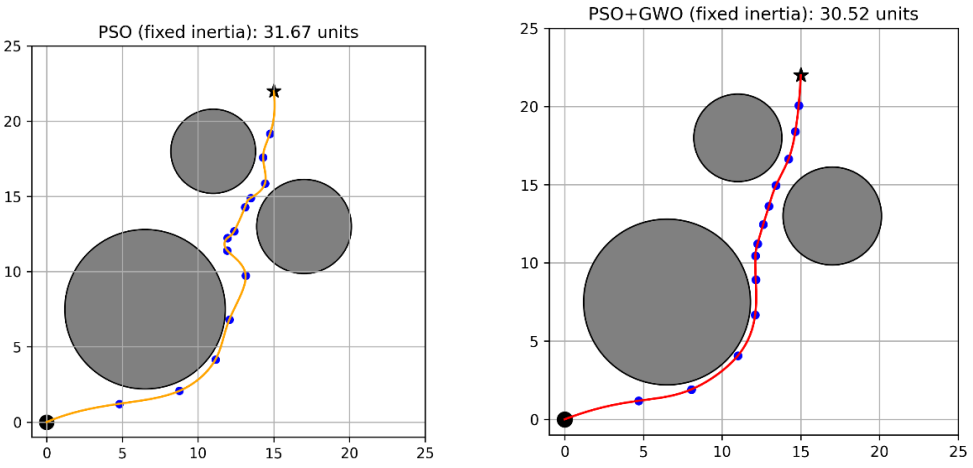

Fig 6.5 Path to (15, 22) of PSO and PSO+GWO with fixed inertia respectively

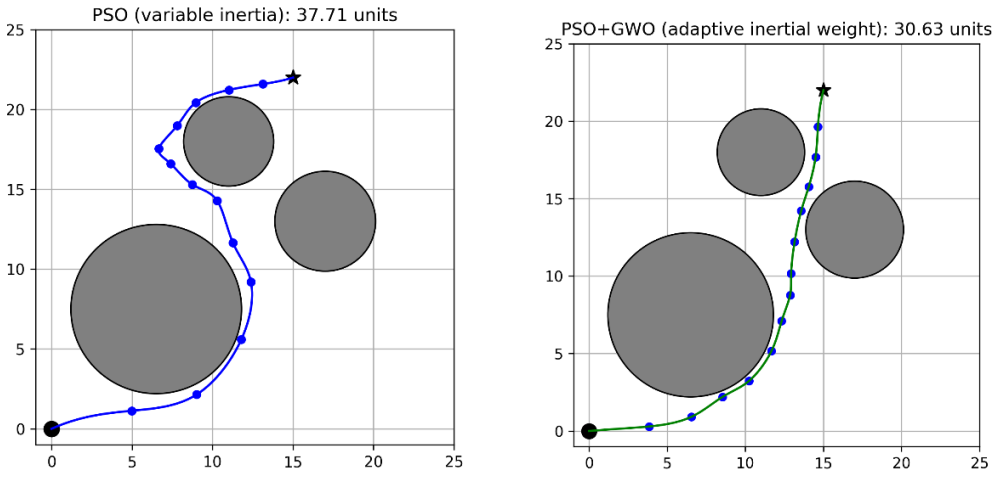

Fig 6.6 Path to (15,22) of PSO with variable inertia and PSO+GWO with adaptive inertia respectively

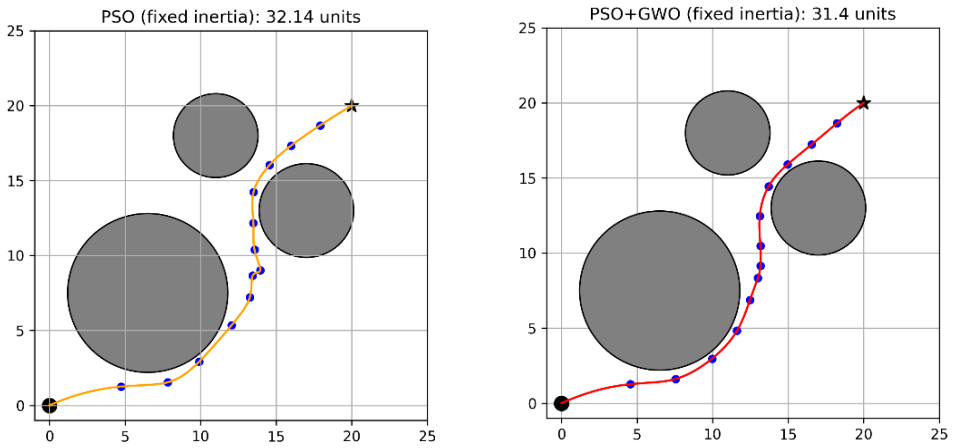

Fig 6.7 Path to (20,20) of PSO and PSO+GWO with fixed inertia respectively

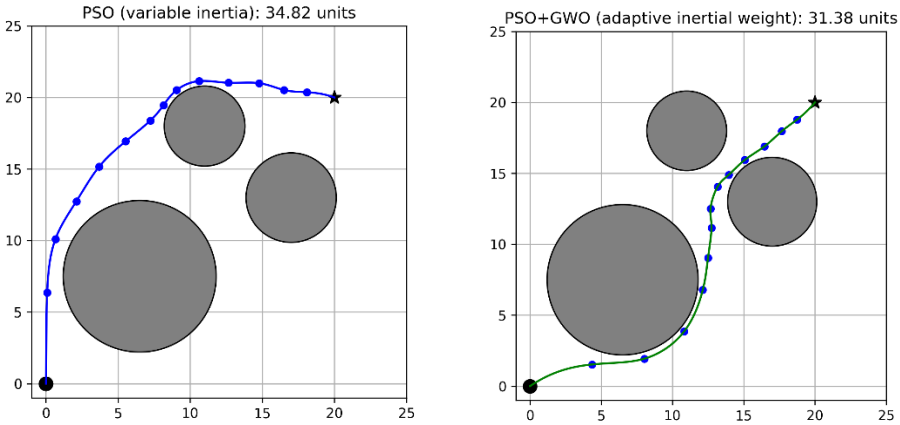

Fig 6.8 Path to (20,20) of PSO with variable inertia and PSO+GWO with adaptive inertia respectively

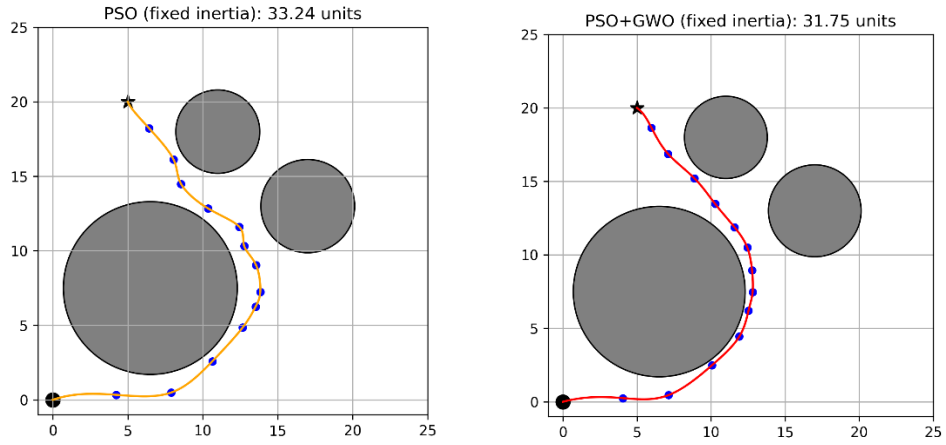

Fig 6.9 Path to (5, 20) of PSO and PSO+GWO with fixed inertia respectively

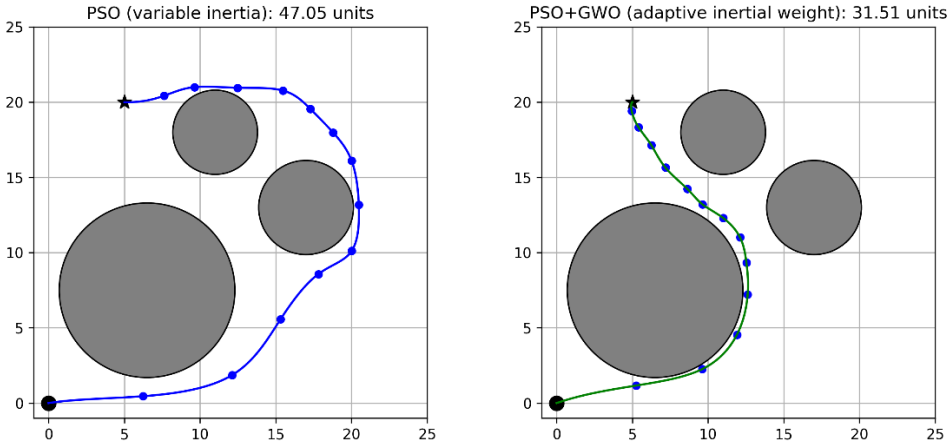

Fig 6.10 Path to (5,20) of PSO with variable inertia and PSO+GWO with adaptive inertia respectively

Chapter 7

Results and Discussion

The results of this study aimed to show the efficiency of the PSO+GWO concatenated approach with adaptive inertia in path planning from source to destination while avoiding all the obstacles in the search space as compared to the traditional PSO algorithm.

The concatenated approach produced path lengths that were shorter by 2 to 3 units as compared to the traditional PSO approach. The path produced by the concatenated approach was also smoother with a huge decrease in the number of abrupt and sharp corners in the path. This shows a smooth and seamless path.

Destination	PSO with fixed inertia (units)	PSO+GWO with fixed inertia (units)	PSO with variable inertia (units)	PSO+GWO with adaptive inertia (units)
(5,20)	30.08	28.69	32.95	26.78
(15,5)	21.48	20.10	24.7	18.81
(12,12)	24.28	21.23	25.6	20.86
(15,22)	32.42	30.91	34.88	30.73
(20,20)	32.58	31.81	33.13	31.92

Table 7.1 Path lengths of various approaches with different testing parameters

7.1 Producing a smoother path

The below figure 7.1 show that the hybrid approach produced a much smoother path with lesser number of haphazard turns.

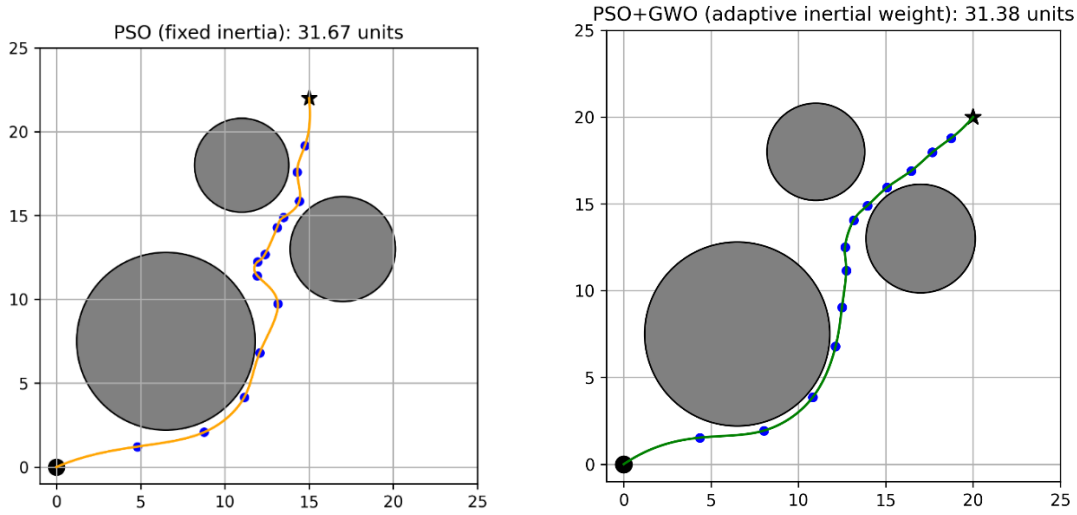

Fig 7.1 Reduction in path length as well as a much more refined path.

7.2 Performance on corner cases

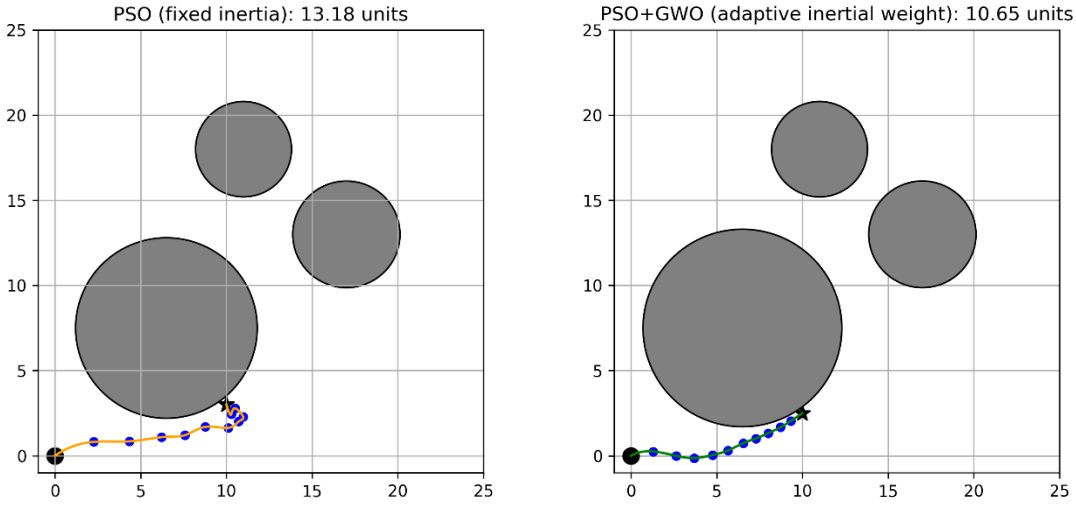

Fig 7.2 Path from (0,0) to (10,2.5)

In certain corner case, as show in in above figure 7.2, where the goal is very close to the start, the traditional PSO didn't produce an optimal path but the PSO+GWO approach gave a much better and optimized path.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

From the results of this study, it can be concluded that the proposed adaptation of PSO and GWO is advisable and useful in solving the problem of path planning for autonomous robots. PSO was used for the fast search for good solutions while GWO was used to fine-tune them, thus we proposed a powerful hybrid algorithm that generates short optimal routes in a timely manner. The unconventional nature of paths identified by the PSO algorithm allows for a rapid exploration of the search space and delivers a set of potential paths in a short time, while GWO optimizes the selection of such paths in terms of travelled distance and obstacle avoidance. This concatenated approach of optimization is very useful as it makes full use of both these algorithms, which although work in similar ways, have their pros and cons and when use together, yields much better results.

The findings of this study provide the insight that the concatenated approach of PSO with GWO along with adaptive weight adjustment performed better than the traditional PSO as well as some other methods with different adjustment techniques.

8.2 Future Work

Future enhancements could include:

- Incorporating dynamic obstacles: Add obstacles that move over time to simulate more realistic environments.
- Expanding grid size: Increase the grid dimensions to simulate larger environments for broader application testing.
- Integrating additional algorithms: Compare the hybrid PSO-GWO approach with other optimization algorithms for comprehensive analysis.
- Implementing real-time adjustments: Enable the algorithm to adjust paths instantaneously in response to environmental changes.

Details of Research Publication

The details of our research publication are as follows:

References

[1] K. Hao, J. L. Zhao, B. B. Wang, Y. L. Liu, and C. Q. Wang, "The application of an adaptive genetic algorithm based on collision detection in path planning of mobile robots," *Computational Intelligence and Neuroscience*, vol. 2021, 2021.

[2] Z. Zhang, Y. Wan, Y. Wang, X. Guan, W. Ren, and G. Li, "Improved hybrid A* path planning method for spherical mobile robot based on pendulum," in *International Journal of Advanced Robotic Systems*, vol. 18, no. 1, 2021.

[3] C. Miao, G. Chen, C. Yan, and Y. Wu, "Path planning optimization of indoor mobile robot based on adaptive ant colony algorithm," *Computers & Industrial Engineering*, vol. 156, pp. 107230, 2021.

[4] R. Sarkar, D. Barman, and N. Chowdhury, “Domain knowledge based genetic algorithms for mobile robot path planning having single and multiple targets,” *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 7, 2022.

[5] Örnek BN, Aydemir SB, Düzenli T, Özak B. "A novel version of slime mould algorithm for global optimization and real-world engineering problems: Enhanced slime mould algorithm." *Mathematics and Computers in Simulation*, vol. 198, pp. 253-288, Aug. 2022.

[6] X. B. Cheng, L. C. Zhu, H. H. Lu, J. Z. Wei, and N. Wu, "Robot path planning based on an improved salp swarm algorithm," *Journal of Sensors*, vol. 2022, Article ID 2559955, 16 pages, 2022.

[7] Yu, Z. Si, X. Li, D. Wang, and H. Song, "A novel hybrid particle swarm optimization algorithm for path planning of UAVs," *IEEE Internet of Things Journal*, vol. 9, no. 22, pp. 22547–22558, 2022.

[8] Xun Li, Dandan Wu, Jingjing He, Muhammad Bashir, Ma Liping, "An Improved Method of Particle Swarm Optimization for Path Planning of Mobile

Robot", *Journal of Control Science and Engineering*, vol. 2020, Article ID 3857894, 12 pages, 2020.

[9] Cheng X, Li J, Zheng C, Zhang J and Zhao M (2021) An Improved PSO-GWO Algorithm With Chaos and Adaptive Inertial Weight for Robot Path Planning. *Front. Neurorobot.* 15:770361.