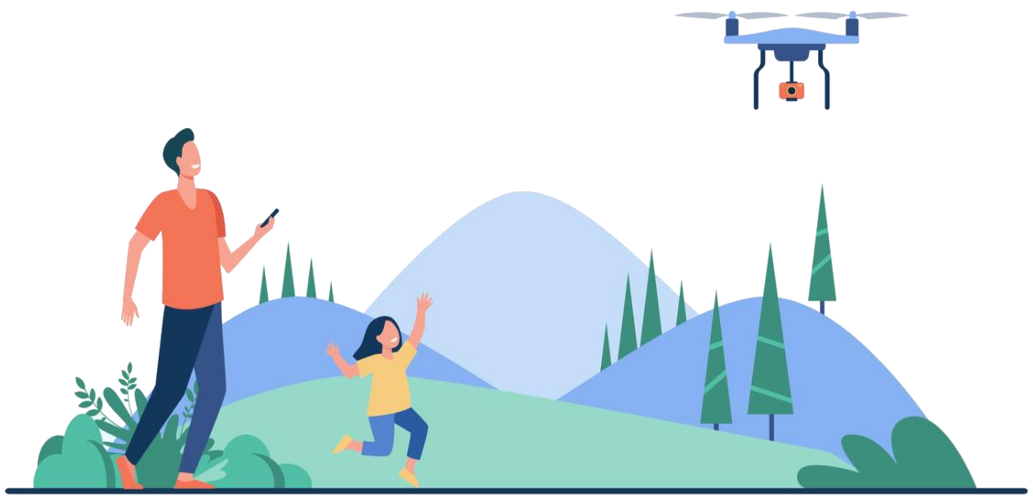


MEDICAL DELIVERY DRONE

Delivery of Medical equipment with the help of drones in Hilly terrain using Unsupervised Machine Learning and Google map APIs.

CONTENTS

ABSTRACT

This project presents a drone simulation that employs neuro-evolution to enable autonomous obstacle avoidance. The simulation is built using Python and integrates a genetic algorithm to optimize the drone's decision-making process over multiple generations. The learning process is based on natural selection, where the best-performing drones from each generation are chosen and their parameters are used to create improved versions through mutation.

The simulation consists of two main components: Game.py, which provides a graphical environment where drones navigate through obstacles, and drone.py, which implements the genetic algorithm responsible for training the drone. The fitness of each drone is determined by a function that considers the distance travelled and the drone's alignment with obstacle gaps. The evolution process continues until a highly efficient drone emerges.

This project demonstrates the power of artificial intelligence and evolutionary computation in training autonomous agents. It has potential applications in robotics, drone navigation, and reinforcement learning research.

INTRODUCTION

The demand for fast and reliable delivery of medical supplies has increased due to various factors, including population growth, pandemics, and natural disasters. Conventional transportation methods face several challenges, including traffic congestion, poor road infrastructure, and geographical barriers. These inefficiencies often result in delayed medical aid, leading to adverse health outcomes, especially in critical emergencies.

In recent years, Unmanned Aerial Vehicles (UAVs), commonly known as drones, have emerged as a revolutionary solution to healthcare delivery challenges. Drones provide an innovative means of overcoming logistical barriers by enabling rapid, autonomous transportation of medical supplies to hard-to-reach locations. With advancements in artificial intelligence, battery technology, and GPS-based navigation, drones can now be deployed for highly optimized, automated delivery routes. Many countries have started exploring drone-based medical logistics to address last-mile delivery problems and improve access to essential healthcare services.

The objective of this project is to develop a robust drone path simulation framework that accounts for critical variables such as battery limitations, obstacle avoidance, and regulatory constraints to ensure the feasibility of drone-assisted medical deliveries. This project will simulate real-world conditions to evaluate the efficiency, safety, and reliability of drone-based logistics. By integrating state-of-the-art pathfinding algorithms, the system will help identify the most efficient routes, ensuring that drones deliver medical supplies in the shortest possible time while minimizing risks.

Additionally, this simulation will assess the impact of different environmental and operational factors on drone performance. Variables such as terrain complexity, wind speed, no-fly zones, and airspace restrictions will be modeled to develop a realistic framework for drone navigation. The insights derived from this project can inform policy decisions, infrastructure planning, and technological advancements needed to scale up drone-based medical supply chains worldwide

PROBLEM STATEMENT

Timely access to medical supplies is a critical factor in saving lives, especially in emergencies. However, many remote and rural areas lack the infrastructure necessary for rapid medical logistics. Traditional transportation systems often face several inefficiencies, including slow delivery times, high operational costs, and vulnerability to external disruptions such as traffic congestion, roadblocks, and natural disasters. These delays can be life-threatening, particularly in urgent medical scenarios requiring immediate attention.

The inefficiencies in the current medical supply chain highlight the need for a faster, more reliable alternative. The integration of drones into medical logistics presents a promising solution to address these challenges. Drones can significantly reduce delivery times, bypass road-related obstacles, and provide a more cost-effective transportation method for medical goods. However, implementing drone-based logistics comes with its own set of challenges, such as optimizing flight paths, ensuring regulatory compliance, and managing environmental constraints.

This project aims to develop an optimized simulation model that enhances the efficiency of medical drone delivery. The model will focus on refining navigation algorithms to determine the most effective routes while considering factors such as weather conditions, airspace restrictions, and battery limitations. By testing various scenarios within a simulation framework, this research will provide critical insights into the feasibility and scalability of drone-assisted healthcare logistics. The ultimate goal is to develop a system that ensures medical supplies reach their destinations in the shortest possible time while overcoming logistical challenges and minimizing risks.

SCOPE OF THE PROJECT

T he scope of this project is limited to the simulation and optimization of drone flight paths for medical deliveries. It does not involve the physical implementation of drones or hardware testing. Instead, the project aims to create a software-based simulation environment that evaluates real-world constraints and optimizes drone navigation to ensure the efficient delivery of medical supplies.

The key areas covered in this project include:

- Geographical Challenges: The simulation will incorporate complex terrain features such as mountains, rivers, forests, and urban structures to analyze the impact of environmental factors on drone navigation.
- Weather Conditions: It will assess the effect of wind speed, rain, temperature, and atmospheric pressure on drone performance.
- Regulatory Constraints: The project will account for no-fly zones, air traffic regulations, and legal compliance requirements to ensure safe and lawful drone operations.
- Drone Limitations: Battery life, payload capacity, communication range, and flight endurance will be considered when designing flight paths to optimize performance and reliability.

By focusing on these key areas, the project will provide valuable insights into the feasibility of medical drone deliveries. The findings from this research will contribute to developing optimized, scalable, and data-driven solutions for real-world implementation in healthcare logistics.

METHODOLOGY

The project follows a structured approach to simulate and optimize drone navigation. The methodology includes:

1. Algorithm Development: Implementing advanced pathfinding algorithms such as A* and Dijkstra's algorithm to determine the most efficient routes.
2. Simulation Framework: Using MATLAB, Python, and GIS data to model realistic terrain conditions and airspace constraints.
3. Data Integration: Incorporating real-time weather data and regulatory guidelines to improve simulation accuracy.
4. Validation: Running multiple test scenarios under varying conditions to evaluate drone performance and optimize flight paths.
5. Performance Analysis: Assessing key metrics such as energy consumption, delivery speed, and obstacle avoidance to enhance the efficiency of drone logistics.

This methodology ensures that the simulation closely represents real-world scenarios, allowing researchers and developers to refine drone-based medical logistics before practical implementation.

SYSTEM REQUIREMENTS

- **Software Requirements**:
  - MATLAB for simulation modeling.
  - Python for algorithm development.
  - GIS software for mapping and terrain analysis.
  - Machine learning libraries for predictive analysis.
- **Hardware Requirements**:
  - High-performance computing system (16GB RAM, multi-core processor).
  - Cloud storage for handling large datasets.
  - High-resolution display for visualization.

IMPLEMENTATION

Drone.py-Python (backend)

```
import pygame
import random
import math
import os
import numpy as np

'''
Game Variables
'''

WHITE = (255,255,255)
BLACK = (0,0,0)
RED = (255,0,0)
GREEN = (0,100,0)

display_width = 1400            #Game Window Width
display_height = 800            #Game Window Height
FPS = 60                        #Game FPS
pipe_gap = 200                  #Manages the pipe gap between the upper and bottom pipe
between_pipe = 200              #Manages the distance between consecutive sets of pipe
pipe_width = 100                #Manages the width of each set of pipe
pipe_speed = 6                  #Manages the speed with which the pipe move towards the bird
score = 0                       #Init the score
velocity = 10                   #Velocity of the Bird
```

```
pipe_count = display_width//(pipe_width+between_pipe)+2              #Calculates the the
number of pipes that would
                                                     #fit the screen.


'''
Game Variables
'''


'''
Genetic Variables
'''


population = 350    # Total Population
hidden_nodes = 8    # Number of Nodes inside the hidden layer
inp = 4             # Number of inputs + Bias
bias1 = np.random.uniform() # Input Layer Bias
bias2 = np.random.uniform() # Hidden Layer Bias
'''
Inputs :
1. Velocity of the Bird
2. Distance between the center of the drone and the center of the gap of the pipe vertically
3. Distance between the center of the drone and the center of the gap of the pipe horizontally
4. Bias
'''


game_folder = os.path.dirname(__file__)


master_parameters =
[np.zeros(shape=(inp,hidden_nodes)),np.zeros(shape=hidden_nodes+1)]
```

```
master_parameters = np.asarray(master_parameters)  # Will be used to store the best drone
and mutate it.


'''
Genetic Variables
'''


'''
Genetic Algorithm
'''


def sigmoid(value):
    value = float(math.exp(-value))
    value = float(value + 1)
    value = float(1/value)
    return value


def nn(arr,paras,bias2):
    hidden_activations = np.dot(arr,paras[0])
    hidden_activations = [bias2] + list(map(sigmoid,hidden_activations))
    return sigmoid(np.dot(hidden_activations,paras[1]))


def mutate(master):
    mutation = np.random.normal(scale=1)
    return (master+mutation)


def make_parameters(master,population):
    para_list = [master]
    for make in range(population-1):
        para_list.append(mutate(np.asarray(master)))
    return para_list


'''
Genetic Algorithm
'''


'''
Pygame Sprite Classes
'''


class Bird(pygame.sprite.Sprite):
    """ Will contain the bird attributes.
        Args:
        x_loc (int): X - coordinate of the center of the drone sprite
        y_loc (int): Y - coordinate of the center of the drone sprite
        velocity (int): Velocity of the drone sprite. """
    def __init__(self, x_loc, y_loc, velocity):
        super(Bird, self).__init__()
        self.check = 0
        self.velocity = velocity
        self.x_loc = x_loc
        self.y_loc = y_loc
        self.image = pygame.image.load(os.path.join(game_folder,"index.png")).convert()
        self.image.set_colorkey(WHITE)
        self.image = pygame.transform.scale(self.image,(40,40))
        self.rect = self.image.get_rect()
        self.rect.center = (x_loc,y_loc)
        self.mask = pygame.mask.from_surface(self.image)
    def update(self):
        self.rect.y += self.velocity
        self.velocity = self.velocity+1
    def jump(self):
        self.velocity = -10
    def boundary_collison(self):
        if self.rect.bottom+100>=display_height or self.rect.top<=0:
            return True
    def bird_center(self):
        return self.rect.center
    def vel(self):
        return velocity



class UpperPipe(pygame.sprite.Sprite):
    """ Will contain the upper pipe's attributes.
        Args:
        pipe_x (int): X - coordinate of the starting of the pipe
        pipe_height (int): Height of the upper pipe
        pipe_speed (int): Pipe speed with which they pipe's will move horizontally. """
    def __init__(self, pipe_x, pipe_height, pipe_speed):
        super(UpperPipe, self).__init__()
        self.pipe_speed = pipe_speed
        self.pipe_height = pipe_height
        self.image = pygame.Surface((pipe_width, pipe_height))
        self.image.fill(GREEN)
        self.image.set_colorkey(WHITE)
        self.rect = self.image.get_rect()
        self.rect.x = (pipe_x)
        self.rect.y = (0)
        self.mask = pygame.mask.from_surface(self.image)
    def update(self):
        self.rect.x -= self.pipe_speed
    def x_cord(self):
        return self.rect.x
    def y_cord(self):
        return (self.rect.y+self.pipe_height)

class LowerPipe(pygame.sprite.Sprite):
    """ Will contain the lower pipe's attributes.
        Args:
        pipe_x (int): X - coordinate of the starting of the pipe
        pipe_height (int): Height of the lower pipe
        pipe_speed (int): Pipe speed with which they pipe's will move horizontally. """
    def __init__(self, pipe_x, pipe_height, pipe_speed):
        super(LowerPipe, self).__init__()
        self.pipe_speed = pipe_speed
        self.image = pygame.Surface((pipe_width, display_height-(pipe_gap+pipe_height)))
        self.image.fill(GREEN)
        self.image.set_colorkey(WHITE)
        self.rect = self.image.get_rect()
        self.rect.x = (pipe_x)
        self.rect.y = (pipe_height+pipe_gap)
        self.mask = pygame.mask.from_surface(self.image)
    def update(self):
        self.rect.x -= self.pipe_speed
    def x_cord(self):
        return self.rect.x
    def y_cord(self):
        return self.rect.y


'''
Pygame Sprite Classes
'''


# Will create a new set of upper and lower pipe everytime an existing
# pipe moves off the screen.
def new_pipe(pipe):
    pipe_x = pipe[0].x_cord()+between_pipe+pipe_width
    pipe_height = (round(np.random.uniform(0.15,0.85), 2))*(display_height-pipe_gap)
    upper = UpperPipe(pipe_x,pipe_height,pipe_speed)
    lower = LowerPipe(pipe_x,pipe_height,pipe_speed)
    add_pipe = [upper,lower]
    pipe_group.add(upper)
    pipe_group.add(lower)
    return(add_pipe)


def init_para():
    parameter_list = []
    for iii in range(population):
        m_parameters =
[np.random.normal(size=(inp,hidden_nodes)),np.random.normal(size=hidden_nodes+1)]
        m_parameters = np.asarray(m_parameters)
        parameter_list.append(m_parameters)
    return parameter_list


def init_bias():
    bias = np.random.normal(size = (population,2))
    return bias


'''
Main Game Run Function
'''


def run_game(generation,score,bias_list):
    myfont = pygame.font.SysFont("monospace", 16)
    run_score = 0
    best_score = []
    best_bias = []
    manage = []
    global check
    global gameExit
    global master_parameters
    global pipe_collision
    global pipe
    cur_index = 0
    while not gameExit:
        clock.tick(FPS)
        gameDisplay.fill(WHITE)
        for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        gameExit = True
                    if event.key == pygame.K_SPACE:
                        for bird_index in range(len(bird_list)):
                            if check[bird_index]==0:
```

```
                bird = bird_list[bird_index]
                bird.jump()
                if bird.boundary_collison():
                    pygame.sprite.Sprite.kill(bird)
                    check[bird_index] = 1
                    return parameter_list[bird_index],run_score,bias_list[bird_index]
                for x in pipe_collision:
                    c=0
                    if pygame.sprite.collide_rect(bird,x):
                        bird_hits =
pygame.sprite.spritecollide(bird,pipe_group,False,pygame.sprite.collide_mask)
                        if bird_hits:
                            c = 1
                            pygame.sprite.Sprite.kill(bird)
                            check[bird_index] = 1
                            return parameter_list[bird_index],run_score,bias_list[bird_index]



    for bird_index in range(len(bird_list)):


        if check[bird_index]==0:
            bias1,bias2 = bias_list[bird_index][0],bias_list[bird_index][1]
            bird = bird_list[bird_index]
            arr = [bias1]
            bird_x,bird_y = bird.bird_center()
            arr.append(bird.vel())
            xpip = (pipe_collision[0].x_cord())
            draw_x,draw_y = 0,0
            '''
```

Parameter Selection Code

```
For each pipe the input feature will change as the drone moves through it
>>First input feature will be the middle of the starting of the gap.
>>After change the second input feature will be the middle point at the
  middle of the gap.
>>Finally its going to change into the middle point at the end of the gap.
'''
if bird_x <= xpip:
    xpip = (pipe_collision[0].x_cord())/(display_width)
    ypip = (pipe_collision[0].y_cord()+(pipe_gap//2))/(display_height)
    arr.append((bird_x/display_width)-xpip)
    arr.append((bird_y/display_height)-ypip)
    draw_x = pipe_collision[0].x_cord()
    draw_y = pipe_collision[0].y_cord()+(pipe_gap//2)
elif (bird_x) <= (xpip+(pipe_width//2)):
    xpip = (pipe_collision[0].x_cord()+(pipe_width//2))/(display_width)
    ypip = (pipe_collision[0].y_cord()+(pipe_gap//2))/(display_height)
    arr.append((bird_x/display_width)-xpip)
    arr.append((bird_y/display_height)-ypip)
    draw_x = pipe_collision[0].x_cord()+(pipe_width//2)
    draw_y = pipe_collision[0].y_cord()+(pipe_gap//2)
else:
    xpip = (pipe_collision[0].x_cord()+pipe_width)/(display_width)
    ypip = (pipe_collision[0].y_cord()+(pipe_gap//2))/(display_height)
    arr.append((bird_x/display_width)-xpip)
    arr.append((bird_y/display_height)-ypip)
    draw_x = pipe_collision[0].x_cord()+pipe_width
    draw_y = pipe_collision[0].y_cord()+(pipe_gap//2)
'''
```

```
            Parameter Selection Done
            '''
            pygame.draw.circle(gameDisplay,RED,(draw_x,int(draw_y)),5)
            direct_distance = (arr[1]**2)+(arr[2]**2)
            direct_distance = math.sqrt(direct_distance)
            fitness = run_score - direct_distance
            out = nn(arr,parameter_list[bird_index],bias2)
            if out>0.5:
                bird.jump()



            '''
            This section checks for collison of the drone with a boundary
            or the currently selected pair of pipes
            '''


            # -----------Boundary Collision-----------
            if bird.boundary_collison():
                pygame.sprite.Sprite.kill(bird)
                best_score.append(fitness)
                best_bias.append([bias1,bias2])
                check[bird_index] = 1
            # -----------Pipe Collision-----------
            for x in pipe_collision:
                c=0
                if pygame.sprite.collide_rect(bird,x) and check[bird_index]==0:
                    bird_hits =
pygame.sprite.spritecollide(bird,pipe_group,False,pygame.sprite.collide_mask)
                    if bird_hits:
```

```
                c = 1
                pygame.sprite.Sprite.kill(bird)
                best_score.append(fitness)
                best_bias.append([bias1,bias2])
                check[bird_index] = 1
                break
            if c==1:
                break

    '''
    Once all the drone are dead this part removes the best parameters
    and returns those back so that they can be mutated
    '''
    if sum(check)==len(check):
        bias_return = []
        if max(best_score)>score:
            try:
                master_parameters = parameter_list[list(best_score).index(max(best_score))]
                score = max(best_score)
                bias_return = bias_list[list(best_bias).index(max(best_bias))]
            except:
                print("Debug Check 1",sum(check))
                print("Debug Check 1",len(best_score))
                pygame.quit()
                quit()
        return master_parameters,score,bias_return

'''
Game Window Updates
```

```
'''
sprites.update()
pipe_group.update()
sprites.draw(gameDisplay)
pipe_group.draw(gameDisplay)
'''
Game Window Updates
'''



'''
Delete the pipe that has left the screen
and add a new one in its place.

Also Update the current pipe to check for collison of the drone
with a pipe
'''
if (pipe[0].x_cord())+pipe_width <= 0:
    for k in pipe:
        pygame.sprite.Sprite.kill(k)
    pipe = pipe_list[1]
    del pipe_list[0]
    pipe_list.append(new_pipe(pipe_list[-1]))



if ((pipe_collision[0].x_cord()+pipe_width)<x_loc):
    pipe_collision = pipe_list[1]
'''
Pipe Updates
```

```
    '''

    gen = myfont.render("Genertation {0}".format(generation), 1, (0,0,0))
    highest = myfont.render("Highest Score {0}".format(int(round(score))), 1, (0,0,0))
    current = myfont.render("Current Score {0}".format(run_score), 1, (0,0,0))
    gameDisplay.blit(gen, (5, 10))
    gameDisplay.blit(highest, (5, 35))
    gameDisplay.blit(current, (5, 60))
    run_score += 1
    pygame.display.flip()

'''
Main Game Run Function
'''

score = 0   # Starting Score Initialization.
generation = 1  # Generation Initialization.

parameter_list = init_para() #Describing unique parameters for every bird in the population.
bias_list = init_bias()

'''
Main control loop -
Will be responsible to run the game multiple times and
mutate the best drone after the end of each game
'''
threshold_score = 0
threshold = 5
threshold_count = 0
```

```
bias = []
while True:
    pygame.init()
    gameDisplay = pygame.display.set_mode((display_width,display_height))
    clock = pygame.time.Clock()
    pygame.display.set_caption("Drone Simulation- NeuroEvolution")

    x_loc = display_width//8
    y_loc = display_height//2

    sprites = pygame.sprite.Group()
    pipe_group = pygame.sprite.Group()
    y_locations = np.random.randint(low = 0,high=display_height,size=population)
    bird_list = []
    for make_bird in range(population):
        bird = Bird(x_loc,y_locations[make_bird],velocity)
        bird_list.append(bird)
        sprites.add(bird)

    pipe_list = []
    init_pipe_x = 500
    for make in range(pipe_count):
        pipe_x = init_pipe_x+((between_pipe+pipe_width)*make)
        pipe_height = (round(np.random.uniform(0.15,0.85), 2))*(display_height-pipe_gap)
        upper = UpperPipe(pipe_x,pipe_height,pipe_speed)
        lower = LowerPipe(pipe_x,pipe_height,pipe_speed)
        add_pipe = [upper,lower]
        pipe_list.append(add_pipe)
        pipe_group.add(upper)
        pipe_group.add(lower)
    pipe = pipe_list[0]
    pipe_collision = pipe_list[0]
    gameExit = False
    check=[]
    for i in range(population):
        check.append(0)
    cpy_mp,cpy_score,cpy_bias = master_parameters,score,bias
    master_parameters,score,bias = run_game(generation,score,bias_list)
    if len(bias)==0:
        master_parameters,score,bias = cpy_mp,cpy_score,cpy_bias
    pygame.quit()
    print("Generation =",generation,"---------- Highest Score =",int(score))
    '''
    This section of code acts like threshold limit
    If the drone fails to evolve in a certain amount of evolutions
    Behaving like a God, I cause a purge and create a new beginning here.
    Let's name it
    Purge Code.
    '''
    if threshold_score < score:
        threshold_score = score
        threshold_count = 0
    else:
        threshold_count+=1
    if threshold_count==threshold:
        threshold_count = 0
        print("********** Purged **********")
        parameter_list = init_para()
        bias_list = init_bias()
    else:
        parameter_list1 = [master_parameters]*int(population*0.4)
        parameter_list2 = make_parameters(master_parameters,int(population*0.6))
        parameter_list = parameter_list1+parameter_list2
        bias_list1 = [bias]*int(population*0.4)
        bias_list2 = make_parameters(bias,int(population*0.6))
        bias_list = bias_list1+bias_list2

    generation+=1
```

Html, css, and js: (Frontend)

1. Index.html:

```
<!DOCTYPE html>

<html>

<head>

   <meta charset="utf-8" />

   <meta name="viewport" content="width=device-width, initial-scale=1" />

   <meta name="theme-color" content="#000000" />


   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css"
rel="stylesheet">

   <link href="style.css" rel="stylesheet">


   <!--Poppins Font-->

   <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600;700;800&
display=swap" rel="stylesheet">

   <link id='favicon' rel="shortcut icon" href="App/Content/drone.png" type="image/x-
png">

   <title>Medical Delivery using Drones</title>

  </head>

<body>


   <noscript>You need to enable JavaScript to run this app.</noscript>


   <div class="overlay">

   </div>


   <!--Navigation Bar-->
   <div class="container-fluid d-flex align-items-center justify-content-between"
id="navbar">
        <h2>Medical Delivery using Drones</h2>
        <div>
            <a id="get-started-btn" class="get-started-btn" href="App/index.html">Get
Started</a>
        </div>
   </div>


   <!--Hero Section-->
   <section id="hero" class="
```

2. Style.css:

```
body {
   font-family: poppins;
}
.overlay{
   height: 100vh;
   position: fixed;
   background-image: url(/hero-bg.png);
   background-position: center;
   background-repeat: no-repeat;
   background-size: cover;
   background-attachment: fixed;
   width: 100%;
   z-index: -1;
}
/*Section*/
section {
   padding: 60px 0;
   overflow: hidden;
  }

  .section-header {
   text-align: center;
   padding-bottom: 60px;
  }

  .section-header h2 {
   font-size: 13px;
   letter-spacing: 1px;
```

```
    font-weight: 700;
    margin: 0;
    color: #4154f1;
    text-transform: uppercase;
  }

  .section-header p {
    margin: 10px 0 0 0;
    padding: 0;
    font-size: 38px;
    line-height: 42px;
    font-weight: 600;
    color: #012970;
  }

  @media (max-width: 768px) {
    .section-header p {
      font-size: 28px;
      line-height: 32px;
    }
  }

/*Sidenav*/
.sidenav {
    height: 100%;
    width: 0;
    position: fixed;
    z-index: 1000;
    top: 0;
  right: 0;
  background-color: #111;
  overflow-x: hidden;
  transition: 0.5s;
  padding-top: 60px;
}

.sidenav a {
  padding: 8px 8px 8px 32px;
  text-decoration: none;
  font-size: 25px;
  color: #818181;
  display: block;
  transition: 0.3s;
}

.sidenav a:hover {
  color: #f1f1f1;
}

.sidenav .closebtn {
  position: absolute;
  top: 0;
  right: 25px;
  font-size: 36px;
  margin-left: 50px;
}

@media screen and (max-height: 450px) {
  .sidenav {padding-top: 15px;}
  .sidenav a {font-size: 18px;}
}

/*Navigation Bar*/
#navbar{
   padding: 15px 120px;
   box-shadow: 2px 2px 4px rgba(0, 16, 43, 0.685);
   position: fixed;
   background-color: #fff;
   z-index: 999;
}

@media (max-width: 450px){
   #navbar{
      padding: 10px 10px;
   }

}

#navbar h2{
   font-family: poppins;
   color: #012970;
}

.get-started-btn{
   text-decoration: none;
   background-color: #012970;
   color: white;
```

```
    padding: 10px 20px 10px 20px;
    margin-right: 20px;
    border-radius: 8px;
    font-family: poppins;
    font-size: 17px;
    margin-bottom: 5px;
    transition: 0.2s;
    border: none;
}

.get-started-btn:hover{
    box-shadow: 0px 21px 20px rgba(1, 41, 112, 0.1);
    color: white;
    transform: translateY(-5px);
}

/*Hero Section*/
.hero {
  width: 100%;
  height: 100vh;
}

.hero h1 {
  margin: 0;
  /*font-size: 60px;*/
  font-weight: 600;
  color: #012970;
}
```

```
.hero h2 {
  color: #444444;
  margin: 15px 0 0 0;
  font-size: 26px;
}

.hero .hero-img {
  text-align: right;
}

@media (min-width: 1024px) {
  .hero {
    background-attachment: fixed;
  }
}

@media (max-width: 991px) {
  .hero {
    height: auto;
    padding: 120px 0 60px 0;
    text-align: center;
  }
  .hero .hero-img {
    text-align: center;
    margin-top: 80px;
  }
  .hero .hero-img img {
    width: 80%;
  }
```

```
}

@media (max-width: 768px) {
  .hero {
    text-align: center;
  }
  .hero h1 {
    font-size: 32px;
  }
  .hero h2 {
    font-size: 24px;
  }
  .hero .hero-img img {
    width: 100%;
  }
}

@media (min-width: 768px){
  .hero h1{
    font-size: 50px;
  }
  .hero .hero-img{
    margin-top: 100px;
  }
}

@media (min-width: 1200px){
  .hero h1{
    font-size: 60px;
    }
  }

  /*Team Section*/

.team {
    padding: 60px 0;
  }

  .team .member {
    overflow: hidden;
    text-align: center;
    border-radius: 5px;
    background: #fff;
    box-shadow: 0px 0 30px rgba(1, 41, 112, 0.08);
    transition: 0.3s;
  }

  .team .member .member-img {
    position: relative;
    overflow: hidden;
  }

  .team .member .member-img:after {
    position: absolute;
    content: "";
    left: 0;
    bottom: 0;
    height: 100%;
    width: 100%;
    background: url(../img/team-shape.svg) no-repeat center bottom;
    background-size: contain;
    z-index: 1;
  }

  .team .member .social {
    position: absolute;
    right: -100%;
    top: 30px;
    opacity: 0;
    border-radius: 4px;
    transition: 0.5s;
    background: rgba(255, 255, 255, 0.3);
    z-index: 2;
  }

  .team .member .social a {
    transition: color 0.3s;
    color: rgba(1, 41, 112, 0.5);
    margin: 15px 12px;
    display: block;
    line-height: 0;
    text-align: center;
  }

  .team .member .social a:hover {
    color: rgba(1, 41, 112, 0.8);
  }
```

```
.team .member .social i {
  font-size: 18px;
}

.team .member .member-info {
  padding: 10px 15px 20px 15px;
}

.team .member .member-info h4 {
  font-weight: 700;
  margin-bottom: 5px;
  font-size: 20px;
  color: #012970;
}

.team .member .member-info span {
  display: block;
  font-size: 14px;
  font-weight: 400;
  color: #aaaaaa;
}

.team .member .member-info p {
  font-style: italic;
  font-size: 14px;
  padding-top: 15px;
  line-height: 26px;
  color: #5e5e5e;
```

```
}

.team .member:hover {
  transform: scale(1.08);
  box-shadow: 0px 0 30px rgba(1, 41, 112, 0.1);
}

.team .member:hover .social {
  right: 8px;
  opacity: 1;
}

/* Contact Form */
.contact .info-box {
  color: #444444;
  background: #fafbff;
  padding: 30px;
}

.contact .info-box i {
  font-size: 38px;
  line-height: 0;
  color: #4154f1;
}

.contact .info-box h3 {
  font-size: 20px;
  color: #012970;
  font-weight: 700;
  margin: 20px 0 10px 0;
}

.contact .info-box p {
  padding: 0;
  line-height: 24px;
  font-size: 14px;
  margin-bottom: 0;
}

.contact .php-email-form {
  background: #fafbff;
  padding: 30px;
  height: 100%;
}

.contact .php-email-form .error-message {
  display: none;
  color: #fff;
  background: #ed3c0d;
  text-align: left;
  padding: 15px;
  margin-bottom: 24px;
  font-weight: 600;
}

.contact .php-email-form .sent-message {
  display: none;
  color: #fff;
  background: #18d26e;
  text-align: center;
  padding: 15px;
  margin-bottom: 24px;
  font-weight: 600;
}

.contact .php-email-form .success{
  display: block;
}


.contact .php-email-form .loading {
  display: none;
  background: #fff;
  text-align: center;
  padding: 15px;
  margin-bottom: 24px;
}

.contact .php-email-form .loading:before {
  content: "";
  display: inline-block;
  border-radius: 50%;
  width: 24px;
  height: 24px;
  margin: 0 10px -6px 0;
  border: 3px solid #18d26e;
  border-top-color: #eee;
  -webkit-animation: animate-loading 1s linear infinite;
  animation: animate-loading 1s linear infinite;
}

.contact .php-email-form input, .contact .php-email-form textarea {
  border-radius: 0;
  box-shadow: none;
  font-size: 14px;
  border-radius: 0;
}

.contact .php-email-form input:focus, .contact .php-email-form textarea:focus {
  border-color: #4154f1;
}
.contact .php-email-form input {
  padding: 10px 15px;
}

.contact .php-email-form textarea {
  padding: 12px 15px;
}

.contact .php-email-form button[type="submit"] {
  background: #4154f1;
  border: 0;
  padding: 10px 30px;
  color: #fff;
  transition: 0.4s;
  border-radius: 4px;
```

```
}

.contact .php-email-form button[type="submit"]:hover {
  background: #5969f3;
}

@-webkit-keyframes animate-loading {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}
@keyframes animate-loading {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}

.footer {
  padding: 15px 0;
  background: #012970;
  min-height: 40px;
  margin-top: 82px;
  color: #fff;
  }

  @media (max-width: 992px) {
    .footer {
      margin-top: 85px;
    }
  }
/*styling.css*/
/*add font family, background image*/

body {
  color: #3602ad;
  font-family: "Poppins", sans-serif;
}

.fa-map-marker-alt,
.fa-dot-circle {
  color: #00d323;
}

/*Jumbotron*/
.jumbotron {
  background-color: transparent;
  margin: 0;
  padding: 10px;

.jumbotron h1 {
  letter-spacing: 2.5px;
  font-size: 3.5em;
}
.jumbotron h1,
.jumbotron p {
  text-align: center;
}

/*map*/
#googleMap {
  width: 80%;
  height: 400px;
  margin: 10px auto;
}
/*output box*/
#output {
  text-align: center;
  font-size: 2em;
  margin: 20px auto;
}

#mode {
  color: black;
}

form{
  margin-left: 25%;
}
button{
  margin-left: 35%;
}
```

3. Main.js

```
//javascript.js
//set map options
var myLatLng = { lat: 19.69522, lng: 73.5626 };
var mapOptions = {
   center: myLatLng,
   zoom: 7,
   mapTypeId: google.maps.MapTypeId.HYBRID

};

//create map
var map = new google.maps.Map(document.getElementById('googleMap'), mapOptions);

//create a DirectionsService object to use the route method and get a result for our request
var directionsService = new google.maps.DirectionsService();

//create a DirectionsRenderer object which we will use to display the route
var directionsDisplay = new google.maps.DirectionsRenderer();

//bind the DirectionsRenderer to the map
directionsDisplay.setMap(map);



//define calcRoute function
function calcRoute() {
   //create request
   var request = {
        origin: document.getElementById("from").value,

        destination: document.getElementById("to").value,

        travelMode: google.maps.TravelMode.DRIVING, //WALKING, BYCYCLING,
TRANSIT

        unitSystem: google.maps.UnitSystem.IMPERIAL

    }


    //pass the request to the route method

    directionsService.route(request, function (result, status) {

        if (status == google.maps.DirectionsStatus.OK) {


            //Get distance and time

            const output = document.querySelector('#output');

            output.innerHTML = "<div class='alert-info'>From: " +
document.getElementById("from").value + ".<br />To: " +
document.getElementById("to").value + ".<br /> Distance <i class='fas fa-road'></i> : " +
result.routes[0].legs[0].distance.text + ".<br />Duration <i class='fas fa-hourglass-start'></i> :
" + result.routes[0].legs[0].duration.text + ".</div>";


            //display route

            directionsDisplay.setDirections(result);

        } else {

            //delete route from map

            directionsDisplay.setDirections({ routes: [] });

            //center map in London

            map.setCenter(myLatLng);


            //show error message

            output.innerHTML = "<div class='alert-danger'><i class='fas fa-exclamation-
triangle'></i> Could not retrieve distance.</div>";

        }
    });

}



//create autocomplete objects for all inputs
var options = {
    types: ['(cities)']
}

var input1 = document.getElementById("from");
var autocomplete1 = new google.maps.places.Autocomplete(input1, options);

var input2 = document.getElementById("to");
var autocomplete2 = new google.maps.places.Autocomplete(input2, options);
hero d-flex align-items-center ">
        <div class="container">
            <div class="row">
                <div class="col-lg-6 d-flex flex-column justify-content-center">
                    <h1>Medicinal Delivery using Drones</h1>
                    <h2>Easily deliver the medicines using Drones </h2>
                </div>


                <div class="col-lg-6 hero-img">
                    <img src="assets/drone-new.png" class="img-fluid" alt="">
                </div>
            </div>
```

```
      </div>
    </section>
    <!--Hero Section End-->

    <!--Team Section-->
    <section id="team" class="team">

      <div class="container" data-aos="fade-up">

        <header class="section-header">
          <p>Our Team</p>
        </header>

        <div class="row gy-4 justify-content-around">

          <div class="col-lg-3 col-md-6 d-flex align-items-stretch" data-aos="fade-up" data-aos-
delay="100">
            <div class="member">
              <div class="member-img">
                <img src="/assets/Gaurav-Patil1.png" class="img-fluid" width="300" height="400"
alt="">
              </div>
              <div class="member-info">
                <h4>Gaurav Patil</h4>
                <span>AI Student</span>
              </div>
            </div>
          </div>
        <div class="col-lg-3 col-md-6 d-flex align-items-stretch" data-aos="fade-up" data-aos-
delay="100">
          <div class="member">
            <div class="member-img">
              <img src="/assets/Hrugved.png" class="img-fluid" width="300" height="400"
alt="">
              <div class="social">
                <!-- <a href="https://www.linkedin.com/in/hrugved-kolhe-364881193/"><i
class="bi bi-linkedin"></i></a> -->
              </div>
            </div>
            <div class="member-info">
              <h4>Hrugved Kolhe</h4>
              <span>AI Student</span>
            </div>
          </div>
        </div>



          <div class="col-lg-3 col-md-6 d-flex align-items-stretch" data-aos="fade-up" data-
aos-delay="100">
          <div class="member">
            <div class="member-img">
              <img src="/assets/Rohit1.png" class="img-fluid" width="300" height="400"
alt="">
              <div class="social">
                <!--<a href="#"><i class="bi bi-linkedin"></i></a>-->
              </div>
            </div>
            <div class="member-info">
              <h4>Rohit Chitte</h4>
              <span>AI Student</span>
            </div>
          </div>
        </div>



          <div class="col-lg-3 col-md-6 d-flex align-items-stretch" data-aos="fade-up" data-aos-
delay="100">
            <div class="member">
              <div class="member-img">
                <img src="/assets/AtharvaD1.png" class="img-fluid" width="300" height="400"
alt="">
                <div class="social">
                  <!--<a href="#"><i class="bi bi-linkedin"></i></a>-->
                </div>
              </div>
              <div class="member-info">
                <h4>Atharva Deolalikar</h4>
                <span>AI Student</span>
              </div>
            </div>
          </div>


        </div>


      </div>





    </section>
    <!-- End Team Section -->

    <!--Contact Form-->
    <div id="contact" class="contact mt-5">

      <div class="container" data-aos="fade-up">

        <header class="section-header">
          <p>Contact Us</p>
        </header>

        <div class="row gy-4">

          <div class="col-lg-6 hero-img">
                <img src="/assets/contact.png" class="img-fluid" alt="">
          </div>

          <div class="col-lg-6">
           <form id="php-email-form" class="php-email-form" accept-charset="UTF-8">
             <div class="row gy-4">

               <div class="col-md-6">
                 <input type="text" name="name" class="form-control" placeholder="Your Name"
required>
               </div>

               <div class="col-md-6 ">
                 <input type="email" class="form-control" name="email" placeholder="Your
Email" required>
               </div>
               <div class="col-md-12">
                 <input type="text" class="form-control" name="subject" placeholder="Subject"
required>
               </div>

               <div class="col-md-12">
                 <textarea class="form-control" name="message" rows="6"
placeholder="Message" required></textarea>
               </div>
               <input type="hidden" name="_redirect"value="http://localhost:5501/contact/"/>
               <div class="col-md-12 text-center">
                 <div class="loading">Loading</div>
                 <div class="error-message"></div>
                 <div id="sent" class="sent-message">Your message has been sent. Thank
you!</div>

                 <button class="contact-us-btn" type="submit">Send Message</button>
               </div>

             </div>
            </form>

          </div>

        </div>

      </div>

    </div>
```

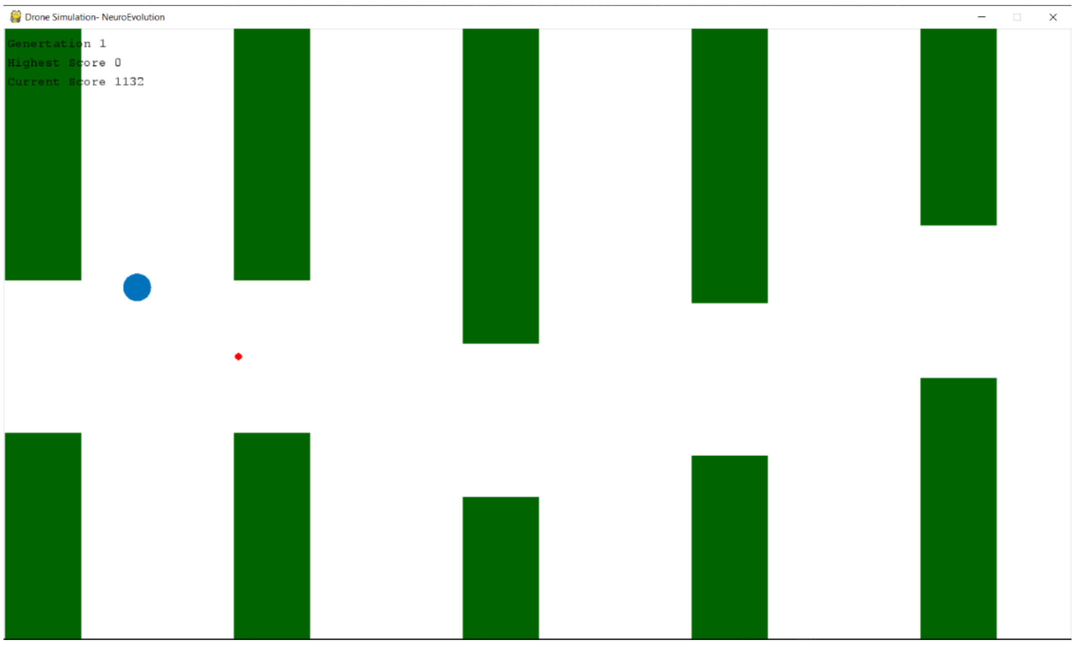
```
    <!--Contact Form - End-->

    <div class="footer">
        <div class="container d-flex align-items-center justify-content-center">
            © Copyright. All Rights Reserved
        </div>
    </div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/js/bootstrap.min.js"></script>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script src="{% static 'assets/js/sidebar.js' %} "></script>

</body>
</html>
```

Simulation display:



```
Genertation 1
Highest Score 0
Current Score 1132
```

Explanation of the Simulation Display:

The simulation display represents a drone path optimization model based on neuro-evolution. This graphical interface is a core component of the Medical Drone Delivery Simulation, where drones are trained to navigate through obstacles autonomously.

Key Elements in the Display:

1. Drone Representation (Blue Circle)

The blue circle represents the drone, which is navigating through a series of vertical obstacles (green bars).

The drone must avoid collisions while maintaining a steady flight path.

2. Obstacles (Green Vertical Bars)

These green bars simulate buildings, trees, or restricted airspaces that the drone must maneuver around.

The spaces between them represent possible flight paths that the drone must learn to follow.

3. Target Point (Small Red Dot)

This red dot may indicate a reference point for navigation or a checkpoint.

The drone must pass through these open spaces while avoiding obstacles.

4. Performance Metrics (Top Left Corner)

Generation: Indicates the number of iterations in the genetic algorithm. Each generation represents a new batch of drones learning to improve their navigation.

Highest Score: Shows the best performance recorded across multiple runs.

Current Score: Represents the real-time distance traveled or fitness of the current drone.

How the Simulation Works:

The Genetic Algorithm (GA) is used to train the drone by adjusting parameters such as speed, movement decisions, and path selection.

Initially, the drones make random movements, but over time, they evolve by learning optimal flight paths based on a fitness function (distance traveled and precision in avoiding obstacles).

The best-performing drones are used as a template for the next generation, with small mutations introduced to improve their decision-making capabilities.

The simulation continues iterating until the drone can autonomously navigate without crashing into obstacles.

Purpose of the Simulation:

This simulation is crucial for developing an optimized medical drone delivery system, as it ensures:

1. Efficient obstacle avoidance in real-world environments.
2. Adaptive learning for different weather and terrain conditions.
3. Safe and reliable delivery routes for medical supplies in emergency situations.

This visualization helps researchers and developers fine-tune drone behavior before real-world deployment, making medical drone deliveries more accurate, faster, and safer.

CONCLUSION

The implementation of medical drone delivery systems represents a significant leap forward in healthcare logistics. By leveraging advanced AI-driven navigation systems, this project aims to enhance the accessibility, efficiency, and reliability of medical supply distribution. The ability to bypass traditional transportation constraints enables faster emergency responses, which is particularly beneficial in remote and disaster-stricken regions.

The developed simulation model will not only optimize flight paths but also provide valuable insights into the feasibility of real-world deployment. With continuous improvements in battery life, machine learning algorithms, and regulatory support, drones have the potential to become a fundamental component of modern healthcare infrastructure.

The project's findings will contribute to ongoing research and development efforts, assisting policymakers and logistics experts in designing effective drone-based delivery frameworks. By integrating drones into the medical supply chain, we can significantly reduce delivery times, minimize costs, and improve patient outcomes. Ultimately, this research paves the way for a transformative shift in medical logistics, where technology bridges the gap between healthcare services and those in need.

REFERENCES

1. Hrugved, Medical Drone Delivery - GitHub Repository
2. World Health Organization. (2021). "Emergency Medical Supply Chain Logistics."
3. FAA Drone Regulations - Federal Aviation Administration (FAA), https://www.faa.gov
4. Sharma, R., & Kumar, P. (2020). "AI-Powered Drone Navigation for Healthcare Logistics." International Journal of Advanced Research in AI, 35(4), 102-118.
5. UNICEF Drone Delivery Network, https://www.unicef.org/innovation/drones