Cairo University

Faculty of Computers Science and Artificial Intelligence

Artificial Intelligence Department

Phase#1

Drone Delivery Agent

With the increasing demand for fast and reliable delivery services, businesses are looking for innovative ways to improve their logistics operations. One of the most promising solutions is the use of unmanned aerial vehicles (UAVs), also known as drones, to deliver packages and other items.

in this document, we will explore the various aspects of drone delivery agents with 5 destinations and one initial point(warehouse), including their design, functionality, and benefits. We will also examine the challenges and limitations of this technology and discuss potential solutions to overcome them.

1- Definition of Drone Delivery and Planning:

A drone delivery agent with 5 destinations is a specific type of UAV system that can be programmed to deliver packages to up to five different locations within a relatively small geographic area. This autonomous agent is equipped with sensors, cameras, and other advanced technologies that allows it to navigate through the airspace, avoid obstacles.

The drone starts in a single initial state(warehouse), which is typically a central location where packages are loaded onto the drone for delivery. From there, the drone travels to each of the five designated destinations, delivering packages along the way. The destinations can be pre-programmed into the drone's software or communicated to the drone in real-time by the delivery system.

2- Environment Type:

The environment type for a ***Drone Drop*** depends on the application and use case of the drone. However, in general, delivery drones are designed to operate in urban and suburban environments, where they can quickly and efficiently deliver packages and goods to customers. These environments are typically characterized by:

- **Obstacles:** Drones Drop need to navigate through buildings, trees, power lines, and other obstacles in urban and suburban areas. They must be able to detect and avoid obstacles using sensors such as cameras, lidar, and radar.

- **Weather conditions:** Drones Drop must be able to operate in a variety of weather conditions, including rain, wind, and snow. They must be equipped with weather-resistant materials and components to ensure reliable operation.

- **Traffic:** Drones Drop need to navigate through busy urban and suburban areas with heavy traffic. They must be able to avoid collisions with other vehicles and pedestrians and obey traffic laws and regulations.

- **Security:** Drones Drop need to be secure to prevent theft or damage to packages. They may be equipped with anti-theft features such as GPS tracking, remote disablement, and tamper-proof packaging.

- **Communication:** Drones Drop require a robust communication system to communicate with ground-based controllers and other drones in the airspace. They may use technologies such as Wi-Fi, cellular networks, or satellite communication to ensure reliable communication.

Overall, Drones Drop must be designed to operate safely and efficiently in complex and dynamic environments, where they must navigate through obstacles, weather, traffic, and security concerns to deliver packages and goods to customers.

3- Agent Type:

The agent type for a Drone Drop is an autonomous agent, as it is designed to operate without human intervention. Delivery drones are equipped with sensors, navigation systems, and control algorithms that enable them to navigate through environments, detect obstacles, and make decisions in real-time.

The agent type for Drones Drop is a combination of multi-agents, including:

1. **Perception agent:** This agent is responsible for sensing the environment using sensors such as cameras, lidar, and radar, and creating a representation of the environment.

2. **Navigation agent:** This agent is responsible for planning the drone's trajectory and controlling its movement through the environment, while avoiding obstacles and staying on track.

3. **Delivery agent:** This agent is responsible for identifying the delivery location and delivering the package to the designated location with precision.

Overall, the autonomous agent type for Drones Drop is critical for enabling them to operate autonomously, efficiently, and safely, while delivering packages to customers.

4- PEAS:

PEAS stands for Performance measure, Environment, Actuators, and Sensors. It is a framework used to define an intelligent agent's task and environment. Here is the PEAS analysis for a Drone Drop:

- **Performance measure:** The performance measure for a Drone Drop is the successful and timely delivery of packages to the designated locations while avoiding collisions with obstacles, obeying traffic laws and regulations, and minimizing delivery time and cost.

- **Environment:** The environment for a Drone Drop includes urban and suburban areas, where it must navigate through obstacles, weather conditions, traffic, and security concerns to deliver packages to customers.

- **Actuators:** The actuators for a Drone Drop include propellers, motors, and control surfaces that enable it to move through the environment and deliver packages to the designated locations. The drone's actuators are controlled by its onboard computer and navigation system.

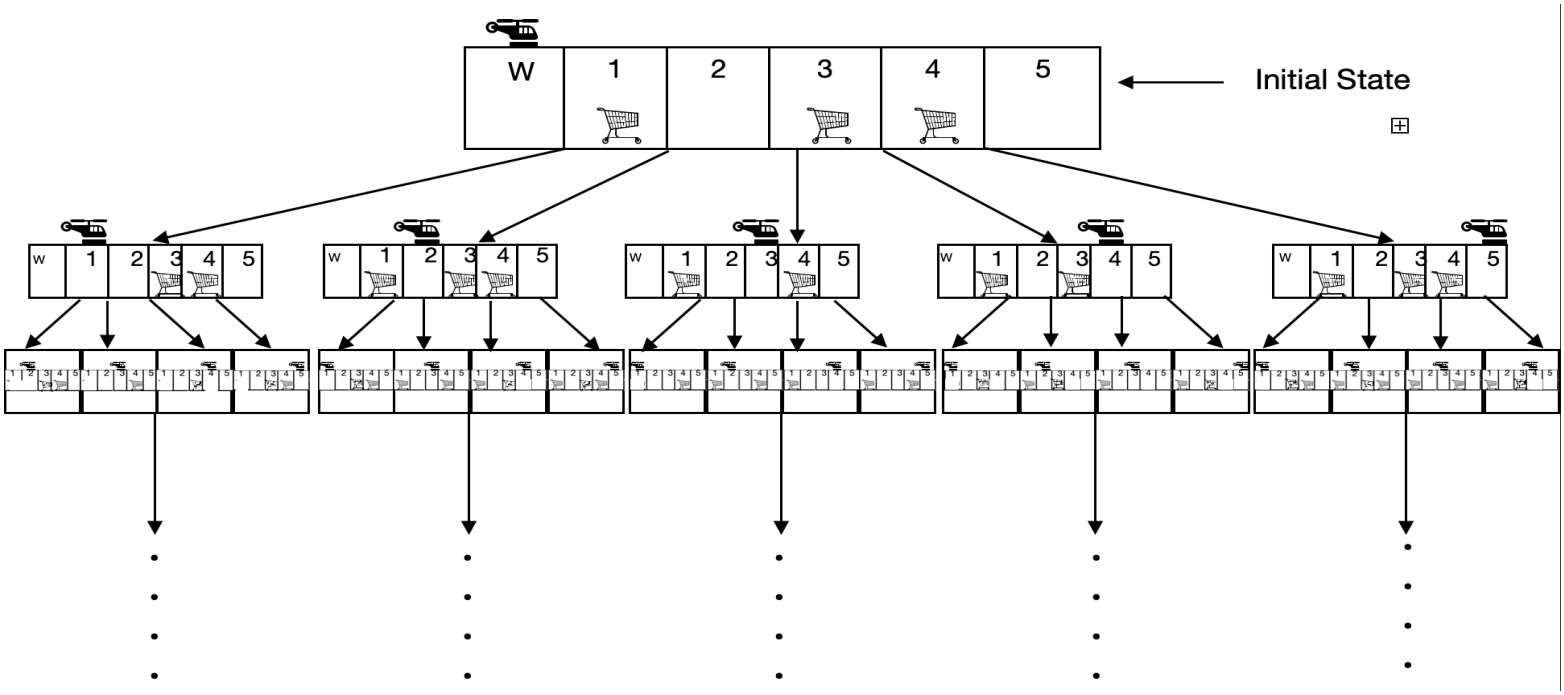
- **Sensors:** The sensors for a Drone Drop include cameras, radar, GPS, and other sensors that enable it to sense the environment, detect obstacles, and navigate through the airspace. The drone's sensors are used to gather information about the environment and to make decisions based on this information.

5- Problem State Space and Initial State:

Since the drone agent is responsible for the distribution of products from a warehouse to 5 selling points then the state space would represent the states of each selling point (placed an order / no order) and the current location of the drone across all 5 locations plus the warehouse.

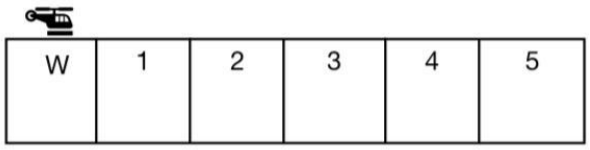
State space = n x 2^n-1 = 6 x 2^5 = 192 possible states.

Notice: we used n-1 here as the warehouse doesn't have ordering states like selling points so it was excluded, but we multiplied by n as the drone could exist there at any point.



With a goal state that looks like this:

With a goal state that looks like this:



Where the drone is back at the warehouse and no orders at selling points.

So given an initial state and the final goal state our objective is to find a path through the previous state tree that fulfills the goal state with minimum cost.

6- Goal test to verify reaching goal state:

Assuming that the goal state is reached when all products have been successfully delivered to their respective branch supermarkets, the following goal test can be performed:

1. Check if all products have been loaded onto the drone at the Store.
2. Check if the drone has successfully navigated to each of the 5 branch supermarkets.
3. Check if the drone has successfully landed at each branch supermarket and delivered the correct products.
4. Verify that the products have been successfully received by the staff at each branch supermarket.
5. Check if there have been any errors, such as delivering the wrong products or missing a branch supermarket.
6. Verify that the drone has returned safely to the Store after completing all deliveries.

7- Successor function with (Set of actions & Paths costs):

- **Move to a neighboring location:**

The drone can move to any neighboring selling point in its current area. This function would consider the drone's current location, the location of the destination, and any obstacles that could affect the drone's flight path.

- **Avoid obstacles:**

This function would identify any obstacles in the drone's flight path, such as buildings or trees, and adjust the drone's flight path and altitude to avoid them. This could be accomplished using sensors or other technologies that detect obstacles in real-time.

- **Manage battery life:**

The drone's battery life is limited, so this function would consider the distance to the destination, the amount of power remaining in the drone's battery, and the estimated power required to complete the delivery. The function would then adjust the drone's flight path to conserve energy and ensure that it arrives at the destination with enough power to complete the delivery.

- **Optimize delivery routes:**

This function would analyze all of the available delivery routes and select the optimal path based on distance, obstacles, and delivery time.

8- Transition Model:

- **Move to a neighboring location:**

- **Inputs:** Drone's current state with its location the destination of the package it is delivering, and its battery level.
- **Outputs:** Available locations for the drone.
- **Steps:**

1- Determine the drone's current location and the location of the package it is delivering.

2- Calculate the shortest path the drone could take to move to a different location or deliver a package.

3- Check the battery level of the drone and if it needs to be recharged.

4- Return the available actions for the drone.

- **Avoid obstacles:**
  - The drone detects a building or any other obstacle in its flight path.
  - The transition model for this function would adjust the drone's flight path to avoid the obstacle. Whether it is by changing the altitude that the drone flies at or calculating a new flight path that goes around the building.

- **Manage battery life:**
  - calculate the amount of power required to move the drone from its current location to the destination.
  - Subtract that amount calculated from the remaining battery life.
  - If the subtraction yields less than 20% the drone has to charge first.

- **Optimize delivery routes:**
    - Choose an optimal route from current location to destination based on cost and a pre-determined heuristic function.
    - Update the optimal route at every node and according to environment changes until it reaches a destination.

9- A solution / plan to show the transformation from initial state to goal state:

1-Drone location: The initial location of the drone, which may be the warehouse or a nearby launch site.

2-Package location: The location of the package or goods that are to be delivered, which may be in the warehouse or at another location. (goal state).

3-Plan the route: Plan the route that the drone will take to move from the initial state to the goal state based on state graph .

4-Optimize the route: Optimize the route to minimize the time or distance required to deliver the package.

5-Delivery schedule: The expected delivery time and date, which may be predetermined or scheduled based on customer preferences.

6-Weather conditions: The current weather conditions, which can affect the safety and feasibility of the delivery.

7-Monitor the drone: Monitor the drone during the delivery process to ensure that it is on the planned route and to detect and respond to any unexpected obstacles or events.

8-Deliver the package: Once the drone reaches the delivery location, it can deliver the package and return to its starting location or move on to the next delivery location.

9-Evaluate the performance: Evaluate the performance of the drone delivery system by measuring metrics like the delivery time, delivery accuracy, and customer satisfaction.

-A* implementation: -

1- Assumptions and Information needed to Apply A* Algorithm: -

Our assumed heuristic function should give a good estimate of the cost of reaching the destined goal, this estimate should provide a reasonable estimate of the cost depending on prior knowledge of the environment and its setting. In the case of our drone delivery problem the Euclidean distance, that is the straight-line distance between the two points should give us a good estimate of the cost giving the fact that it assumes that the drone can move in any direction and has no restrictions on its movement. So given a map of the neighborhood where the warehouse and five selling points are, we could measure the straight-line distance between each location and the goal where the goal is dynamic in this case.

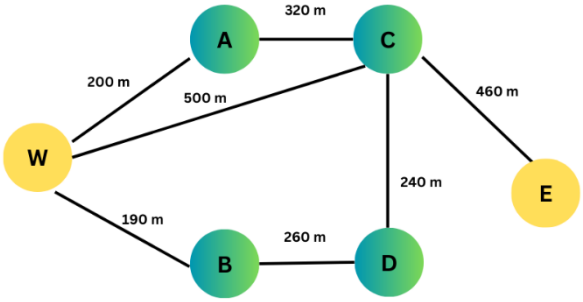
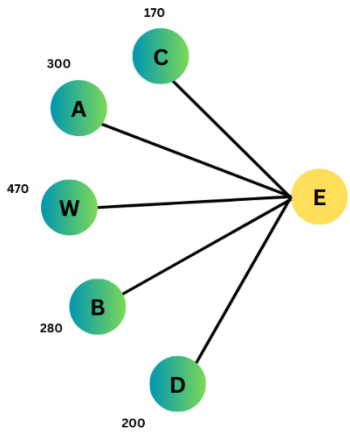


Figure of Actual Distance On map

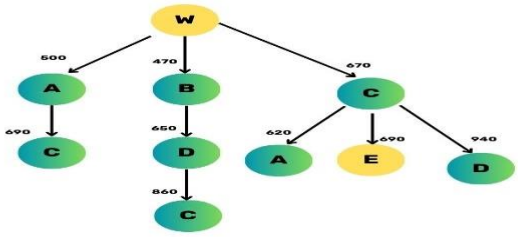


Euclidean distance (heuristic function

So assuming our goal destination at some point is reaching selling point E, the figure to the left demonstrates the distance between the warehouse and selling points on the map, while the figure to the right demonstrates the Euclidean distance (straight line distance) between each location and the goal location with is selling point E, and the total cost will be calculated as the combination of the actual distance and the Euclidean distance : F(n) = E(n) + H(n) Where of course the Euclidean distance is calculated for each goal state.

2- Apply A* search algorithm on the state Tree: -

-when Drone moves from the warehouse (initial point) to the goal (E):



1- from **W** → **A** the cost will be:

200 + 300 = 500

F(n) + g(n)

2- from **W** → **B** the cost will be:

190 + 280 = 470

3- from **W** → **C** the cost will be:

500 + 170 = 670

➔ As we see the minimum cost is at the point **B** so we will continue at this
path (470 < 500 and 670).

4- from **W** → **B** → **D** the cost will be:

180 + 260 + 200 = 650

➔ As we see the minimum cost is at the point **A** so we will continue at this path (470 < 500 and 670).

5- from **W** → **A** → **C** the cost will be:

200 + 320 + 170 = 690.

➔ As we see the minimum cost is at the point **D** so we will continue at this path (650 < 690 and 670).

6- from **W**→ **B** → **D** → **C** the cost will be:

190 + 260 + 240 + 170 = 860.

➔ As we see the minimum cost is at the point **C** so we will continue at this path (670 < 860 and 690).

7- from **W**→ **C** → **E** the cost will be:

500 + 460 = 960 (this is our goal).

-A* Implementation:

In [1]:

```
import tkinter as tk
from tkinter import ttk
import heapq

# Define the graph as a dictionary of nodes and their neighbors with edge costs
graph = {
    'W': {'A': 200, 'B': 190, 'C': 500},
    'A': {'C': 320},
    'B': {'D': 260},
    'D': {'C': 240},
    'C': {'E': 460},
    'E':{}
}

coordinates = {
    'W': (50, 50),
    'A': (150, 50),
    'B': (50, 150),
    'C': (250, 150),
    'D': (150, 250),
    'E': (250, 250)
}


# Define a function to calculate the heuristic value for a given node
def heuristic(node, goal):
    # In this example, we use a simple heuristic that estimates the distance between two nodes as the number of nodes
    # between them
    x1, y1 = coordinates[node]
    x2, y2 = coordinates[goal]
    return int(((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5)


# Define the A* algorithm function
def astar(start, goal):
    # Create empty dictionaries to store the cost from the start node and the parent of each node
    g = {start: 0}
    parent = {start: None}
```

```
# Create an empty heap and add the start node with a priority of 0
heap = []
heapq.heappush(heap, (0, start))

# Loop until the heap is empty
while heap:
    # Pop the node with the lowest cost from the heap
    current_cost, current_node = heapq.heappop(heap)

    # If we have reached the goal node, construct the path and return it
    if current_node == goal:
        path = []
        while current_node is not None:
            path.append(current_node)
            current_node = parent[current_node]
        path.reverse()
        return path

    # Otherwise, loop through the neighbors of the current node
    for neighbor, cost in graph[current_node].items():
        # Calculate the cost from the start node to this neighbor through the current node
        new_cost = g[current_node] + cost

        # If this neighbor has not been visited yet or the new cost is lower than the existing cost, update the
        # cost and parent
        if neighbor not in g or new_cost < g[neighbor]:
            g[neighbor] = new_cost
            parent[neighbor] = current_node
            f = new_cost + heuristic(neighbor, goal)
            heapq.heappush(heap, (f, neighbor))

# If we have exhausted all possible paths and haven't found the goal, return None
return None
```

```
def run_astar():
    start = start_var.get()
    goal = goal_var.get()
    path = astar(start, goal)
    if path is not None:
        path_text.set(' -> '.join(path))
        cost_text.set(str(calculate_path_cost(path)))
        print('Optimal path:', ' -> '.join(path))
        print('Total cost:', calculate_path_cost(path))
        redraw_graph(path)
    else:
        path_text.set('No path found')
        cost_text.set('')
        print('No path found')

def calculate_path_cost(path):
    cost = 0
    for i in range(len(path) - 1):
        cost += graph[path[i]][path[i+1]]
    return cost

def redraw_graph(path=None):
    # Clear the canvas
    canvas.delete('all')

    # Draw the nodes and edges on the canvas
    for node, coords in coordinates.items():
        x, y = coords
        canvas.create_oval(x - 10, y - 10, x + 10, y + 10, fill='#d9e6f2')
        canvas.create_text(x, y, text=node, fill='#333333')

    for node, edges in graph.items():
        x1, y1 = coordinates[node]
        for neighbor, cost in edges.items():
            x2, y2 = coordinates[neighbor]
            if path and ((node, neighbor) in path or (neighbor, node) in path):
                canvas.create_line_with_arrow(x1, y1, x2, y2, arrow='last', arrowshape=(8, 10, 3), fill='#e91e63', width=3)
            else:
                canvas.create_line(x1, y1, x2, y2, fill='#333333', width=2)
```

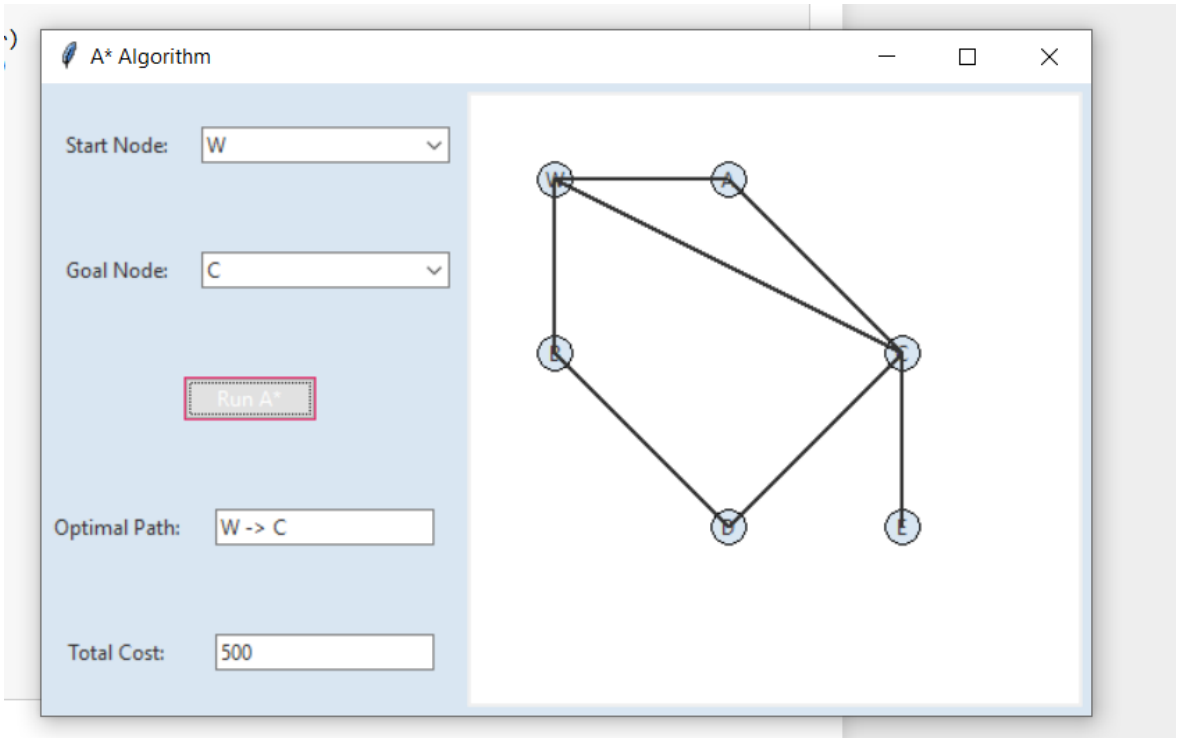
```
# Define a custom function to create a line with an arrow on the canvas
def create_line_with_arrow(self, x1, y1, x2, y2, **kwargs):
    self.create_line(x1, y1, x2, y2, **kwargs)
    dx = x2 - x1
    dy = y2 - y1
    angle = tk.atan2(dy, dx)
    arrow_len = 12
    arrow_x = x2 - arrow_len * tk.cos(angle)
    arrow_y = y2 - arrow_len * tk.sin(angle)
    self.create_line(x2, y2, arrow_x, arrow_y, **kwargs)


# Add the custom function to the Canvas class
tk.Canvas.create_line_with_arrow = create_line_with_arrow


# Create the root window and title it
root = tk.Tk()
root.title('A* Algorithm')

# Define the colors for the GUI
root.configure(bg='#d9e6f2')
ttk.Style().configure('TLabel', background='#d9e6f2', foreground='#333333')
ttk.Style().configure('TCombobox', background='#ffffff', foreground='#333333')
ttk.Style().configure('TButton', background='#e91e63', foreground='#ffffff')
ttk.Style().configure('TEntry', background='#ffffff', foreground='#333333')

# Create the GUI widgets and place them in the window
start_label = ttk.Label(root, text='Start Node:')
start_label.grid(column=0, row=0, padx=5, pady=5)
start_var = tk.StringVar()
start_dropdown = ttk.Combobox(root, textvariable=start_var, values=list(graph.keys()))
start_dropdown.grid(column=1, row=0, padx=5, pady=5)
start_dropdown.current(0)

goal_label = ttk.Label(root, text='Goal Node:')
goal_label.grid(column=0, row=1, padx=5, pady=5)
goal_var = tk.StringVar()
goal_dropdown = ttk.Combobox(root, textvariable=goal_var, values=list(graph.keys()))
goal_dropdown.grid(column=1, row=1, padx=5, pady=5)
goal_dropdown.current(4)
```

```
run_button = ttk.Button(root, text='Run A*', command=run_astar)
run_button.grid(column=0, row=2, columnspan=2, padx=5, pady=5)

path_label = ttk.Label(root, text='Optimal Path:')
path_label.grid(column=0, row=3, padx=5, pady=5)
path_text = tk.StringVar()
path_text.set('')
path_entry = ttk.Entry(root, textvariable=path_text)
path_entry.grid(column=1, row=3, padx=5, pady=5)

cost_label = ttk.Label(root, text='Total Cost:')
cost_label.grid(column=0, row=4, padx=5, pady=5)
cost_text = tk.StringVar()
cost_text.set('')
cost_entry = ttk.Entry(root, textvariable=cost_text)
cost_entry.grid(column=1, row=4, padx=5, pady=5)

canvas = tk.Canvas(root, width=350, height=350, bg='#ffffff')
canvas.grid(column=2, row=0, rowspan=5, padx=5, pady=5)

# Draw the initial graph on the canvas
redraw_graph()

# Start the main event loop
root.mainloop()
```

```
Optimal path: W -> C
Total cost: 500
```

- From the previous code, A* was useful for cases with one goal but it wasn't useful for more than one. Because we have to return to initial point after getting the goal, so it wasn't very useful if we needed to make drone deliver to more than one home.

-GUI:



-This is a Video for Amazon Dron Delivery Agent:

Team Members:

No.	Name	ID
1-	Rana Abdulsalam Mohammed	20200752
2-	Kirollos Meshmesh Abo-Elyamin	20200400
3-	Marwa Shaaban Eid	20200516
4-	Mohammed Ahmed Saleh	20200693
5-	Sahar Hamdi Abdulhafeez	20201089