BOT BRAINS BATTLE – REAL WORLD RUMBLE

Solution 2D

To make sure our drones can operate independently and effectively search for targets, each one will have specific features and follow a structured plan:

I. Flight Path and Collision Avoidance:

We program each drone with a set search pattern that covers different parts of the target area. The LiDAR sensors are always scanning the surroundings to spot any obstacles and avoid crashes by adjusting the flight path in real-time.

II. Object Detection and Measurement:

When a drone comes across an object, it uses its LiDAR sensor to measure its size. If the object is 15 cm tall, the drone goes on to measure its width and depth to make sure it matches the required dimensions.

III. Color Detection:

If the dimensions are correct, the drone activates its color sensor to figure out the object's color. If it's green, the drone knows it's found the target.

IV. Swarm Communication:

As soon as a target is identified, the drone immediately communicates the location and confirms the target to the other drones using our established swarm communication protocol. The other drones then stop their search operations.

V. Information Transmission:

The drone that finds the target sends all the details, including precise coordinates, to the central command or the controlling unit.

This method guarantees an efficient and coordinated search operation by using the strengths of autonomous drones and swarm intelligence to achieve the desired outcome.

Algorithm to Implement the Autonomous Drone Search and Communication System:

I. **Initialization:**

- Initialize each drone with its unique ID and predefined search area coordinates.
- Establish communication links between the drones using swarm communication protocols.

II. **Takeoff and Navigation:**

- Each drone autonomously takes off and navigates to its starting position within the specified search area.

III. **Search Operation:**

- Each drone starts scanning its designated area using LiDAR sensors to detect objects.
- When an object is detected, the drone measures its dimensions.

IV. **Object Verification:**

- If the object's height is 15 cm, the drone measures its width and depth.
- If all dimensions are 15 cm, the drone activates its color sensor to check the object's color.

V. **Color Detection:**

- If the object is green, it is identified as the target.

VI. **Communication:**

- The drone that finds the target sends the target location information to the other drones.
- Upon receiving the information, the other drones stop their search operations.

VII. **Return to Base:**

- All drones return to their base positions and land autonomously.

Calculations for Drone Navigation and Search

Detailed Calculations For Drone Navigation and Search:

To develop a comprehensive autonomous drone system, we will focus on the following key areas:

i) Search Area Division
ii) Flight Path Planning
iii) Collision avoidance
iv) Communication Range.

1) Search Area Division:

Assuming the total search area is a square field with side length S meters.

- Total Search Area (A):

$$A = S^2 \text{ square meters.}$$

- Number of Drones (N):

$$N = 3$$

- Area per Drone (Ad):

$$Ad = \frac{A}{N} = \frac{S^2}{3} \text{ square meters.}$$

If the drones are assigned equal areas, each drone will cover an area of $\frac{S}{\sqrt{3}} \times \frac{S}{\sqrt{3}}$ meters.

2. Flight Path Planning:

Assuming each drone follows a lawnmower pattern for efficient area coverage.

- Search Grid Size (G'):

$$G = \sqrt{Ad} = \sqrt{\frac{S^2}{3}} = \frac{S}{\sqrt{3}} \text{ meters.}$$

- Flight Path Length (L):
  - If the search area is divided into strips of with W meters.

$$L = \frac{G}{W} \times 2 \times G = \frac{S}{\sqrt{3} \times W} \times 2 \times \frac{S}{\sqrt{3}}$$

After simplifying:

$$L = \frac{2S^2}{3W} \text{ meters.}$$

3) Collision Avoidance:

To ensure drones do not collide with each other, maintain a minimum separation distance Ds.

- Minimum Separation Distance (Ds):

Ds = Separation Distance (typically 5 - 10 meters)

Ensure each drone's path does not overlap and is atleast Ds meters apart.

4) Communication Range:

Ensure the communication module (ESP8266) covers the necessary range.

- Communication Range (Rc):

Rc = Typical range of ESP8266, approx 100 meters.

5) Power Consumption and Flight Time :

Estimate the power required to cover the search area.

a) Battery Capacity (C) :

- Given in mAh (milliampere / hour).

b) Drone Power Consumption (P) :

- Given in watts (W).

c) Flight Time (T) :

$$T = \frac{C \times V}{P} \text{ hours}$$

where : V is the voltage of the battery
(Typically 11.5 V is applied for a 3S LiPo battery).

Example Calculations :

Assume the following,

S = 300 meters (total search area side length)

W = 10 meters (width of each strip in lawnmower pattern)

D_S = 10 meters (minimum separation distance)

C = 5000 mAh (battery capacity)

P = 50 W (power capacity).

1. Search Area Division :

$$A = S^2 = 300^2 = 90,000 \text{ square meters}$$

$$Ad = \frac{90000}{3} = 30,000 \text{ square meters.}$$

$$G = \sqrt{30,000} \approx 173.2 \text{ meters.}$$

2. Flight Path Length :

$$L = \frac{2 \times 300^2}{3 \times 10} = \frac{1,80,000}{30} = 6000 \text{ meters}$$

3. Communication Range :

$$R_C = 100 \text{ meters}$$

4. Battery Flight Time :

$$T = \frac{5000 \times 11.1}{50,000} = 1.11 \text{ hours} \approx 66 \text{ minutes.}$$

ALGORITHMS AND PSEUDOCODES

BOT BRAINS BATTLE

Round - 3

Real - World Rumble

Algorithms for Drone Navigation and Search :

Step 1 → Initialization :

Each drone is initialized with its unique ID, search area co-ordinates, and communication parameters.

Algorithm for Initialization :

1) Initialize drone with IDs
2) Assign search area coordinates to each drone.
3) Establish communication links between drones.

Pseudo Code for Initialization :

```
void initializeDrones (vector<Drone> &drones) {
    for (int i=0; i<drones.size(); ++i) {
        drones[i].setID(i+1);
        drones[i].setSearchArea(/* coords here */);
        drones[i].takeOff();
    }
}
```

Step 2 → Search Operation :

Each drone scans its designated area using a lawnmower pattern and checks objects using LiDAR and color sensors.

Algorithm for Search Operation :

1) Drone starts at the designated starting position.
2) Scan Area using LiDAR.
3) If an object is detected, measure its dimensions.
4) If dimensions match the target, use the color sensor to detect color.
5) If the color matches the target, communicate with other drones to inform them.
6) Continue search until the target is found or the area is completely scanned.

Pseudocode for Search Operation :

```
void searchArea (Drone &drone, vector<Drone> &drones) {
    while (true) {
        auto object = drone.scanObject()
        if (verifyDimensions (object.height, object.width, object.depth
                                                                    )) {
            string color = drone.detectColor (object);
            if (verifyColor (color)) {
                communicateTargetLocation (drone, drones);
                return ;
            }
        }
    }
    drone.continueSearch () ;
  }
}
```

3) Object Verification :

Verify if the detected object matches the target Dimensions and Colors.

Algorithm for Object Verification :

1. Measure the height, width and depth of object.
2. Check if the dimensions match the target dimensions.
3. Use the color sensor to detect the color.
4. Check if the color matches the target color.

Pseudocode for Object verification :

```
bool verifyDimensions (double height, double width, double
                                                depth) {
    return (height == TARGET_HEIGHT &&
            width == TARGET_WIDTH &&
            depth == TARGET_DEPTH);
}

bool verifyColor (const string & color) {
    return (color == TARGET_COLOR);
}
```

Algorithm for Communication :

1. The drone that finds the target sends the target location to other drones.

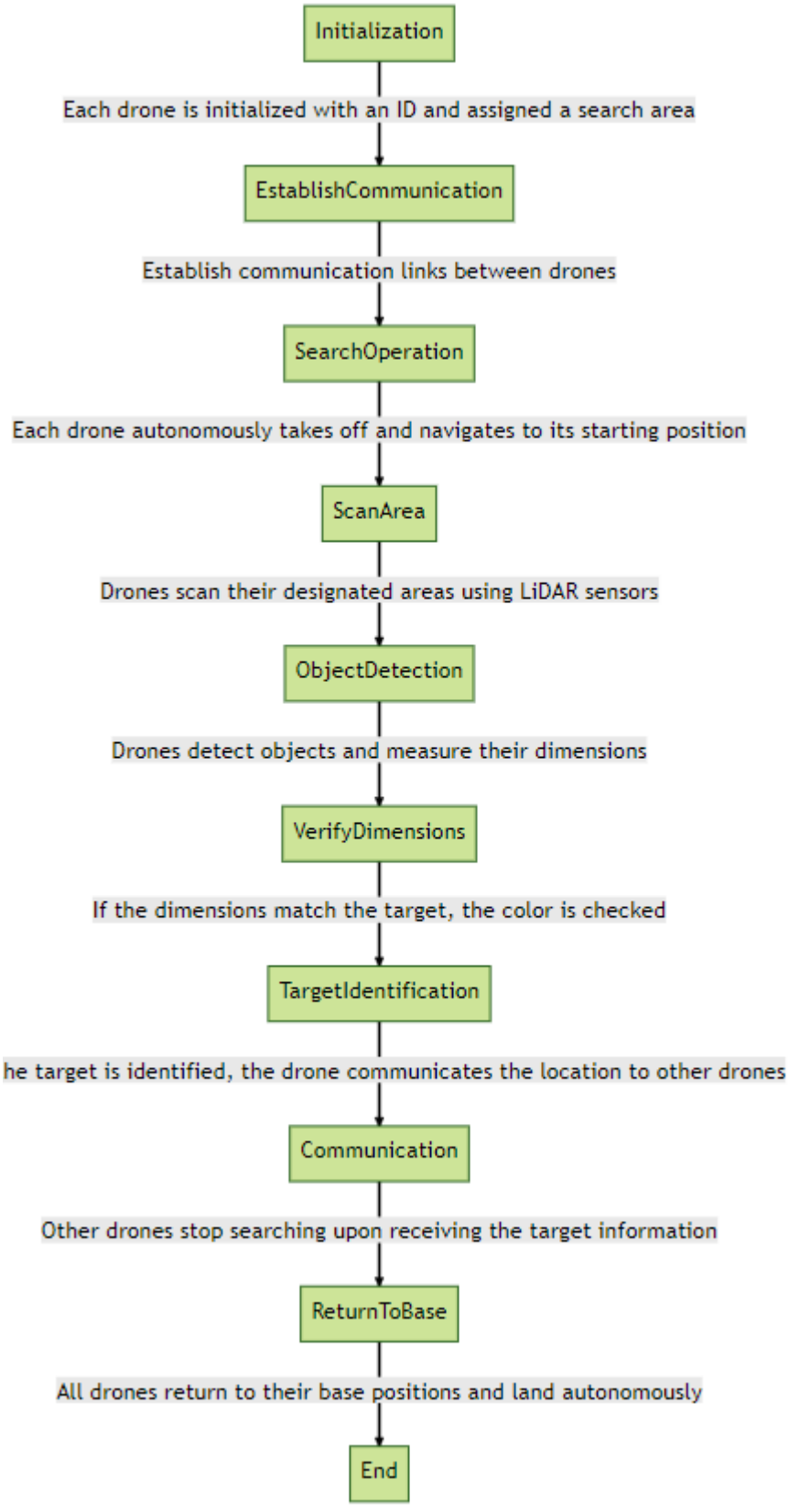
2. Other drones recieve the target location and stop their search.

Pseudocode for Communication :

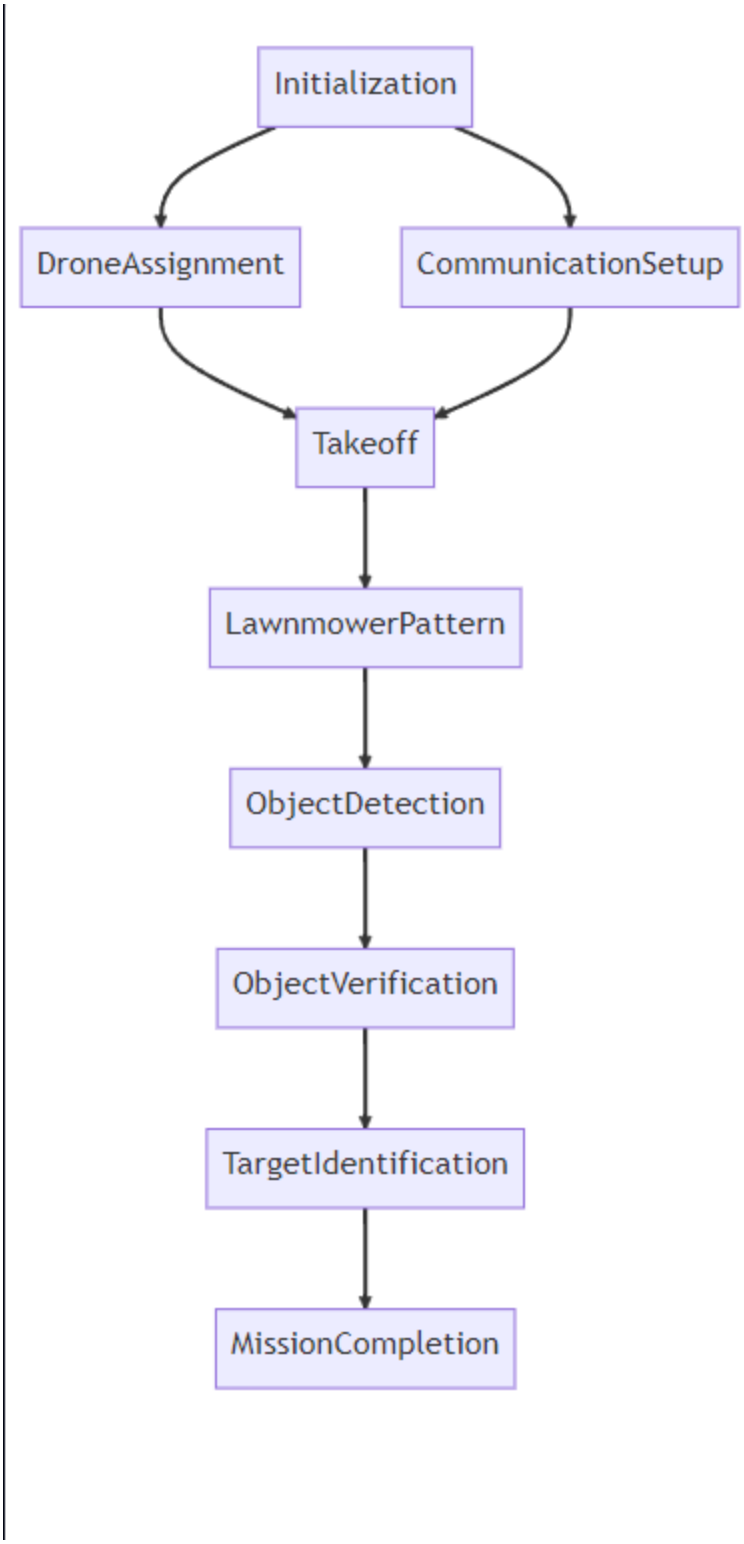
```
void communicateTargetLocation (const Drone &drone,
                                vector<Drone> &drones {
    for (auto &d : drones) {
        d.recieveTargetLocation (drone.getCurrentLocation())
        d.stopSearch();
    }
}
}
```

DIAGRAMS & FLOWCHARTS

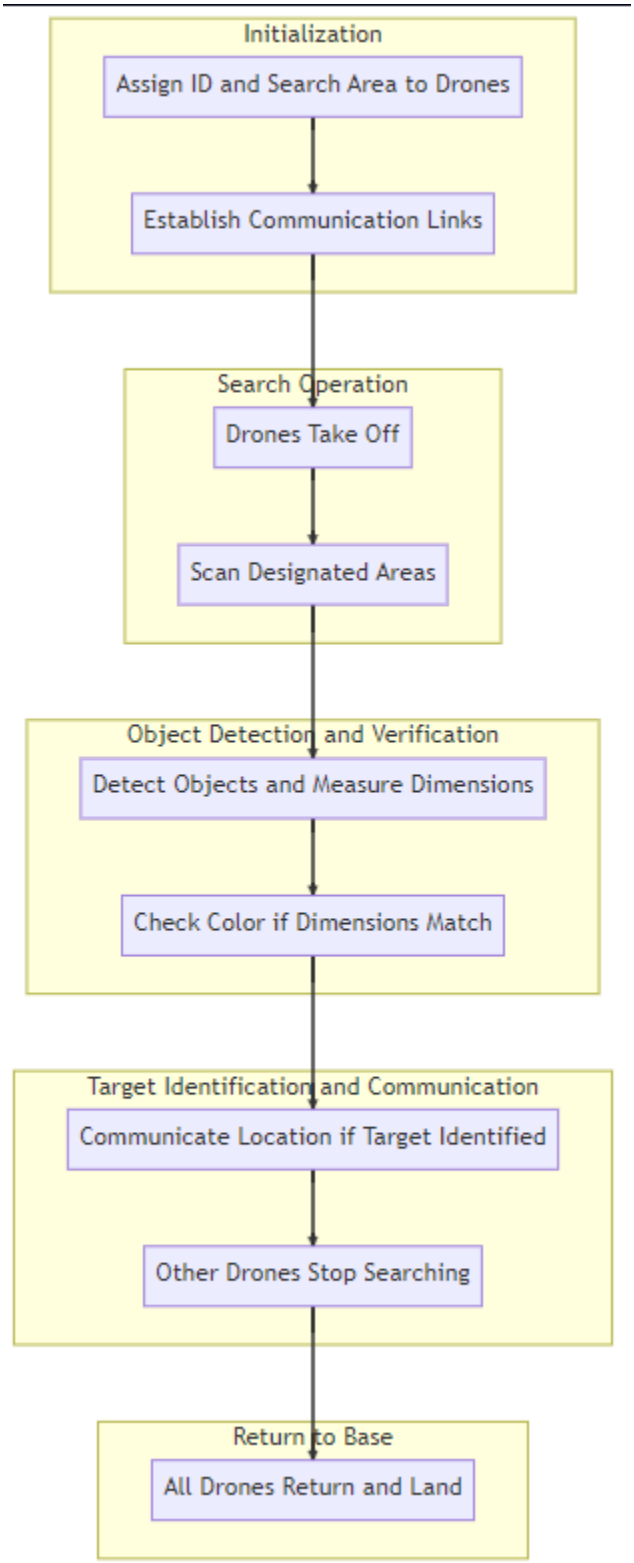
FLOW DIAGRAM:



DRONE FLIGH PATH DIAGRAM:



DRONE SEARCH DIAGRAM:



C++

PROGRAM

(LLD)

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <string>
#include "Drone.h" // Assuming that this header file contains the drone class with necessary methods

// Defining the constants
const double TARGET_HEIGHT = 15.0;
const double TARGET_WIDTH = 15.0;
const double TARGET_DEPTH = 15.0;
const std::string TARGET_COLOR = "Green";

// Function prototypes
void initializeDrones(std::vector<Drone> &drones);
void searchArea(Drone &drone);
bool verifyDimensions(double height, double width, double depth);
bool verifyColor(const std::string &color);
void communicateTargetLocation(const Drone &drone, std::vector<Drone> &drones);

int main() {
    std::vector<Drone> drones(3);
    initializeDrones(drones);

    for (int i = 0; i < drones.size(); ++i) {
        searchArea(drones[i]);
    }

    return 0;
}

void initializeDrones(std::vector<Drone> &drones) {
    // Initializing each drone with ID and search area
    for (int i = 0; i < drones.size(); ++i) {
        drones[i].setID(i + 1);
        drones[i].setSearchArea(/* coordinates based on i */);
        drones[i].takeOff();
    }
}

void searchArea(Drone &drone) {
    while (true) {
        auto object = drone.scanObject();
        if (object.height > 0) {
            if (verifyDimensions(object.height, object.width, object.depth)) {
                std::string color = drone.detectColor(object);
                if (verifyColor(color)) {
                    communicateTargetLocation(drone, drones);
                    return;
                }
            }
        }
        drone.continueSearch();
    }
}

bool verifyDimensions(double height, double width, double depth) {
    return (height == TARGET_HEIGHT && width == TARGET_WIDTH && depth == TARGET_DEPTH);
}

bool verifyColor(const std::string &color) {
    return (color == TARGET_COLOR);
}

void communicateTargetLocation(const Drone &drone, std::vector<Drone> &drones) {
    for (auto &d : drones) {
        if (d.getID() != drone.getID()) {
            d.receiveTargetLocation(drone.getCurrentLocation());
            d.stopSearch();
        }
    }
}
```