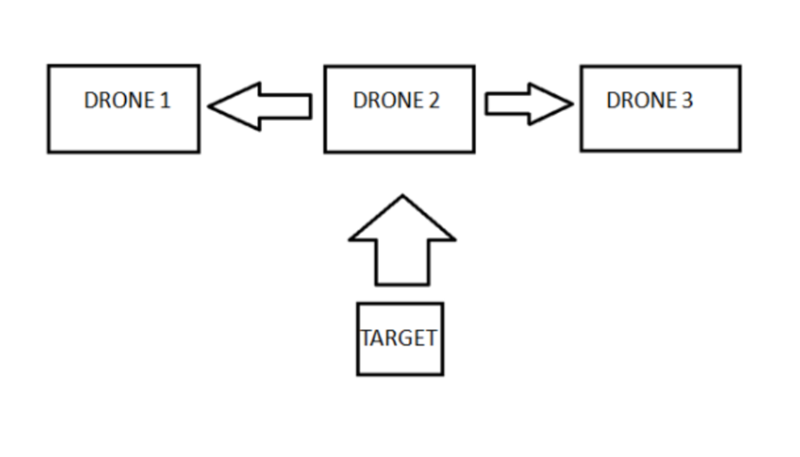
## d) Three drones are made autonomous and these drones move in the target specified location according to the code given to each drone, where the complete search of the specified location for the target and prevention of collision of any two drones are taken care of. Each drone has lidar sensor by which if they come in front of any object of height 15cm, then the drone stops and measures the other dimensions of the object. If the measurement taken by lidar sensor is 15cm for both the other dimensions, then that drone uses color sensor to detect the color and as the target color is specified as green, if the observed color by the color sensor attached is not green, then the drone again continue its journey of search until it encounters an object of height 15cm and checks the object by the above-mentioned mechanism. If the color sensed by color sensor is green in the above case, then it sends this information immediately to other drones through swarm technology. As the required task is accomplished the other two drones stop searching. The block diagram is shown below.



## As anyone of the drone finds the target it sends the information to other two drones.

**Solution:**

## Autonomous Drones Target Search and Communication System

### Overview

This project involves a fleet of three autonomous drones designed to search a specified area for a target object. Each drone is equipped with sensors to detect, identify, and communicate the location of the target to coordinate their search efforts and avoid collisions.

### Components

#### Drones: Each equipped with:

**Lidar Sensor**

* **Function**: Measures distance to objects by emitting laser beams and calculating the time taken for the reflection to return.
* **Integration**: Connect the Lidar sensor to the drone’s microcontroller. Use libraries to interface and read data from the sensor.
* **Usage in Project**: Detects objects, measures their height, width, and depth to identify potential targets.

**Color Sensor**

* **Function**: Uses light to detect the color of an object.
* **Integration**: Connect the Color sensor to the drone’s microcontroller. Use libraries to interface and read data from the sensor.
* **Usage in Project**: Confirms if the detected object’s color matches the target color (green).

**Communication Module**

**Function**

The communication module's primary function is to enable wireless communication between the drones. This communication is crucial for coordinating the search efforts, sharing information about the detected target, and preventing collisions.

**Integration**

1. **Hardware Connection**:
   * Connect the communication module (e.g., NRF24L01 or ESP8266) to the drone’s microcontroller (e.g., Arduino or a compatible microcontroller).
   * Ensure proper wiring: VCC to 3.3V, GND to GND, CE to a digital pin (e.g., D9), CSN to another digital pin (e.g., D10), SCK to the SPI clock pin, MOSI to the SPI master-out pin, and MISO to the SPI master-in pin.
2. **Software Integration**:
   * Use libraries such as RF24 for NRF24L01 or ESP8266WiFi for ESP8266 to manage the sending and receiving of messages.
   * Initialize the communication module in the setup function of the microcontroller’s code.

**Usage in Project**

The communication module is used to send a message to the other drones when one drone finds the target. This message instructs the other drones to halt their search.

**Example with NRF24L01**

**Hardware Setup**

Assuming we use the NRF24L01 module, the connections to an Arduino Uno would be as follows:

* VCC to 3.3V
* GND to GND
* CE to Digital Pin 9
* CSN to Digital Pin 10
* SCK to Digital Pin 13
* MOSI to Digital Pin 11
* MISO to Digital Pin 12

**Software Example**

1. **Library Installation**: Ensure you have the RF24 library installed in your Arduino IDE.
2. **Code for the Transmitter (Drone Finding the Target)**:

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

RF24 radio(9, 10); // CE, CSN pins

const byte address[6] = "00001";

void setup() {

Serial.begin(9600);

radio.begin();

radio.openWritingPipe(address);

radio.setPALevel(RF24\_PA\_HIGH);

radio.stopListening();

}

void loop() {

// Dummy condition for detecting the target

bool targetFound = true;

if (targetFound) {

const char text[] = "Target Found";

radio.write(&text, sizeof(text));

Serial.println("Target Found message sent");

while (true) {

// Stop drone

}

}

}

1. **Code for the Receiver (Other Drones)**:

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

RF24 radio(9, 10); // CE, CSN pins

const byte address[6] = "00001";

void setup() {

Serial.begin(9600);

radio.begin();

radio.openReadingPipe(0, address);

radio.setPALevel(RF24\_PA\_HIGH);

radio.startListening();

}

void loop() {

if (radio.available()) {

char text[32] = "";

radio.read(&text, sizeof(text));

Serial.println(text);

if (strcmp(text, "Target Found") == 0) {

Serial.println("Stopping search...");

while (true) {

// Stop drone

}

}

}

}

**Explanation of the Example**

1. **Initialization**:
   * The RF24 library is used to handle communication with the NRF24L01 module.
   * In the setup function, the communication module is initialized, and the communication pipe is opened using the address "00001".
2. **Transmitter Logic (Target Found)**:
   * The transmitter code checks a dummy condition for detecting the target (bool targetFound = true;).
   * When the target is found, a message "Target Found" is sent to the other drones using radio.write(&text, sizeof(text));.
   * The drone stops its operation by entering an infinite loop.
3. **Receiver Logic (Other Drones)**:
   * The receiver code continuously listens for incoming messages.
   * When a message is received, it is read and compared to "Target Found".
   * Upon receiving the "Target Found" message, the drone stops its search by entering an infinite loop.

**Benefits of This Communication System**

1. **Coordination**: Ensures all drones are aware when the target is found, preventing redundant search efforts.
2. **Efficiency**: Reduces search time by halting the search process immediately once the target is detected.
3. **Safety**: Prevents collisions by enabling real-time communication about each drone’s status and movements.

By integrating the communication module in this manner, the autonomous drones can effectively share critical information and work collaboratively to complete the target search mission.

**Target Specification:**

* **Height**: 15 cm.
* **Color**: Green.

**Operational Workflow**

**Initialization:**

1. **Assign Search Areas**: Each drone is assigned a specific section of the search area to cover.
2. **Set Communication Protocol**: Establish a protocol for inter-drone data exchange to coordinate the search.

**Object Detection and Target Identification in Autonomous Drones**

**Lidar Sensor Scan**

* **Function**:
  + Each drone utilizes a Lidar sensor to scan its surroundings.
  + The Lidar sensor emits laser pulses and measures the time it takes for the pulses to bounce back, creating a 3D map of the environment.
* **Height Detection**:
  + **Condition**: When the Lidar sensor detects an object with a height of 15 cm.
  + **Action**: The drone stops its current movement to focus on the detected object.

**Dimension Measurement**

* **Function**:
  + After detecting the object, the drone proceeds to measure its width and depth using the Lidar sensor.
* **Criteria**:
  + **Requirement**: Both the width and depth must measure 15 cm.
  + **Decision**: If these dimensions match the specified criteria, the drone continues to the next step.

**Color Detection**

* **Color Sensor Check**:
  + **Function**: The drone employs a color sensor to ascertain the color of the detected object.
* **Target Confirmation**:
  + **Criteria**: The specified target color is green.
  + **Action**: If the color sensor identifies the object's color as green, the drone confirms it as the target object.

**Swarm Communication**

* **Target Identification**:
  + **Communication Protocol**: Utilizing a communication module (e.g., ESP8266 or NRF24L01), the drone that identifies the target sends a message to the other drones.
* **Message Content**:
  + The message typically includes a simple identifier like "Target Found".
* **Impact on Other Drones**:
  + **Response**: Upon receiving the "Target Found" message, the other drones immediately cease their search operations.

**Collision Avoidance:**

1. **Environmental Monitoring**: Lidar sensors monitor the environment for obstacles.
2. **Flight Path Adjustment**: Drones adjust their flight paths to avoid collisions based on Lidar data.

#### Pseudocode

#include <LidarSensor.h>

#include <ColorSensor.h>

#include <CommunicationModule.h>

LidarSensor lidar;

ColorSensor colorSensor;

CommunicationModule commModule;

const int TARGET\_HEIGHT = 15; // Target height in cm

const int TARGET\_DIMENSION = 15; // Target dimension in cm (width and depth)

const String TARGET\_COLOR = "green"; // Target color

void setup() {

Serial.begin(115200); // Initialize serial communication for debugging

lidar.begin(); // Initialize the lidar sensor

colorSensor.begin(); // Initialize the color sensor

commModule.begin(); // Initialize the communication module

}

void loop() {

if (lidar.detectObjectHeight() == TARGET\_HEIGHT) { // Check if object height matches the target height

if (lidar.detectObjectWidth() == TARGET\_DIMENSION && lidar.detectObjectDepth() == TARGET\_DIMENSION) { // Check if object dimensions match the target dimensions

String detectedColor = colorSensor.detectColor(); // Detect the color of the object

if (detectedColor == TARGET\_COLOR) { // Check if the detected color matches the target color

commModule.sendMessage("Target Found"); // Send a message to other drones indicating the target has been found

while (true) {

// Stop the drone from continuing its search

delay(1000); // Keep the loop running to simulate the drone stopping

}

}

}

}

// Collision Avoidance Code

avoidCollision();

// Continue Search

continueSearch();

}

void avoidCollision() {

// Lidar-based collision avoidance logic

// This function will check for obstacles and adjust the drone's path to avoid collisions

if (lidar.detectObstacle()) {

// Implement obstacle avoidance logic here

// Example: Adjust drone's path to avoid the obstacle

}

}

void continueSearch() {

// Logic for continuing the search

// This function will control the drone's movement to continue searching for the target

// Example: Move the drone forward, adjust the flight path, etc.

}

**Explanation**

**Libraries and Object Instantiation**

#include <LidarSensor.h>

#include <ColorSensor.h>

#include <CommunicationModule.h>

LidarSensor lidar;

ColorSensor colorSensor;

CommunicationModule commModule;

* **LidarSensor.h**: Assumed to be a library for interacting with the Lidar sensor.
* **ColorSensor.h**: Assumed to be a library for interacting with the Color sensor.
* **CommunicationModule.h**: Assumed to be a library for communication between drones.
* **LidarSensor lidar**: Creates an instance of the Lidar sensor.
* **ColorSensor colorSensor**: Creates an instance of the Color sensor.
* **CommunicationModule commModule**: Creates an instance of the Communication module.

**Constants for Target Specifications**

const int TARGET\_HEIGHT = 15; // Target height in cm

const int TARGET\_DIMENSION = 15; // Target dimension in cm (width and depth)

const String TARGET\_COLOR = "green"; // Target color

* These constants define the target's height, dimensions, and color that the drone will look for.

**Setup Function**

void setup() {

Serial.begin(115200); // Initialize serial communication for debugging

lidar.begin(); // Initialize the lidar sensor

colorSensor.begin(); // Initialize the color sensor

commModule.begin(); // Initialize the communication module

}

* **Serial.begin(115200)**: Starts serial communication at a baud rate of 115200 for debugging purposes.
* **lidar.begin()**: Initializes the Lidar sensor.
* **colorSensor.begin()**: Initializes the Color sensor.
* **commModule.begin()**: Initializes the Communication module.

**Main Loop Function**

void loop() {

if (lidar.detectObjectHeight() == TARGET\_HEIGHT) { // Check if object height matches the target height

if (lidar.detectObjectWidth() == TARGET\_DIMENSION && lidar.detectObjectDepth() == TARGET\_DIMENSION) { // Check if object dimensions match the target dimensions

String detectedColor = colorSensor.detectColor(); // Detect the color of the object

if (detectedColor == TARGET\_COLOR) { // Check if the detected color matches the target color

commModule.sendMessage("Target Found"); // Send a message to other drones indicating the target has been found

while (true) {

// Stop the drone from continuing its search

delay(1000); // Keep the loop running to simulate the drone stopping

}

}

}

}

// Collision Avoidance Code

avoidCollision();

// Continue Search

continueSearch();

}

* **if (lidar.detectObjectHeight() == TARGET\_HEIGHT)**: Checks if the detected object's height is 15 cm.
* **if (lidar.detectObjectWidth() == TARGET\_DIMENSION && lidar.detectObjectDepth() == TARGET\_DIMENSION)**: Checks if the detected object's width and depth are both 15 cm.
* **String detectedColor = colorSensor.detectColor()**: Detects the color of the object.
* **if (detectedColor == TARGET\_COLOR)**: Checks if the detected color is green.
* **commModule.sendMessage("Target Found")**: Sends a message to the other drones that the target has been found.
* **while (true) { delay(1000); }**: Stops the drone from continuing its search by entering an infinite loop with a delay.

**Collision Avoidance Function**

void avoidCollision() {

// Lidar-based collision avoidance logic

// This function will check for obstacles and adjust the drone's path to avoid collisions

if (lidar.detectObstacle()) {

// Implement obstacle avoidance logic here

// Example: Adjust drone's path to avoid the obstacle

}

}

* **avoidCollision()**: A placeholder function where logic for avoiding collisions using the Lidar sensor is implemented.
* **if (lidar.detectObstacle())**: Checks if an obstacle is detected.
* **Adjust drone's path to avoid the obstacle**: Placeholder for the actual logic to avoid collisions.

**Continue Search Function**

void continueSearch() {

// Logic for continuing the search

// This function will control the drone's movement to continue searching for the target

// Example: Move the drone forward, adjust the flight path, etc.

}

* **continueSearch()**: A placeholder function where the logic for continuing the search is implemented.
* **Control the drone's movement**: Placeholder for the actual movement logic to continue the search.

**Additional Notes**

* The specific implementations of the Lidar sensor, Color sensor, and Communication module libraries are not provided. You will need to use appropriate libraries compatible with your hardware and fill in the logic for functions like detectObjectHeight, detectObjectWidth, detectObjectDepth, detectColor, and sendMessage.
* The collision avoidance and search continuation logic need to be implemented based on the specific requirements and behaviors of your drone.
* The provided code is a high-level example to illustrate the structure and workflow. Further details and adjustments might be necessary based on the actual hardware and libraries you use.

**Communication Protocol**

**Message Format**

* **Purpose**:
  + The message format should prioritize simplicity and clarity to ensure that communication between drones is effective and straightforward.
* **Example**:
  + Use concise messages like "Target Found" to convey critical information swiftly.
  + Avoid complex or ambiguous messages to minimize interpretation errors.

**Error Handling**

* **Importance**:
  + Implement robust error-handling mechanisms to maintain reliability in message transmission and reception.
  + Errors can arise from network issues, interference, or module malfunctions.
* **Strategies**:
  + **Acknowledgment**: Require acknowledgment messages to confirm receipt of critical information.
  + **Retry Mechanism**: Implement mechanisms to resend messages in case of transmission failures.
  + **Timeouts**: Set timeouts to handle situations where responses are not received within expected timeframes.

**Implementation Steps**

**Programming Environment**

* **Selection**:
  + Use a suitable environment like the Arduino IDE or similar platforms that support microcontroller programming.
  + Ensure compatibility with the chosen communication modules (e.g., NRF24L01, ESP8266).

**Sensor Integration**

* **Lidar and Color Sensors**:
  + Integrate sensors using compatible libraries that facilitate data acquisition and processing.
  + Implement sensor calibration routines to ensure accuracy in detecting object dimensions and colors.

**Communication Module Setup**

* **Module Selection**:
  + Choose communication modules (e.g., NRF24L01 or ESP8266) based on range, data rate, and power requirements.
  + Configure modules for wireless communication between drones, adhering to specified protocols and standards.

**Testing and Validation**

* **Simulations**:
  + Conduct simulations in controlled environments (e.g., software-based simulations) to validate algorithmic logic and message exchange protocols.
  + Use simulated scenarios to test edge cases and verify system robustness.
* **Field Testing**:
  + Perform real-world tests to evaluate system performance under actual operating conditions.
  + Validate communication reliability, sensor accuracy, and overall system responsiveness in dynamic environments.

**Calibration**

* **Routine Maintenance**:
  + Regularly calibrate sensors to maintain accuracy and reliability over time.
  + Adjust sensor settings based on environmental factors (e.g., light conditions, temperature) to optimize performance.

**Benefits and Applications**

* **Efficiency**:
  + Streamlined communication protocols and error handling mechanisms ensure efficient data exchange and mission execution.
* **Reliability**:
  + Robust testing and calibration practices enhance system reliability, critical for applications such as autonomous navigation, surveillance, and search-and-rescue missions.
* **Scalability**:
  + Scalable architecture allows for integration into larger drone fleets or autonomous systems, facilitating coordinated operations and collaborative tasks.

By following these implementation steps and focusing on effective communication protocols, error management, and thorough testing procedures, autonomous drone systems can achieve high performance and reliability in diverse operational scenarios.