1. Conventional Takeoff

Goal: Achieve a smooth transition from the runway to stable flight.

Steps:

- **Initial Configuration**:
    - Ensure all systems are calibrated, including the control surfaces and sensors (IMUs, barometers).
- **Takeoff Algorithm**:
    - **Throttle Control**: Gradually increase throttle to a predetermined value to achieve lift-off speed.
    - **Pitch Control**: Adjust the elevator to maintain the correct pitch angle for takeoff (typically around 5-10 degrees).
    - **Roll Control**: Keep the aircraft level until it reaches a safe altitude, then adjust for any yawing or rolling tendencies.
- **Stability Check**:
    - Implement PID (Proportional-Integral-Derivative) controllers for pitch, roll, and yaw to maintain stability during takeoff.
    - Monitor altitude and airspeed to confirm a successful transition to stable flight.

2. Payload Delivery

Goal: Deliver the payload to the Designated Landing Zone (DLZ) accurately.

Steps:

- **Trajectory Calculation**:
    - Use GPS coordinates to determine the DLZ's position.
    - Calculate the optimal flight path considering factors like wind speed and direction.

Determine the DLZ Position Using GPS Coordinates

1. **Obtain GPS Coordinates**:

Get the latitude and longitude for the DLZ from GPS data. For example, the coordinates could be in the format:
mathematica
Copy code

```
Latitude: 37.7749° N
Longitude: 122.4194° W
```

    ○

2. **Convert Coordinates to Local System (if needed)**:
    - If your aircraft uses a local coordinate system, convert the GPS coordinates to that system (e.g., UTM, ENU).

Step 2: Calculate the Optimal Flight Path

1. **Define Starting Point**:
   - Identify the starting GPS coordinates of the aircraft.
2. **Gather Wind Data**:
   - Collect real-time data on wind speed and direction at various altitudes. This data can be obtained from weather APIs or onboard sensors.
3. **Calculate Waypoints**:
   - Use the GPS coordinates of the DLZ and the starting point to create waypoints. Depending on the distance and safety, you may want to define multiple waypoints along the path.
4. **Flight Path Calculation**:
   - **Basic Approach**:
     - Use a simple line equation to connect the starting point to the DLZ.
   - **Advanced Approach**:
     - Implement algorithms like A* or Dijkstra's for more complex environments with obstacles.
     - Consider wind effects: Adjust the flight path by calculating the optimal heading to compensate for wind drift. This can involve vector addition of the aircraft's intended direction and the wind vector.
5. **Adjust for Altitude**:
   - Depending on your aircraft's capabilities, adjust the altitude during different segments of the flight to optimize performance and stability.

```
import numpy as np

def calculate_flight_path(start_coords, dlz_coords, wind_speed, wind_direction):
    # Convert GPS coordinates to radians
    lat1, lon1 = np.radians(start_coords)
    lat2, lon2 = np.radians(dlz_coords)

    # Calculate the great-circle distance (Haversine formula)
    d_lat = lat2 - lat1
    d_lon = lon2 - lon1
    a = np.sin(d_lat / 2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(d_lon / 2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    radius = 6371  # Earth radius in kilometers
    distance = radius * c

    # Adjust for wind (simple vector addition)
    wind_vector = np.array([wind_speed * np.cos(np.radians(wind_direction)),
                            wind_speed * np.sin(np.radians(wind_direction))])

    # Calculate optimal heading
    # Assuming aircraft's airspeed is 10 m/s
    aircraft_speed = 10  # m/s
    flight_vector = np.array([distance, 0])  # Simplified
```

```
    # Adjust flight vector with wind
    optimal_vector = flight_vector + wind_vector

    return optimal_vector, distance

# Example usage
start_coords = (37.7749, -122.4194)  # San Francisco
dlz_coords = (37.7750, -122.4184)    # Example DLZ
wind_speed = 5                       # m/s
wind_direction = 90                  # Degrees

optimal_vector, flight_distance = calculate_flight_path(start_coords, dlz_coords, wind_speed, wind_direction)
print("Optimal Flight Vector:", optimal_vector)
print("Flight Distance:", flight_distance, "km")
```

Step 1: Define Descent Parameters

1. **Descent Angle**: Decide on a target descent angle (e.g., 3 degrees).
2. **Descent Rate**: Determine the desired vertical speed (e.g., -2 m/s).
3. **Target Altitude**: Set the target altitude (e.g., the altitude of the DLZ).

Step 2: Gather Sensor Data

1. **Altitude**: Use a barometer or GPS to obtain the current altitude.
2. **Position**: Use GPS to get the current position of the aircraft.
3. **Pitch Angle**: Use an IMU (Inertial Measurement Unit) to monitor the pitch angle.

Step 3: Implement the Descent Algorithm

The descent algorithm will adjust throttle and pitch based on the current altitude, target altitude, and descent angle.

1. **Calculate Desired Descent Rate**:
    - Use the desired descent angle to calculate the horizontal and vertical components of the flight path.
2. **Control Throttle and Pitch**:
    - If the current altitude is above the target altitude, decrease throttle and adjust pitch down to maintain the descent rate.
    - If the aircraft is descending too quickly, adjust the throttle up and pitch up slightly to slow the descent.
    - import numpy as np
    -
    - class Aircraft:
    -     def __init__(self, current_altitude, target_altitude, throttle, pitch):

```
        self.current_altitude = current_altitude
        self.target_altitude = target_altitude
        self.throttle = throttle  # Range from 0 to 1
        self.pitch = pitch        # Degrees

    def calculate_descent(self, desired_descent_angle):
        # Convert angle to radians
        descent_angle_rad = np.radians(desired_descent_angle)

        # Calculate desired vertical speed (m/s)
        desired_vertical_speed = 2 * np.tan(descent_angle_rad)

        # Control logic
        if self.current_altitude > self.target_altitude:
            if desired_vertical_speed < 0:  # If descending
                self.throttle -= 0.1  # Decrease throttle
                self.pitch -= 2       # Pitch down to descend faster
            else:
                self.throttle += 0.1  # Increase throttle
                self.pitch += 2       # Pitch up to slow descent

            # Clamp values to stay within limits
            self.throttle = max(0, min(self.throttle, 1))
            self.pitch = max(-10, min(self.pitch, 10))  # Example pitch limits

        # Update current altitude
        self.current_altitude += self.throttle * desired_vertical_speed

        return self.current_altitude, self.throttle, self.pitch

# Example usage
aircraft = Aircraft(current_altitude=1000, target_altitude=500, throttle=0.5, pitch=0)

# Simulate descent
for _ in range(100):
    current_altitude, throttle, pitch = aircraft.calculate_descent(desired_descent_angle=3)
    print(f"Altitude: {current_altitude:.2f} m, Throttle: {throttle:.2f}, Pitch: {pitch:.2f} degrees")

```

Payload Release:

- Implement a mechanism (servo-controlled or electronic release) that activates at a predetermined altitude or distance from the DLZ.
- Use a timer or distance sensor to ensure timely release.

3. Payload Capture

Goal

Locate and retrieve the previously delivered payload autonomously.

Steps

1. Payload Recognition

Equipment:

- **Camera:** A lightweight camera (e.g., Raspberry Pi Camera, USB webcam) mounted on the aircraft.

Marker System: Use bright, easily recognizable markers (e.g., colored circles, QR codes) on the payload. This helps with detection.

Algorithm:

- **Flight Path Adjustment:**
  - Use the detected payload coordinates to adjust the aircraft's position.
  - Implement a feedback loop using PID (Proportional-Integral-Derivative) control for smooth adjustments.

PID Control Example:

- Set targets based on the recognized payload position.
- Continuously adjust throttle, pitch, and yaw to minimize the error between the aircraft's current position and the target.

Capture Mechanism

Design:

- **Mechanical System:**
  - Use a claw or a hook designed to securely grip the payload.
  - Ensure that the mechanism is lightweight and can be activated remotely or autonomously.

Activation:

- Use a servo motor controlled by the flight controller to operate the capture mechanism.
- The mechanism should only activate when the aircraft is within a certain range of the payload.

```
def activate_capture_mechanism():
        servo_angle = 90  # Adjust this angle to open/close the claw
    servo.write(servo_angle)
```

Qr code detector

```
import cv2

detector = cv2.QRCodeDetector()

image = cv2.imread('qr_code_image.jpg')

data, bbox, _ = detector(image)

if bbox is not None:
    for i in range(len(bbox)):
        cv2.line(image, tuple(bbox[i][0]), tuple(bbox[i][1]), (0, 255, 0), 2)
        cv2.line(image, tuple(bbox[i][1]), tuple(bbox[i][2]), (0, 255, 0), 2)
        cv2.line(image, tuple(bbox[i][2]), tuple(bbox[i][3]), (0, 255, 0), 2)
        cv2.line(image, tuple(bbox[i][3]), tuple(bbox[i][0]), (0, 255, 0), 2)

# Show the detected QR code
cv2.imshow('QR Code Detection', image)
cv2.waitKey(0)
cv2.destroyAllWindows()

if data:
    print("Decoded Data:", data)
```

Camera Options:

- **Raspberry Pi Camera Module V2**: 8 MP, capable of 1080p video.
- **USB Webcam**: Many USB webcams are compatible; look for those that support high resolutions (1080p).
- **Pi HQ Camera**: A high-quality camera module with interchangeable lenses for better optical performance.

1. Selecting Sensors

A. Navigation Sensors

1. **GPS Module**: For real-time location tracking and navigation.
    - **Example**: NEO-6M GPS module.
    - **Use**: Provides latitude and longitude for waypoints and landing zone navigation.
2. **Ultrasonic Distance Sensors**: For altitude measurement and obstacle detection.
    - **Example**: HC-SR04 ultrasonic sensor.
    - **Use**: Measures distance to the ground or nearby obstacles during descent.
3. **Inertial Measurement Unit (IMU)**: For orientation and acceleration data.

    - **Example**: MPU-6050.
    - **Use**: Helps stabilize the aircraft by providing pitch, roll, and yaw data.

B. Payload Retrieval Sensors

1. **Camera Module**: For visual recognition of the payload.
    - **Example**: Raspberry Pi Camera Module.
    - **Use**: Can be used for image processing to identify and align with the payload

-