**DRONE NEST**

**INDEX**

1. Problem Statement
2. Drone Assembly
3. Precision-Landing
4. Docking Station
5. FAQ
6. Abbreviations

**PROBLEM STATEMENT**

The problem statement to be solved is to create a UAV docking station solution, which can provide the following deliverables:

* Autonomous Precision Landing
* Secure Storage of UAV (in all weather conditions)
* A UAV with 1 Hour flight time with 5 minutes Battery change time
* Multiple Batteries Slots (Chargeable)

We will be focusing on the following aspects of the solution:

* Autonomous Precision Landing
* POC of the Docking Station

**Team**

* Vayam Jain – Project Lead
* Kanishk Jha – Software Developer
* Ayush Yadu – Mechanical and Design Specialist

**AUTONOMOUS PRECISION LANDING**

*Situation at Hand –* After a UAV successfully completes its mission list, or when the battery voltage level drops to the failsafe level, the UAV must go back to the geolocation of the Docking Station, aligns its heading according to the Docking Station’s alignment, and precisely land on the base plate inside the Docking Station.

*Tasks to be completed* -

* Preparing the OBC, and installing all the dependencies
* Connecting and Testing the Camera and Gimbal
* Solution using ArUco Marker detection library of OpenCV
* Simulation Testing
* Creating sample missions and real-time testing

**Requirements**

*Hardware*

1. Raspberry Pi 5 (RAM – 8GB)
2. Sandisk SD Card – 64 GB
3. SIYI A8 mini camera-gimbal system
4. CyberOne lite UAV setup

*Software*

1. Ubuntu (18.04 LTS) (Simulation)
2. Gazebo version 8.x or greater (Recomended 9.6) (Simulation)
3. ROS melodic (Required to work with Gazebo) (Simulation)
4. MAVROS (Simulation)
5. Raspbian v12(bookworm) | Kernel v6.6 | System: 32-bit
6. OpenCV-Python v4.9
7. Numpy, Math, Time
8. Pymavlink
9. Dronekit-Python
10. ArduPilot v-lts
11. Mavproxy v-lts

**Preparing the OBC**

Step 1: Download Raspi imager -> [Download Link](https://www.raspberrypi.com/software/)

Step 2: Take a SD card (satisfying the above given specs) and format it

Step 3: Flash the Raspi image as mentioned above using the Raspi Imager app

Step 4: Install the dependencies listed in *“Software”* section (6 - 12) | Alternate: Run the requirements file to download all the required libraries

**Connecting and Testing Camera-Gimbal System**

The hardware connections are explained in the drone assembly section.

Once connected, follow the following link to enable camera and gimbal control via Pixhawk cube orange+ -> [Parameters Setting](https://ardupilot.org/copter/docs/common-siyi-zr10-gimbal.html)

Note: Use Mission Planner on windows and set the params till [RC8\_OPTION](https://ardupilot.org/copter/docs/parameters.html#rc8-option) = 163. Make sure to set the respective controls on the RC Tx as well.

Connect camera to the OBC using the micro-HDMI port (camera) and USB 3.0 (OBC) using a video capture card.

Open the following link to test the camera, allow the required permissions -> [Camera Test](https://webcamtests.com/check)

To test the Gimbal, open terminal and run the gimbalTest.py script from the GitHub repository

* Make sure you are in the GitHub repository directory in terminal
* cd componentTesting
* python3 gimbalTest.py

**Autonomous Precision Landing**

*Algorithm*

1. Import Required Libraries:
   1. Import OpenCV (cv2), NumPy (np), and DroneKit libraries.
   2. Import time, math, and pymavlink.
2. Define the ArUcoDetector Class:
   1. Initialization (\_\_init\_\_ method):
      1. Initialize the ArUco dictionary and detector parameters.
      2. Create an ArUco detector object.
      3. Connect to the drone using DroneKit.
      4. Set the desired takeoff altitude to 8 meters and call arm\_and\_takeoff to take off.
      5. Retrieve the initial altitude from the lidar or the drone's global location.
      6. Set the landing altitude to 1 meter.
      7. Send an initial velocity of (2, 0, 0) to the drone.
      8. Start video capture using the default camera.
3. Video Capture (capture\_video method):
   1. Open the video capture from the default camera.
   2. Enter a loop to continuously read frames from the camera.
   3. For each frame, call process\_frame to process the frame for ArUco markers.
   4. Release the video capture after exiting the loop.
4. Process Frame (process\_frame method):
   1. Convert the frame to grayscale.
   2. Detect ArUco markers and retrieve their corners, IDs, and rejected points.
   3. Update the lidar altitude.
   4. If no ArUco markers are detected:
      1. If altitude is above the landing altitude, send a forward velocity.
      2. If altitude is below or equal to the landing altitude, set the drone to Return-to-Launch (RTL) mode.
   5. If an ArUco marker with ID 4 is detected:
      1. Draw the detected marker and calculate its center point.
      2. Compute displacement from the frame center.
      3. Calculate velocities based on the displacement using proportional coefficients.
      4. Adjust the drone's velocities:
         1. If within desired pixel error, set velocity to zero.
         2. Adjust velocity in the Z-axis based on altitude.
         3. If both X and Y displacements are within the desired pixel error, check if altitude requires landing or continue adjusting velocity.
5. Calculate Desired Pixel Error (calcDesiredPixelError method):
   1. Return a larger pixel error for higher altitudes to allow for greater error tolerance.
   2. Return a smaller pixel error for lower altitudes.
6. Arm and Takeoff (arm\_and\_takeoff method):
   1. Wait until the vehicle is armable.
   2. Set the vehicle mode to "GUIDED" and arm the vehicle.
   3. Wait until the vehicle is armed.
   4. Command the vehicle to take off to the target altitude.
   5. Continuously monitor the altitude until it is within 95% of the target altitude.
7. Send Velocity (send\_velocity method):
   1. Create a MAVLink message to set the velocity in the NED (North-East-Down) frame.
   2. Send the message to the vehicle.
8. Driver Code:
   1. Main Block (if \_\_name\_\_ == '\_\_main\_\_':):
      1. Try to create an instance of ArUcoDetector.
      2. If interrupted by a keyboard event (Ctrl+C), exit gracefully.

**Simulation Testing**

*Creating Environment*

Install “ardupilot\_gazebo\_roscam” from the following [link](https://github.com/r0ch1n/ardupilot_gazebo_roscam). Once the setup from the above link is complete add the “my\_code” folder in the “src/ardupilot\_gazebo” folder and rebuild the ROS package “ardupilot\_gazebo\_roscam.”

Add the given below lines to your bash.rc if not already

source /opt/ros/melodic/setup.bash

export GAZEBO\_MODEL\_PATH=~/ardupilot\_gazebo/models:$GAZEBO\_MODEL\_PATH

exportGAZEBO\_MODEL\_PATH=~/ardupilot\_gazebo\_roscam/src/ardupilot\_gazebo/models:$GAZEBO\_MODEL\_PATH

exportGAZEBO\_PLUGIN\_PATH=/usr/lib/x86\_64-linux-gnu/gazebo-9/plugins:$GAZEBO\_PLUGIN\_PATH

export GAZEBO\_PLUGIN\_PATH=/opt/ros/melodic/lib:$GAZEBO\_PLUGIN\_PATH

*Simulation Testing*

Open a Terminal and run these lines

source ~/ardupilot\_gazebo\_roscam/devel/setup.bash   
 roslaunch ardupilot\_gazebo iris\_with\_roscam.launch

In a second terminal run these lines

cd ~/ardupilot/ArduCopter   
 sim\_vehicle.py -f gazebo-iris --console --map --out 127.0.0.1:6969

In third Terminal run these lines

source ~/ardupilot\_gazebo\_roscam/devel/setup.bash   
 Rosrun ardupilot\_gazebo testCodePh3ROS.py

**Sample Mission and Results**

Mission 1: Takeoff from ground. Go to in a straight path. Detect ArUco Marker (on a plate) and land on the plate.

Status and Result: Success | Landing Accuracy - 98.5%

Observations: Drone was landing based on camera’s center, not the vehicle’s center.

Mission 2: Takeoff from the ground. Go in a straight path. Detect ArUco marker (inside the docking station box) and land inside the box.

Status and Result: Failed | Landing Accuracy – 98.5%

Observations: Single propeller brushed with dock station’s cover’s edge. This could be solved using a better opening mechanism for the Docking Station.

Mission 3: Dock Station open. Takeoff from inside the Dock Station. Perform mission. Return to Dock Station. Detect ArUco marker (inside the docking station box) and land inside the box.

Status and Result: TBD