Robotics Software Engineer Assignment

Git Repo Link

Tasks:

1. Simulation environment setup:

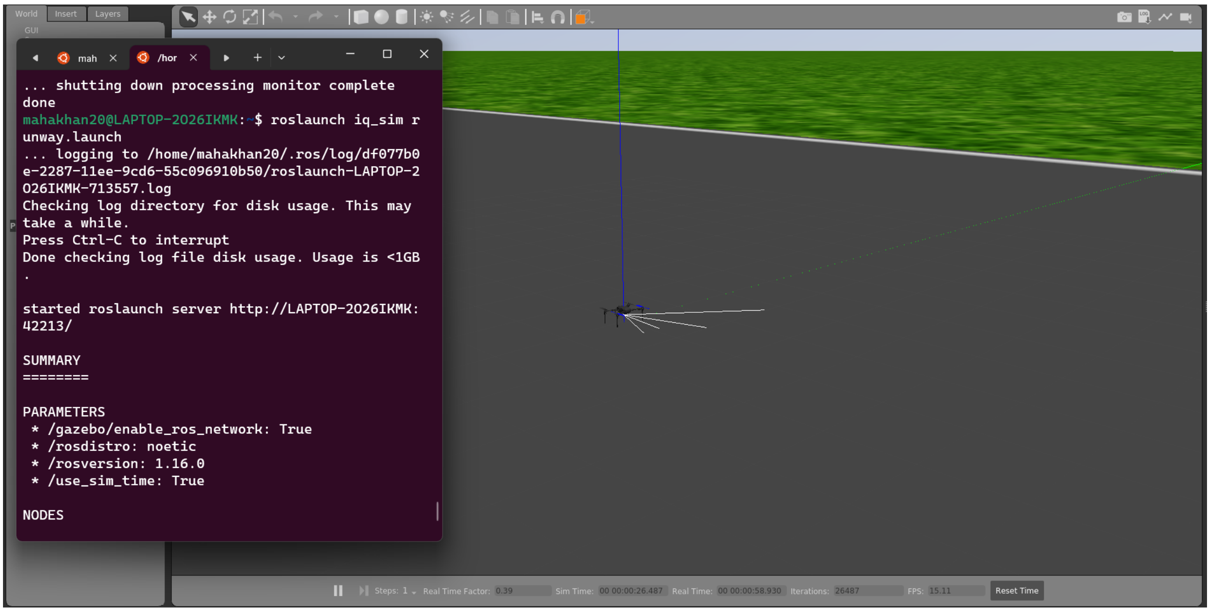
To set up the Gazebo environment, we recommend following the documentation provided by Ardupilot or PX4. Once the setup is complete, you can use libraries such as Dronekits, MAVSDK
and pymavlink to connect and control the drones. Implement basic functionality like takeoff, landing and going to a specified waypoint.

For my task I have chosen Ardupilot to get started with, the reason being easy to follow documentation on integrating it with ros using the Mavros package, also having some experience with ROS is easy to get started with communicating with Ardupilot and Gazebo at the same time . Below is a diagram showing the relationship among them.

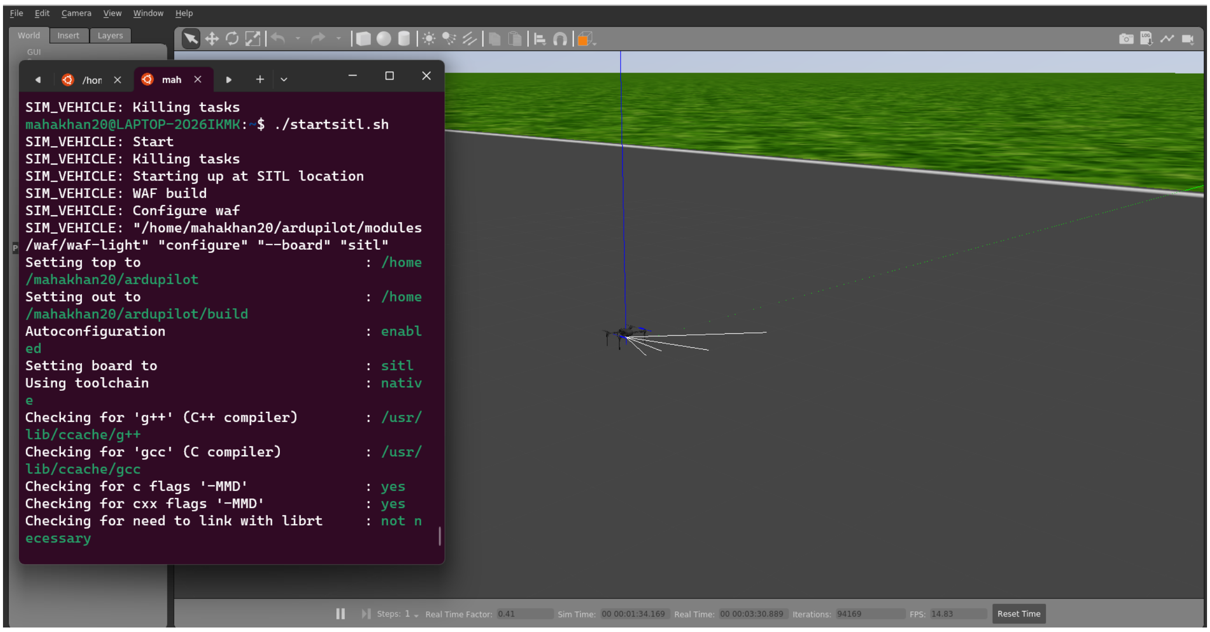
Mavlink is the protocol that will help us to communicate between Mavproxy (ground station control that allows us to control the drone through command line tools) and the vehicle (Ardupilot drone simulated in Gazebo). SITL or Software in the Loop allows us to send control signals for our simulated drone without the need for actual hardware.

1.a. Controlling the drone using Mavproxy Console:

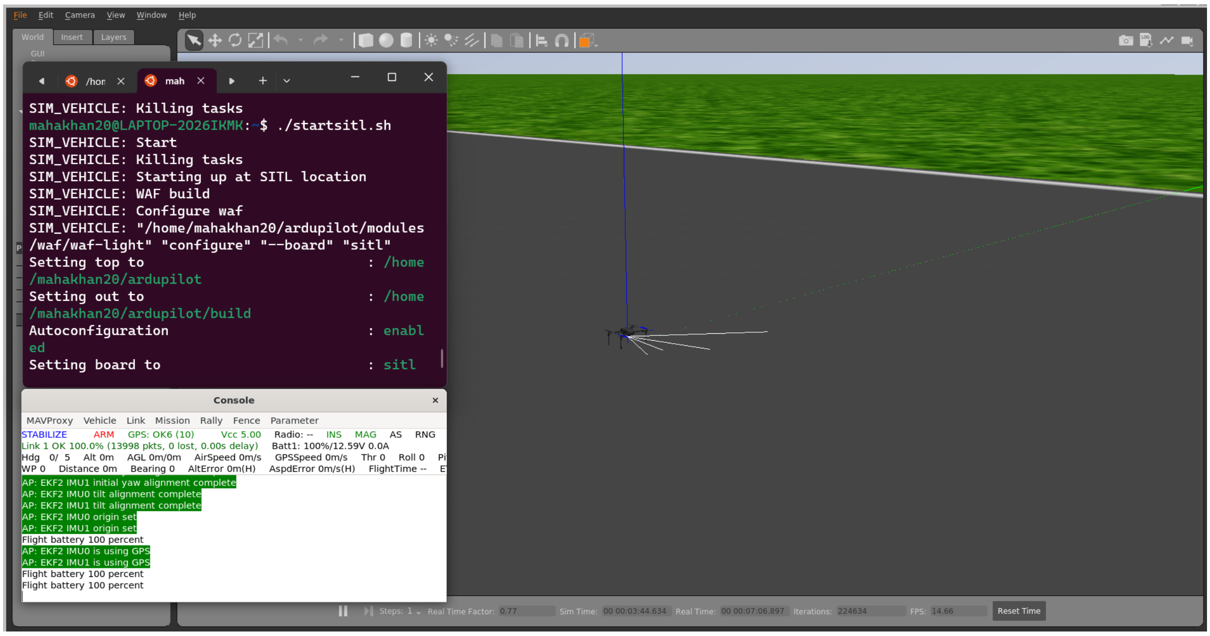
Now let’s start our mavproxy server and control our simulated drone in Gazebo.



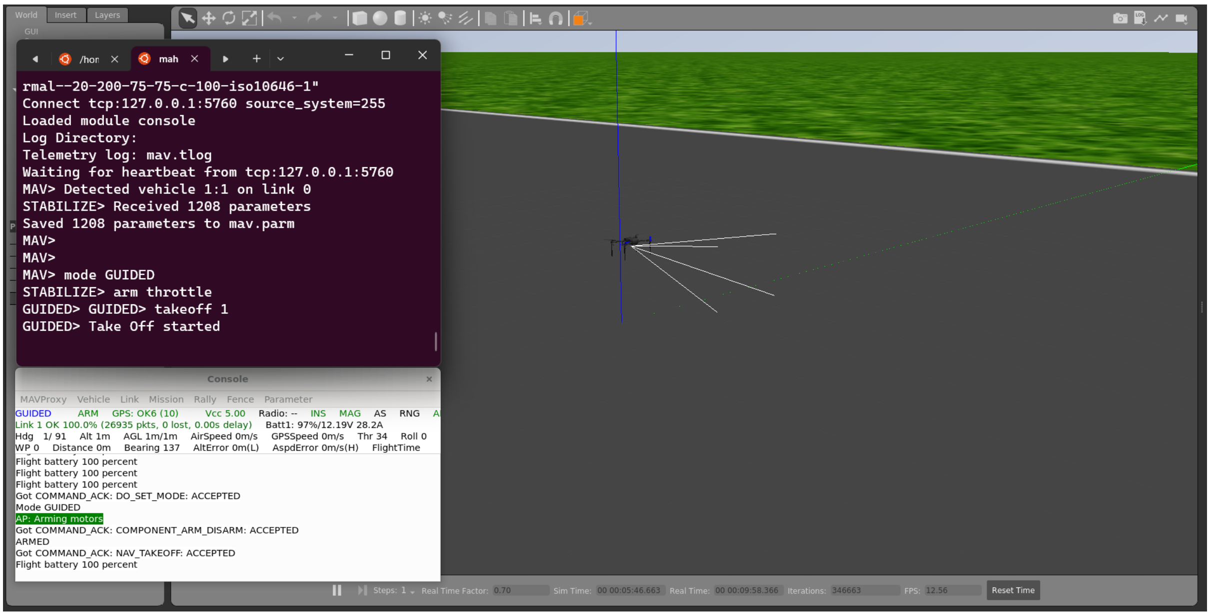
Launching the runway world which has our iris-arducopter.



I have prepared a script to launch the mavproxy console. Now we wait for “EKF2 IMU0 is using GPS” to appear on our console before we start controlling our drone.



Mavproxy has some useful command line tools. Set the flight mode to “GUIDED” on the terminal, arm the drone and takeoff to check if it is working. Note : after arming the drone you should send takeoff command quickly, otherwise the drone will automatically disarm.



Takeoff drone to 1 metre. You can use the ‘mode land’ command to land the drone on the same place.

1.b .Controlling the drone using Mavros:

Lets control our drone using a script. I have forked the iq package for high level control of our drone. We will be using some helper functions to set the mode of our drone, arm take off and send to a specific waypoint.

Here is a script in C++ :

```cpp
// Contains all necessary functions for drone set-up
#include <gnc_functions.hpp>

int main(int argc, char** argv)
{
    //initialize ros
    ros::init(argc, argv, "gnc_node");
    ros::NodeHandle gnc_node("~");
    init_publisher_subscriber(gnc_node);

    // Waitng for connection from FCU
    wait4connect();

    // Setting to mode guided
    set_mode("GUIDED");

    //create local reference frame: the pose of the drone from where it is first launched
    initialize_local_frame();

    // Take off attitude in meters
    takeoff(10);


    // Setting the horizontal speed in meters/sec
    set_speed(10);

    //specify control loop rate. We recommend a low frequency to not  overload the FCU with messages. Too many messages will cause the drone to be sluggish
    ros::Rate rate(2.0);

    while(ros::ok())
    {
        ros::spinOnce();
        rate.sleep();

        // Check for position tolerance
        if(check_waypoint_reached(.3) == 0)
        {
            // set destination to 0 mts in x direction, 30 mts in y direction and maintain 10 mts attitude, with desired yaw angle set to 0 degrees
            set_destination(0,30,10, 0);
        }

    }
}
```

The video to show the running of the script is available on my git-repo.

2. Spiral path implementation:

To implement a spiral path I have used the following equation:

x = r * sin(2*pi*n*t);
y = vf*t;
z = takeoff_alt + r * cos(2*pi*n*t);

Where:

r is the radius of the circle
n is no. of rotations per t
t is the parameter to control number of samples

C++ script to show the generated trajectory with guided mode:

```
// Spiral path in guided mode
// Contains all necessary functions for drone set-up
#include <gnc_functions.hpp>


int main(int argc, char** argv)
{
	//initialize ros
	ros::init(argc, argv, "gnc_node");
	ros::NodeHandle gnc_node("~");

	//initialize control publisher/subscribers
	init_publisher_subscriber(gnc_node);

	// wait for FCU connection
	wait4connect();

	//wait for used to switch to mode GUIDED
	wait4start();


	//create local reference frame
	initialize_local_frame();

	set_speed(7);

	// TakeOff
	int takeoff_alt = 10;
	takeoff(takeoff_alt);
```

```
	//specify some waypoints
	std::vector<gnc_api_waypoint> waypointList;
	gnc_api_waypoint nextWayPoint;


	int num_samples = 10;
	float pi = 3.14;
	int vf = 30; // forward velocity
	double r = 3.0; // radius of spiral
	int n  = 5; // No. of rotations per t
	float angular_velocity = 2*pi*n

	// start location of the spiral
	nextWayPoint.x = 0;
	nextWayPoint.y = r;
	nextWayPoint.z = takeoff_alt;
	nextWayPoint.psi = 0;
	waypointList.push_back(nextWayPoint); // push waypoints in the
list


	// Spiral parallel to y-axis
	for(float t = 0.0; t<=1.0; t = t + 1.0/num_samples)
	{
		nextWayPoint.x = r * sin(2*pi*n*t);
		nextWayPoint.y = vf*t;
		nextWayPoint.z = takeoff_alt + r * cos(2*pi*n*t);
		nextWayPoint.psi = 0;
		// nextWayPoint.psi = -360*t;
		waypointList.push_back(nextWayPoint);
	}



	// return to origin
	nextWayPoint.x = 0;
	nextWayPoint.y = 0;
	nextWayPoint.z = takeoff_alt;
	nextWayPoint.psi = 0;
	waypointList.push_back(nextWayPoint);


	//specify control loop rate. We recommend a low frequency to not
overload the FCU with messages. Too many messages will cause the drone
to be sluggish
```

```
      ros::Rate rate(4.0);
      int counter = 0;
      while(ros::ok())
      {
            ros::spinOnce();
            rate.sleep();
            if(check_waypoint_reached(.3) == 1)
            {
                  if (counter < waypointList.size()) //check if the
waypoint list is empty
                  {
set_destination(waypointList[counter].x,waypointList[counter].y,waypoint
List[counter].z, waypointList[counter].psi);
                        counter++;
                  }else{
                        //land after all waypoints are reached
                        land();
                  }
            }

      }
      return 0;
}
```

C++ script for spiral trajectory using auto mode:

This script shows the implementation of a spiral path using auto mode. auto mode accepts a list of waypoints in global coordinates (latitude, longitude and altitude) as ‘missions’ and automatically traces through each waypoint as a spline curve. Note that in guided mode the velocity reaches 0 before heading to the next waypoint, in auto mode the velocity slows down but does not tend to 0, resulting in a smoother trajectory. This script can be used to generate path parallel to x, y or z-axis by commenting out the respective blocks.

```
// Spiral path in auto mode

// Contains all necessary functions for drone set-up
#include <gnc_functions.hpp>

#define PI 3.14159265359

double current_lat = 0.0;
double current_lon = 0.0;

// Get current position of the drone in latitude and longitude
void globalPositionCallback(const sensor_msgs::NavSatFix::ConstPtr& msg) {
```

```
    current_lat = msg->latitude;
    current_lon = msg->longitude;
}

int main(int argc, char** argv) {
    // Initialize ros node
    ros::init(argc, argv, "gnc_node");
    ros::NodeHandle gnc_node("~");

    // Service clients for MAVROS mission commands
    ros::ServiceClient waypoint_push_client =
gnc_node.serviceClient<mavros_msgs::WaypointPush>("/mavros/mission/push");
    ros::ServiceClient waypoint_clear_client =
gnc_node.serviceClient<mavros_msgs::WaypointClear>("/mavros/mission/clear");

    //initialize control publisher/subscribers
    init_publisher_subscriber(gnc_node);

    // wait for FCU connection
    wait4connect();

    //wait for used to switch to mode GUIDED
    wait4start();

    double takeoff_alt = 20.0;
    takeoff(takeoff_alt);

    // Subscribe to global position updates
    ros::Subscriber global_position_sub =
gnc_node.subscribe<sensor_msgs::NavSatFix>("/mavros/global_position/global", 10,
globalPositionCallback);

    // Wait for the initial latitude and longitude to be received
    while (ros::ok() && (current_lat == 0.0 || current_lon == 0.0)) {
        ROS_INFO("Waiting for current latitude and longitude...");
        ros::spinOnce();
        ros::Duration(0.5).sleep();
    }

    // Clear existing waypoints
    mavros_msgs::WaypointClear waypoint_clear;
    waypoint_clear_client.call(waypoint_clear);

    // Parameters for the spiral
    double radius = 3.0;  // Radius of the spiral in meters
    double vertical_speed = 1.0;  // Vertical speed in meters/second
```

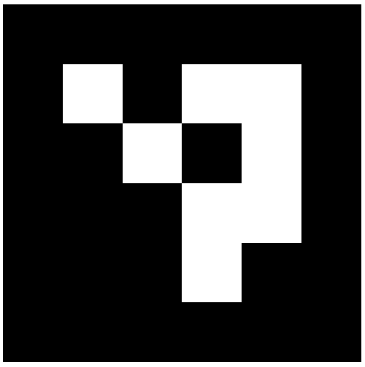
```
    double horizontal_speed = 3.0; // Horizontal speed in meters/second
    double angular_speed = 1.0;  // Angular speed of rotation in radians/second
    double max_height = 50.0;  // Maximum height of the spiral in meters

    // Specify some waypoints without using the iq_gnc library
    std::vector<mavros_msgs::Waypoint> waypoints;
    mavros_msgs::Waypoint waypoint;

    waypoint.frame = mavros_msgs::Waypoint::FRAME_GLOBAL_REL_ALT;
    waypoint.command = mavros_msgs::CommandCode::NAV_WAYPOINT;
    waypoint.autocontinue = true;

    double current_height = takeoff_alt;
    double current_time = 0.0;
    double current_position = 0.0;
    double max_position = 70.0;


    // // Spiral parallel to z axis
    // while (current_height < max_height)
    // {
    //     mavros_msgs::Waypoint waypoint;
    //     waypoint.frame = mavros_msgs::Waypoint::FRAME_GLOBAL_REL_ALT;
    //     waypoint.command = mavros_msgs::CommandCode::NAV_WAYPOINT;
    //     waypoint.is_current = false;
    //     waypoint.autocontinue = true;
    //     waypoint.x_lat = current_lat;
    //     waypoint.y_long = current_lon;
    //     waypoint.z_alt = current_height;
    //     waypoint.param1 = 0.0;
    //     waypoint.param2 = 0.0;
    //     waypoint.param3 = 0.0;
    //     waypoint.param4 = 0.0;

    //     // Generate setpoint coordinates for the spiral trajectory
    //     double x = radius * cos(angular_speed * current_time);
    //     double y = radius * sin(angular_speed * current_time);

    //     waypoint.x_lat += x / 111111.0;  // Convert x-coordinate to latitude
    //     waypoint.y_long += y / (111111.0 * cos(current_lat * PI / 180.0));
// Convert y-coordinate to longitude

    //     waypoints.push_back(waypoint);

    //     current_height += vertical_speed * 0.1;
    //     current_time += 0.1;  // Increase time
```

```cpp
    // }


    // Spiral parallel to x axis (same for y axis  just interchange the ode for
x axis with y axis)
    while (current_position < max_position)
    {
        mavros_msgs::Waypoint waypoint;
        waypoint.frame = mavros_msgs::Waypoint::FRAME_GLOBAL_REL_ALT;
        waypoint.command = mavros_msgs::CommandCode::NAV_WAYPOINT;
        waypoint.is_current = false;
        waypoint.autocontinue = true;
        waypoint.x_lat = current_lat;
        waypoint.y_long = current_lon;
        waypoint.z_alt = current_height;
        waypoint.param1 = 0.0;
        waypoint.param2 = 0.0;
        waypoint.param3 = 0.0;
        waypoint.param4 = 0.0;

        // Generate setpoint coordinates for the spiral trajectory
        double x = horizontal_speed*current_time;
        double y = radius * cos(angular_speed * current_time);
        double z = radius * sin(angular_speed * current_time);

        waypoint.x_lat += x / 111111.0;  // Convert x-coordinate to latitude
        waypoint.y_long += y / (111111.0 * cos(current_lat * PI / 180.0));  //
Convert y-coordinate to longitude
        waypoint.z_alt += z;

        waypoints.push_back(waypoint);

        current_position+= horizontal_speed * 0.1;
        current_time += 0.1;  // Increase time
    }

    // Push waypoints to the drone
    mavros_msgs::WaypointPush waypoint_push;
    waypoint_push.request.waypoints = waypoints;
    waypoint_push_client.call(waypoint_push);

    //wait for user to switch to mode AUTO
    wait4start_auto();
    return 0;
}
```

Check out the video for implementation of script on my git repo:

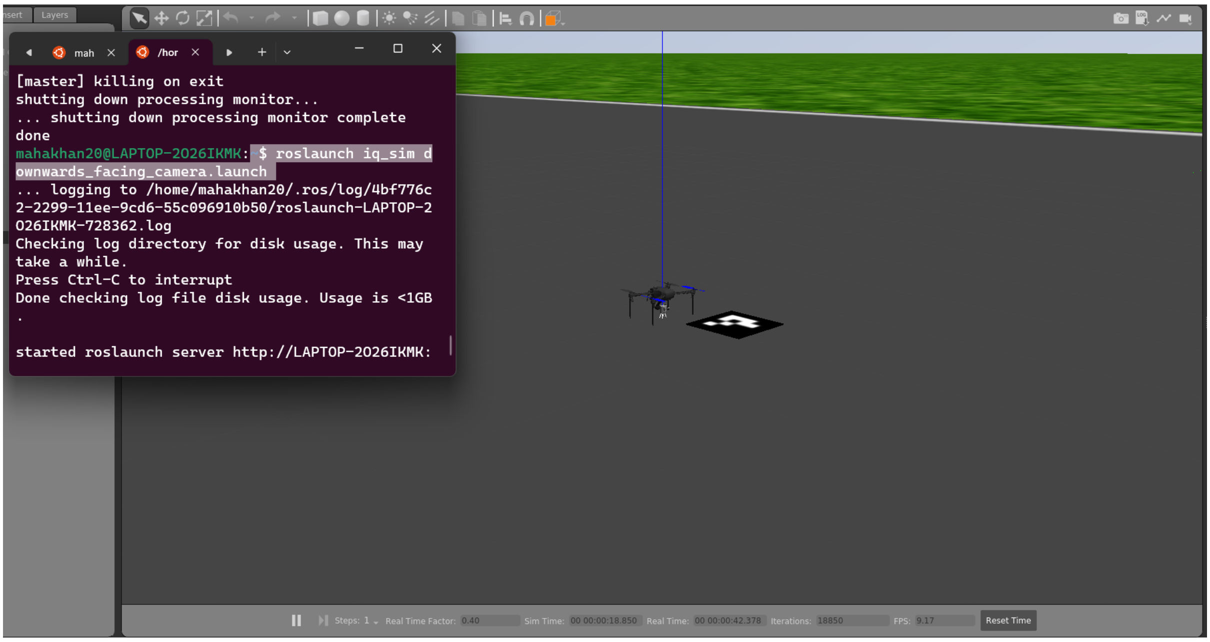
3. AruCo landing:

AruCo markers are used to estimate pose of the camera detecting them in an environment using only RGB images. For the task , I have used the opencv library for aruCo generation and detection.



Dictionary: 4x4 (50,)
Marker ID: 0
Marker size, mm: 500

I have prepared a script to integrate a downward facing camera with the iris-copter and a model of the marker and placed it in the Gazebo environment at a distance of (0, 1, 0) [x,y,z] meters from the launch location of the drone.



Here is a brief explanation of the C++ code to simulate precision landing of the drone:

The aruco library requires camera calibration parameters and BGR image data to perform image detection. The node subscribes to \webcam\cameraInfo topic for the specific camera calibrations, and the \webcam\image_raw topic for frame data.

The drone takes off at an attitude of 3 meters, simultaneously the camera looks for an aruco marker, if the marker is detected it hovers over the marker at an attitude of 0.1 meters while waiting for land command.

```
// Include required opencv and mat libraries for aruco detection
#include <gnc_functions.hpp>
#include <sensor_msgs/Image.h>
#include <sensor_msgs/CameraInfo.h>
#include <cv_bridge/cv_bridge.h>
#include <opencv2/opencv.hpp>
#include <opencv2/aruco.hpp>
#include <opencv2/core/mat.hpp>


//specify waypoint
std::vector<gnc_api_waypoint> waypointList;
gnc_api_waypoint nextWayPoint;

cv::Ptr<cv::aruco::Dictionary> dictionary =
cv::aruco::getPredefinedDictionary(cv::aruco::DICT_4X4_50);
cv::Mat cameraMatrix, distCoeffs;
bool markerDetected = false;
double markerSize = 0.5; // Size of the marker in meters
bool isCameraInfo = false;

void cameraInfoCallback(const sensor_msgs::CameraInfo::ConstPtr& msg) {
    ROS_INFO("CameraInfo");
    cameraMatrix = cv::Mat(3, 3, CV_64F,
const_cast<double*>(msg->K.data())).clone();
    distCoeffs = cv::Mat(1, 5, CV_64F,
const_cast<double*>(msg->D.data())).clone();
    isCameraInfo = true;
}

void imageCallback(const sensor_msgs::Image::ConstPtr& msg)
{

    if(!markerDetected)
    {   ROS_INFO("Detecting Marker");
        try
        {
            cv_bridge::CvImagePtr cvImage = cv_bridge::toCvCopy(msg,
sensor_msgs::image_encodings::BGR8);

            if (cvImage->image.empty())
            {
            ROS_ERROR("Empty image received");
```

```
            return;
            }

            cv::Mat frame = cvImage->image;
            cv::imshow("Camera Feed", frame);
            cv::waitKey(1);

            std::vector<int> markerIds;
            std::vector<std::vector<cv::Point2f>> markerCorners,
rejectedCandidates;
            cv::Ptr<cv::aruco::DetectorParameters> parameters =
cv::aruco::DetectorParameters::create();



            // Detect markers
            cv::aruco::detectMarkers(frame, dictionary, markerCorners,
markerIds, parameters, rejectedCandidates);

            // Draw marker outlines and IDs
            if (markerIds.size() > 0)
            {
                ROS_INFO("Marker Detected");
                cv::aruco::drawDetectedMarkers(frame, markerCorners,
markerIds);
                ROS_INFO("Marker Drawn");

                // Get the center of the first detected marker
                cv::Point2f markerCenter = (markerCorners[0][0] +
markerCorners[0][1] + markerCorners[0][2] + markerCorners[0][3]) * 0.25;
                ROS_INFO("Marker Center");


                if(isCameraInfo){
                markerDetected = true;
                // Calculate the 3D pose of the marker
                std::vector<cv::Vec3d> rvecs; std::vector<cv::Vec3d> tvecs;
                cv::aruco::estimatePoseSingleMarkers(markerCorners, markerSize,
cameraMatrix, distCoeffs, rvecs, tvecs);

                ROS_INFO("Pose estimated");

                // Get the translation vector of the marker center
                cv::Vec3d tvec = (tvecs[0] + tvecs[1] + tvecs[2] + tvecs[3]) *
0.25;
```

```
                cv::Vec3d rvec = rvecs[0];

                // gett the rotation vector of the marker
                float rx = -rvec[1];
                float ry = -rvec[0];
                float rz = -rvec[2];
                // receive yaw angle by converting the rotation vector
                nextWayPoint.psi =  rotationVector2eulerAngles(rx,ry,rz) *
180/3.14;


                // Save desired waypoint
                nextWayPoint.x = -tvec[1];
                nextWayPoint.y = -tvec[0];
                nextWayPoint.z = -tvec[2];


                ROS_INFO("Waypoint send");
                ROS_INFO("Aruco Marker pose wrt Camera : %f y: %f z: %f psi:
%f", nextWayPoint.x, nextWayPoint
                    .y, nextWayPoint.z, nextWayPoint.psi);

                // Converting the pose of the marker from the camera frame to
local reference frame
                set_destination_camera2local_frame(nextWayPoint.x,
nextWayPoint.y, nextWayPoint.z, nextWayPoint.psi);

                // if the marker is close enough on detection, directly land
                if (tvec[2]<0.2)
                {
                    land();
                }

            }
            }
        }
        catch (cv_bridge::Exception& e)
        {
            ROS_ERROR("cv_bridge exception: %s", e.what());
        }
    }
}

int main(int argc, char** argv) {
    // Initialise ros node
    ros::init(argc, argv, "gnc_node");
```

```cpp
    ros::NodeHandle gnc_node("~");

    init_publisher_subscriber(gnc_node);

    // Waiting for FCU to get a connection
    wait4connect();
    // Waiting for guided mode
    set_mode("GUIDED");

    //create local reference frame
    initialize_local_frame();

    // take off to 3 meters
    takeoff(3);
    set_speed(1);

    // recieve the  camera parameters information for calibration
    ros::Subscriber cameraInfoSub =
gnc_node.subscribe<sensor_msgs::CameraInfo>("/webcam/camera_info", 10,
cameraInfoCallback);
    // receive the frames as a bgr image
    ros::Subscriber imageSub =
gnc_node.subscribe<sensor_msgs::Image>("/webcam/image_raw", 1, imageCallback);
    ROS_INFO("Subscribing to webcam");

    ros::Rate rate(2.0);

    while(ros::ok())
    {
        ros::spinOnce();
        rate.sleep();
    }

    cv::destroyAllWindows();

    return 0;
}
```

The link to the video is available on my git repo.

— End of Assignment —

Feel free to suggest improvements and suggestions!
Contact me by email : mahakhan9455@gmail.com
LinkedIn : https://www.linkedin.com/in/maha-khan-9eve9/