# Implementation of Robot Operating System in Raspberry Pi 4 for Autonomous Landing Quadrotor on ArUco Marker

Atcha Daspan[1], Anukoon Nimsongprasert[1], Prathan Srichai[2], and Pijirawuch Wiengchanda[1,*]

[1] Marine Engineering Department, Academic Branch, Royal Thai Naval Academy, Samut Parkan, Thailand; Email: yeen.cup@gmail.com (A.D.), anukoonnim@gmail.com (A.N.)

[2] Department of Mechanical Engineering, Faculty of Engineering, Princess of Naradhiwas University, Naradhiwas, Thailand; Email: prathan.s@pnu.ac.th (P.S.)
*Correspondence: pijirawuch.w@navy.mi.th (P.W.)

***Abstract*—This article describes the novel design and implementation of the Robot Operating System (ROS) for the autonomous quadrotor landing on ArUco marker application. On a Raspberry Pi 4 companion computer, the ROS was set up using Ubuntu Mate 18.04. Then, to create communication between program nodes, ROS was put into practice together with autonomous landing. In the control approach, the Visual Inertial Odometry (VIO) technique, which uses vision-based localization, is employed to estimate the 3D posture. For computing the command control to direct movement quadrotor to landing on ArUco marker, the autolanding application is built. In experimental, 25 landing experiment trials were completed. The distance between the Drone's camera's center and the ArUco marker's center was calculated. In the results, the average distance accuracy during experimental validation was 11.12 cm, with a standard deviation of 3.67 cm.**

***Keywords*—quadrotor, raspberry pi, autonomous landing, robot operating system, ArUco marker**

## I. Introduction

Nowadays, quadrotors have been interested in developing to extent that several applications such as terrain modeling [1], surveillance of civilian protection facilities [2], aerial surveillance [3] for military purposes, and utilization of delivery operations for commercial purposes [4], are gradually expanding. However, a precise landing without the use of GPS (for example, inside) frequently necessitates a large amount of human involvement to achieve a successful landing. Realizing that most quadrotor crashes are caused by an unexpectedly forceful landing during the landing procedure, this becomes especially crucial for vital missions and when a quadrotor is carrying expensive equipment onboard [5]. Thus, reducing a number of mishaps is often accomplished by using landing aid devices. By increasing the accuracy of a quadrotor's posture estimation, the issue of accurate landing might be solved.

The ROS architecture is currently a fully functional platform for developing robots [6–8]. It is a group of programs, libraries, and protocols designed to make building sophisticated and reliable robotic systems easier. Another robotics system with a comparable platform can use its common foundation, for instance, robots are used today, example, in food packaging or in warehouse logistics, using autonomous robots to successfully solve a real life needed, such as, wheelchair [9–10]. In addition, applications with aerial robots (drones) have been put out in light of their MAVROS compatibility with Pixhawk including automated landing, surveillance, and coordination between aircraft and ground stations [11].

Pixhawk is an MAV open hardware platform. It has a wide user and developer community, as well as communication compatibility across varied hardware and software, which makes designing application control systems and installing sensor interfaces easier. Furthermore, these benefits make it simple to develop cross-enabled systems designed for robotic applications, since PX4 Firmware can simply communicate with Ubuntu of embedded systems installed on Pixhawk [12].

The aim of this project is to build, manufacture, and program a quadrotor capable of landing autonomously on the ArUco marker. A unique challenge will be to develop a ROS–enabled quadrotor capable of implementing all algorithms on a single board embedded computer.

## II. Hardware

### A. Quadrotor Desing

The quadrotor is a multirotor which is propelled by four sets of rotors. Based on S500 Drone Frame Kit with landing frame, 920 KV Brushless Motor with a 30 A Electronic Speed Controller (ESC) are mounted on each arm. Onto each motor, one black plastic 10×4.5 propeller is equipped. The main companion computer is a Raspberry Pi 4 device running Ubuntu Mate 18.04. The ROS Melodic version is installed on this operating

Manuscript received November 17, 2022; revised December 24, 2022; accepted February 7, 2023.

system. The companion computer performed a number of tasks, including sending control command to Pixhawk 2.4.8 flight controller and receiving data from the camera and laser rangefinder. It also processed pose estimates and the distance between the quadrotor and the marker. ArUco marker is detected using the ArUco ROS package. A 14.8 V 5200mAh 4S 45C LiPo battery pack provides DC electric power storage for all system components. The quadrotor with all its components is represented in Figs. 1 and 2. In its final configuration the quadrotor weights about 2.5 kgs and has a maximum diameter of 70 cms.

Figure 1. A quadrotor that the researcher has developed and fully assembled.

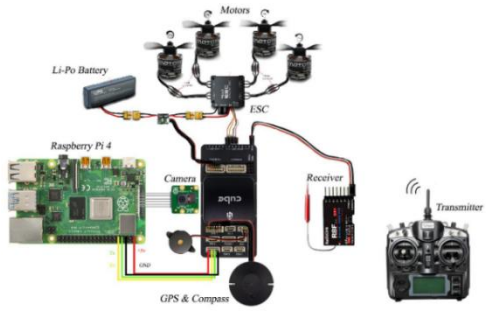


Figure 2. Shows all components of the quadrotor system used in this research.

### B. System Configuration

Fig. 3 shows the overall system configuration of this research. It consists of two major modules that are ArUco detect and pose estimation application module, and navigation module. The ArUco detect and pose estimation application module receive data from the camera, continuously. These data will next be subjected to an algorithm for posture estimation and image processing. The navigation module will then get the processed information.

### C. Electronics Component Interface

The schematic diagram of electronics circuit of the electronics hardware interface is represented in Fig. 4.

The optical camera that captures the ArUco marker is connected to serial port of raspberry Pi 4. The laser range finder that measured distance between quadrotor and ArUco marker is connected to GPIO16 pin. The flight control as Pixhawk is connected from telemetry 2 port to raspberry Pi 4 as GPIO14 and GPIO15 pin, shown in Fig. 3.

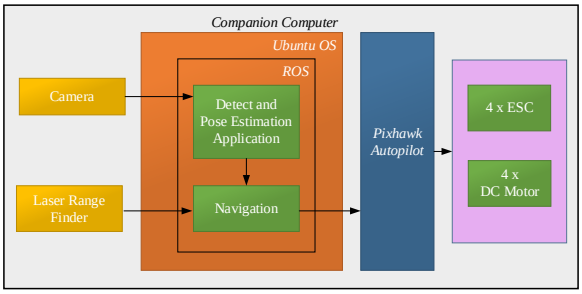


Figure 3. Block diagram of system configuration.

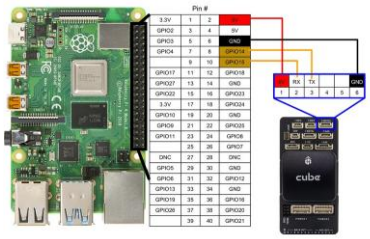

Figure 4. Schematic diagram of the connection between raspberry Pi and fight controller.

The 5VDC LIPO battery, which has a maximum current of 2.5 A, has been the system's power source.

## III. Software

### A. ROS Installation on Raspberry Pi

On Linux system simply communicate to the Raspberry Pi 4 from a terminal, with the Secure Shell (SSH) protocal by *ssh username@ip-address.*

*$ ssh pi@192.168.1.10*

The user will then be prompted by the system to enter their login and password. Pi is the default user name, while *raspberry* is the default password. The Raspberry Pi 4 needs network connectivity in order to install ROS.

Based on [13–14], the installation instructions that follow:

Step 1: Setting up the repositories. The following command must be used to set the system locale:

*$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'*

Step 2: By typing the following command on a terminal, the sources.*list* is configured to accept software from *packages.ros.org*:

*$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'*

Step 3: Before beginning the installation, the key should be added to Ubuntu to make sure the download is coming from a trusted server. Use the terminal to run the command below:

*$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654*

Step 4: Use the following command to update the list of packages:

*$ sudo apt upgrade*

Step 5: Use the command following to install ROS Melodic Desktop Install:

*$ sudo apt install ros-melodic-desktop*

Step 6: Use the following command on the terminal to initialize *RosDep*:

*$ sudo apt-get install python-rosdep*
*$ sudo rosdep init*
*$ rosdep update*

Step 7: The command; sets up the ROS environment.

*$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc*
*$ source ~/.bashrc*

Step 8: When all is done and the prerequisites are in place, the ROS may be installed with the following command:

*$ sudo apt-get install python-rosintall*

### *B. ROS Firmware Configulation*

A catkin workspace is a directory (folder) where researcher may build new catkin packages or edit ones that already exist. The catkin structure makes it easier to generate and install ROS packages. The following command may be used to create the catkin workspace:

*$ mkdir –p ~/catkin_ws/src*
*$ cd ~/catkin_ws/src*
*$ catkin_init_workspace*

Using the following command to creating the initial workspace:

*$ cd ~/catkin_ws/*
*$ catkin_make*

The catkin workspaces are created by the *catkin_make* command. Now, build and *devel* directories ought to be present in your ROS directory. The *setup.bash* file must then be sourced in order to add this workspace to your ROS environment:

*$ source ~/catkin_ws/devel/setup.bash*

### *C. Connecting Pixhawk to Raspberry Pi 4 with MAVROS Protocal*

Any programmable serial connection, including the Ethernet port, may be used by PX4 to connect to companion computers, Raspberry Pi 4. The *MAVLink* protocol is used to transmit messages across the connection.

### *D. Vision–based Localization*

In this study, the 3D pose estimation (location and orientation) and velocity of a moving quadrotor in relation to a local starting point are estimated using the Visual Inertial Odometry (VIO) technique, a kind of vision–based localization. Visual Odometry (VO) and inertial data from the IMU are used by VIO to determine the quadrotor's posture from camera pictures. Optical camera with global shutter is downward facing mounted. ArUco marker is detected and published their positions in ROS topics and as *TF* frames by *aruco_detect* module. The following installation guideline is installed on *catkin_ws* workspace by following *https://github.com/immersive–command system/Pose–Estimation–Aruco–Marker-Ros* [15].

### *E. Control Strategy for Autolanding on ArUco marker*

The controller algorithm supports autonomous mission execution by serving as an interface between the system's component parts. To interact with the remaining nodes, such as to read the pose estimated of the ArUco marker, it uses ROS topics. The *MAVROS* package [16] offers a communication driver and exposes a number of commands, system state variables, and interfaces as ROS subjects despite the fact that the autopilot employs the MAVLink communication protocol. For instance, the autopilot may be told to launch and fly to a certain location by publishing to the relevant MAVROS topic. Additionally, the local location of the quadrotor may be tracked and processed by utilizing a callback function. The second iteration of the transform ROS package, known as *tf2*, is used by the control logic block to maintain track of and easily transition between several coordinate frames.

Python programming language is used for autolanding node software development. As in Algorithm 1, once the program is running, the ArUco will be detected. The information obtained from ArUco detected is IDs, state detection and pose estimation which related between maker and quadrotor. The laser rangefinder is used for measure distance between floor and quadrotor. If altitude of quadrotor is less than or equal to 0 m, the system of quadrotor is descended. If altitude of quadrotor is less than 2 m, the landing angle ($\alpha$) of quadrotor is 5 °. If altitude of quadrotor is less than 15 m, the landing angle ($\alpha$) of quadrotor is 15 °.

In Fig. 5, The quadrotor's mobility is controlled by the autolanding node compute command. This node will continuously subscribe to the pose estimation topic from the ArUco detect ROS package and the *attitude_detected* topic from the sensor node. Additionally, this node will always publish to *cmd_vel* topic to fight controller as PikHawk.

In order to show the distance from the quadrotor to the ground during the present movement, the sensor node continually collects information data from the laser rangefinder sensor and translates that information data into centimeters. The check message is always subscribed to by the sensor node. The sensor node will run the software to determine the separation between the

quadrotor and the ground below when the check message is received. The autonomous landing node, which is receiving the most recent distance detected by the quadrotor, receives the distance detected as an *attitude_detected* message. The illustration is depicted as Fig. 6.

**Algorithm:** Autolanding on ArUco marker strategy

```
Start
while altitude < 15m and AutolandMode_selected do
    IDs, detected ← SearchArUcoMarker()
    if ¬detected then
        Loiter()
    else
        ID ← detected ID
        Get Pose Estimation()
        if altitude < 15m then
            landing_angle = 15°
        else if altitude < 2m then
            landing_angle = 5°
        else if altitude <= 0m then
            Descend()
        end if
        Autolanding()
    end if
end while
Descend and disarm Quadrotor
```

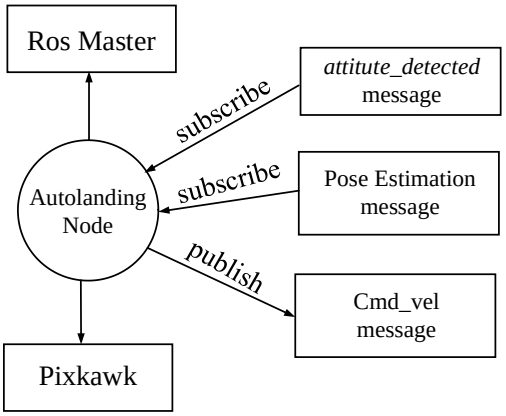


Figure 5. Autolanding node diagram.

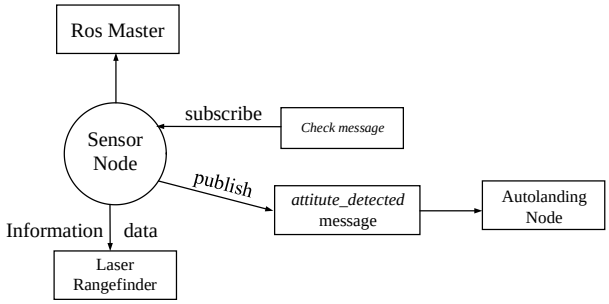


Figure 6. Sensor node diagram.

## IV. Experimental Settings

The ROS implementation in the autonomous quadrotor landing on the ArUco marking system is planned on the basis of the communication between several nodes. Numerous nodes are broadcasting with one another for an application in a ROS system. In ROS system, each node is communicating with other through manipulation by *roscore*, then, *roscore* must be execution, firstly.

### A. ArUco Marker

In this research, ArUco markers are created using an inside marker that is 7×7 cm in size and an exterior marking that is 30x30 cm in size. As seen in Fig. 7, the single black encoding square is replaced by the inner marker, which is positioned in the middle of the outer marker.

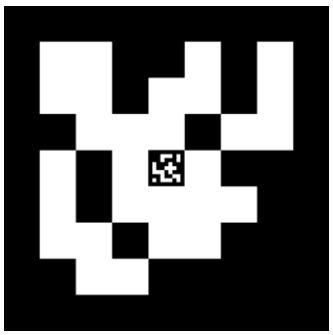

Figure 7. ArUco Marker used in this research.

The lowest and greatest distances at which ArUco markers may be identified are shown in Table I. These markers can be detected from a distance of 0.2 m to 15 m. The frequently used ArUco detect ROS package is used to find original ArUco markers. The non-modified *aruco_detect* ROS package, which is frequently used for authentic ArUco markers detection, carries out the detection.

### B. Ros Nodes Execution

Each node in the ROS system interacts with the others through manipulation by *roscore*, hence *roscore* must first be executed as shown in Fig.8. As soon as *roscore* launches, one of them will be launched.

- ROS Master
- ROS Parameter Sever
- *rosout* logging node.

To begin the connection between nodes, all of the aforementioned systems are necessary. It may be stopped using a keyboard interrupt, which will also turn off the ROS system. To enable access to the ROS commands and build into the workspace, the ROS must be source first before performing a *roscore*:

```
$ cd ~/catkin_ws
$ source /opt/ros/melodics/setup.bash
$ source devel/setup.bash
```

TABLE I. Describes Lowest and Greatest Distances

| Marker | Size (cm×cm) | Distances | |
|---|---|---|---|
| | | Lowest (m) | Greatest (m) |
| Inside ArUco | 7×7 | 0.2 | 0.8 |
| Exterior ArUco | 30×30 | 0.5 | 15 |

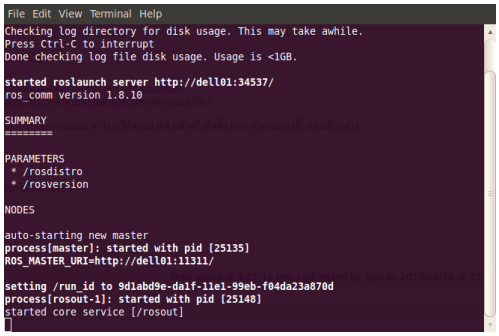


Figure 8. The roscore are executed in the terminal.

### C. Application Node Execution

Using the *rosrun* or *roslaunch* command, execute the applications node once the *roscore* has been launched. To publish and subscribe to a topic or message, broadcasting with another node requires several nodes. Therefore, in order to perform all of an application's functionality, all nodes must be running. As following command:

In terminal 1 to execute *MAVROS* package

*$ roslaunch mavros px4.launch fcu_url:="udp://:14540@192.168.1.20:14557"*

In terminal 2 to execute *aruco_detect* ROS package

*$ roslaunch asr_aruco_marker_recognition aruco_marker_recognition.launch*

In terminal 3 to execute *autolanding* node

*$ rosrun autolanding autolanding_aruco.py*

In terminal 4 to execute *sensor* node

*$ rosrun sensor distance_sensor.py*

### D. Testing Validation

For the purpose of assessing the viability system, numerous landing experiment trials were conducted. In the five directions of drone from maker, as well as forward (trail 1–5), backward (trail 6–10), right (trail 11–15), left(trail 16–20), and above (trail 21–25), in each direction, 5 landing trials were performed. 25 experiments were completed. The quadrotor lifted off to a height of 10m during the test, then awaited instructions to land on the ArUco marker (shown in Fig. 7). Following receipt of the order, the quadrotor carried out the aforementioned landing procedure. The distance between the Drone's camera's center and the ArUco marker's center was calculated once the landing was complete.

## V. Results and Discussion

### A. ROS Graph

ROS graph serves as a visualization graph. It helps to visualize how distinct nodes communicate with one another. Each node initializes a particular topic to interact with that node only. The autonomous landing technique has been implemented in this system using the nodes and subjects indicated in Fig. 9. These nodes have the ability to publish or subscribe to predefined topics, sending messages between them.

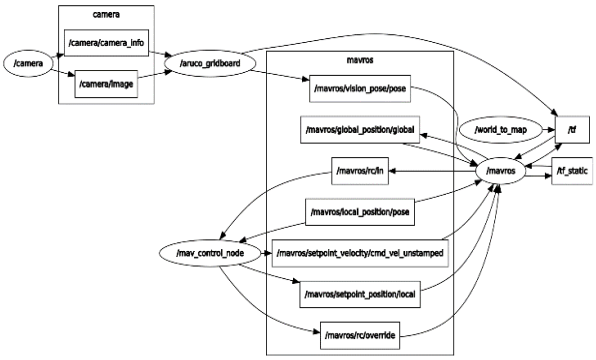


Figure 9. The peer-to-peer network of ROS processes that are processing data together (ROS Graph).

### B. Experimental Validation of System

To gauge the effectiveness of the created technology, several autolanding tests were conducted. 25 experiments in all were plan to the test. The quadrotor lifted up to a height of 6 m during the experiment and waited for the signal to autoland on the ArUco marking. The quadrotor executed the aforementioned landing process after getting the order. After landing, the distance between the quadrotor's frame's center and the ArUco marker's center was calculated. Fig. 10 shows the distances that were measured throughout all trials. The landing position accuracy increased substantially when Algorithm 1was adopted. In particular, experiments showed that the error ranged from only 6 to 19 cm, with average accuracy was 11.12 cm and the standard deviation was 3.67 cm.

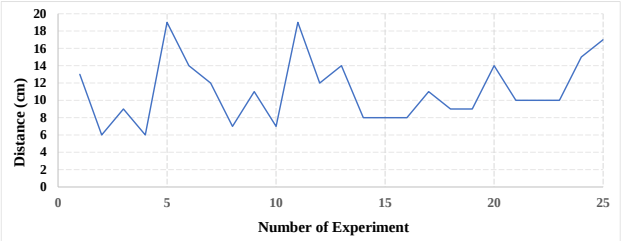


Figure 10. Distances between the quadrotor frame and the ArUco marker center (cm).

## VI. Conclusion and Future Work

Based on the ROS framework, this article proposes a novel method for autolanding quadrotor using visual sensory data. A new type of fiducial marker called embedded ArUco was developed to address the challenge of robust marker detection over a wide range of distances.

The applicability of the produced markers was confirmed using quadrotor landing tests. The ROS framework was used to implement and test a developed marker and landing algorithm. Our virtual tests revealed a 3.67 cm standard deviation and an average landing accuracy of 11.12 cm.

In order to improve the system, we intend to research sensor fusion using both IMU data and the vision system. On the other hand, a camera will be mounted on a gimbal to enable the quadrotor to track a moving subject.

## CONFLICT OF INTEREST

The authors declare no conflict of interest.

## AUTHOR CONTRIBUTIONS

Pijirawuch Wiengchanda performed the study, evaluated the data, and produced the report; the final draft was approved by all authors.

## ACKNOWLEDGMENT

This research has been funded by the Thailand Science Research and Innovation (TSRI).

## REFERENCES

[1] K. Kyoung–Ho, *et al*., "3D library platform construction using drone images and its application to Kangwha Dolmen," *Cartoon and Animation Studies*, Ser No.48, KOSCAS, pp. 199-215, 2017.
[2] K. Nagarjuna, and G. R. Suresh, "Design of effective landing mechanism for fully autonomous unmanned aerial vehicle," in *Proc. International Conference on Signal Processing, Communication and Networking (lCSCN)*, 2015, pp 1-6.
[3] M. A. Ma'sum et al., "Simulation of intelligent Unmanned Aerial Vehicle (UAV) for military surveillance," in *Proc. 2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 2013, pp. 161-166.
[4] Amazon.com: Amazon Prime Air, 2016. [Online]. Available: http://www.amazon.com/b?node=8037720011
[5] S. Emel'yanov, D. Makarov, A. I. Panov, and K. Yakovlev, "Multilayer cognitive architecture for UAV control," *Cognitive Systems Research*, vol. 39, pp. 58–72, 2016.
[6] About ROS. [Online]. Available: *http://www.ros.org/about-ros/* [Accessed: 26 Mar 2022]
[7] A. Koubaa, *Robot Operating System (ROS): The Complete Reference*. Springer, 2016.
[8] H. Lim, J. Park, D. Lee, and H. Kim, "Build your own quadrotor," *IEEE Robotics and Automation Magazine*, Sep.2012
[9] A. Elkodama, D. Saleem, S. Ayoub, C. Potrous, M. Sabri, and M. Badran, "Design, manufacture, and test a ROS operated smart obstacle avoidance wheelchair," *International Journal of Mechanical Engineering and Robotics Research*, vol. 9, no. 7, pp. 931-936, July 2020.
[10] A. B. Wahid, U. Siraj, M. Affan, H. Ahmed, F. Islam, U. Ansari, M. Naveed, and Y. Ayaz, "Development of modular framework for the semi-autonomous RISE wheelchair with multiple user interfaces using robot operating system (ROS)," *International Journal of Mechanical Engineering and Robotics Research*, vol. 7, no. 5, pp. 515-520, September 2018.
[11] F. Cocchioni, V. Pierfelice, A. Benini, A. Mancini, E. Frontoni, P. Zingaretti, G. Ippoliti, and S. Longhi, "Unmanned ground and aerial vehicles in extended range indoor and outdoor missions," in *Proc. International Conference on Unmanned Aircraft Systems*, 2014, pp. 374–382.
[12] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded opensource robotics framework for deeply embedded platforms," in *Proc. IEEE International Conference on Robotics and Automation*, 2015, pp. 6235–6240.
[13] E. Fernández, L. S. Crespo, A. Mahtani, and A. Martinez, *Learning ROS for Robotics Programming*, Packt Publishing Ltd, 2015.
[14] L. Joseph, J. Cacace, *Mastering ROS for Robotics Programming: Best Practices and Troubleshooting Solutions When Working with ROS*, Packt Publishing Ltd, 2021.
[15] Human-UAV interaction research group. [Online]. Available: https://github.com/immersive-command-system/Pose-Estimation-Aruco-Marker-Ros [Accessed on 20 August 2022]
[16] Mavros. [Online]. Available: https://github.com/mavlink/mavros/ [Accessed on 4 May 2022]