imav2019

Team Aerial Robotics
Indian Institute of Technology, Kanpur

Proposal for IMAV 2019

Outdoor Problem Statement

Date of Submission:

August 5, 2020

Contents

Supplementary Material

- GitBook Documentation : *https://aerial-robotics-iitk.gitbook.io/aerial-robotics-iitk/*
- Github : *https://github.com/AerialRobotics-IITK/*
- YouTube : *https://www.youtube.com/channel/UC0A50yxfhMAYRSOMTG87o6A*

1 About the Team

We are a team of undergraduate students at the Indian Institute of Technology Kanpur, working on the development of autonomous aerial vehicles, under the guidance of Dr. Indranil Saha and Dr. Mangal Kothari.

1.1 Mission and Vision

The mission of our team broadly encompasses the following aspects:

- Making valuable contributions to the global robotics and open source community.
- Representing our institute at various National/International Aerial Robotics competitions.
- Increasing awareness and promoting the utility of UAVs.
- Providing autonomous UAV based open-source solutions for real-life problems.

The vision of our team is to achieve recognition among the top student teams globally, to work towards bridging the gap between academia and the industry, and to participate at premier competitions.

1.2 Past Work

Our team has worked with state-of-the-art techniques in the domain of aerial vehicles, including the field of controls, localization and mapping.

1.2.1 Controls

- **PID Controller**
  We have an implemented an optimally tuned PID controller for the MAVs. It provides a basic lightweight control framework for tasks that require low to moderate level of control accuracy such as surveying.
- **Nonlinear Model Predictive Controller (NMPC)**
  We have implemented and rigourously tested a NMPC (1) for robust and accurate controls, even in the presence of environmental disturbances, for tasks that require high control accuracy such as precision landing or object delivery.

1.2.2 Localization

- **Visual Inertial Odometry (VIO)**
  To have accurate position feedback in indoor GPS-denied environments, we have successfully implemented a Visual Inertial Odometry framework. This system uses a monocular global shutter camera and a precise IMU for accurate predictions during high speed motions and long missions. This system helps the MAV to navigate in an indoor environment with centimetre level accuracy locally and with a slight drift in the global frame. Our current architecture exploits the open source algorithm, ROVIO (2).

- **GPS**
  The Real Time Kinematic GPS (RTK-GPS) is used to get accurate position feedback in outdoor environments. It uses measurements of the phase of the signal's carrier wave and provides real time correction with reference to a base station module, achieving up to centimetre-level accuracy.

- **Motion Capture**
  We use the *VICON* (3) system to benchmark/test our algorithms and systems. The *VICON* system provides millimetre position accuracy with much higher rates (1 kHz).

1.2.3 Simulation

To ensure the proper functioning of newly implemented algorithms and to minimize the risk of damage suffered by the MAVs during testing, our algorithms are first tested on a simulation platform which we created by modifying the RotorS (4) simulator. RotorS is a Gazebo (5) based MAV simulator with models of some of the most used commercial MAVs, sensors and modules. It also has built-in plugins and directly supports ROS (6) communication.

1.2.4 Vision-based localization and landing

We have also implemented a ArUco marker based localization module and a precision landing module based on it. The system uses *aruco_ros* (7) libraries and provides precise pose of a specific marker in the camera frame which is later transformed into global coordinates for high-level control. This method was further extended to detect coloured quadrilateral objects and perform the landing task.

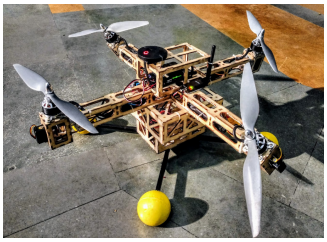
1.3 Past Competitions

- **3rd Position, Warehouse Inventory Check :: 6th Inter IIT Tech Meet, IIT Madras**
  The primary objective was to develop a MAV which could fly indoors (with any on-board or off-board computation and tracking system) and localize itself using lines on the warehouse floor. The aerial robot had to perform a complete warehouse inventory check mission, starting with autonomous take-off, then line following, scanning the warehouse goods, and finally autonomous landing. The complete problem statement was divided into three sub-modules and integrated using ROS:

  - **Vehicle Design and Controls:** Our pipeline used a tilt-quadrotor (Figure 1a) as the MAV (8) and a high-level PID controller based on the feedback provided from the floor lines.
  - **Localization:** The tilt-quadrotor was mounted with a monocular nadir camera for line detection. The guiding line also had cross markers over it at certain places where the vehicle was required to hover so that it could scan the warehouse goods.
  - **Height Estimation and Inventory Management:** The system used ultrasonic sensors and an IMU for height estimation as shown in (9). For the warehouse inventory, the information was extracted from QR codes which was then arranged sequentially.

- **2nd Position, PlutoX Hackathon :: 7th Inter IIT Tech Meet, IIT Bombay**
  This was an on-the-spot 24 hour long hackathon on the platform PlutoX (Figure 1b), developed by Drona Aviation. The problem statement was to program the PlutoX to pass through parallel walls without any collisions autonomously. The event was followed by presentation rounds.

Our team managed to successfully complete the problem statement. Our solution involved mounting ultrasonic sensors on the PlutoX and implementing a high-level PID controller using filtered and denoised feedback from the sensors.



(a) Tilt-Quad, the platform used in the Warehouse Inventory Check challenge

(b) PlutoX, the platform provided for the hackathon

Figure 1: MAVs used in past competitions

1.4 Motivation for participating in the Outdoor Challenge

The outdoor problem statement correlates with the work that our team wishes to pursue. Apart from this, package delivery is one of the most sought after MAV applications. In the context of our vision, we want to accomplish this task and subsequently make our work available to the community for further application.

2 System Description

(a) MAV-1: A system with heavy payload carrying capabilities, equipped with Intel NUC and VIO sensor

(b) MAV-2: A system capable of high speed surveying equipped with RTK-GPS and a high resolution camera

Figure 2: Protoype Systems

2.1 State Estimation

Accurate and robust state estimation is the most crucial element in the execution of fast and precise trajectories, which is essential for solving the outdoor challenge. This section briefly introduces the methods that we have implemented for state estimation.

- **Translation Estimate:** The translational degrees of freedom is estimated with the position data from RTK-GPS, *VICON*, VIO, and LiDaRs. For outdoor missions, the MAV uses GPS for horizontal position estimation and a LiDaR for height estimation. For indoor

missions, the MAV uses VIO for complete odometry estimation. For robust and aggressive indoor testing, the MAV uses *VICON* for complete state estimation. For velocity estimation, the MAV uses fusion of attitude (provided from the on-board IMU/ *VICON*) and position data.

- **Rotational Estimate:** For rotational degrees of freedom, the MAV uses on-board IMUs with sensor fusion using an Extended Kalman Filter (EKF) algorithm provided by the PX4 autopilot (10).

2.1.1 Kalman Filter

The noisy position estimates are filtered using a Linear Kalman Filter algorithm (1). Newtonian mechanics have been used to formulate the system model, with the acceleration varying linearly between two consecutive iterations. This acceleration is measured by the PixHawk's on-board IMU, and used in the prediction step. Then the predicted data is fused with the incoming position data, in the measurement step. The filtered position estimate is more accurate as compared to the unfiltered position estimate, because data from both the sensors is being used to calculate the belief of the position of the robot. Figure 3 depicts the noisy position data (in this case the height), compared to the filtered data.

Algorithm 1 Linear Kalman Filter

Input : (X_{k-1},P_{k-1}, U_k, Z_k, F_k, B_k, Q_k, H_k, R_k): X_{k-1} - initial belief vector; P_{k-1} - initial covariance matrix; U_k - control vector; Z_k - measurement vector; F_k - state transition model; B_k - control input model; Q_k - process noise covariance; H_k - observation model; R_k - observation noise covariance

Output: (X_k, P_k): X_k - final belief vector; P_k - final covariance matrix

$P_k \longleftarrow$ Identity Matrix; $X_k \longleftarrow$ Identity Vector

repeat

// **Prediction step**

$\hat{X}_k \leftarrow F_k * X_{k-1} + B_k * U_k$

$\hat{P}_k \leftarrow F_k * P_{k-1} * F_k^T + Q_k$

// **Update Step**

$K \leftarrow \hat{P}_k * H_k^T * (H_k * \hat{P}_k * H_k^T + R_k)^{-1}$

$X_k \leftarrow \hat{X}_k + K * (Z_k - H_k * \hat{X}_k)$

$P_k \leftarrow \hat{P}_k - K * H_k * \hat{P}_k$

Publish X_k **and** P_k

until *end of input*;

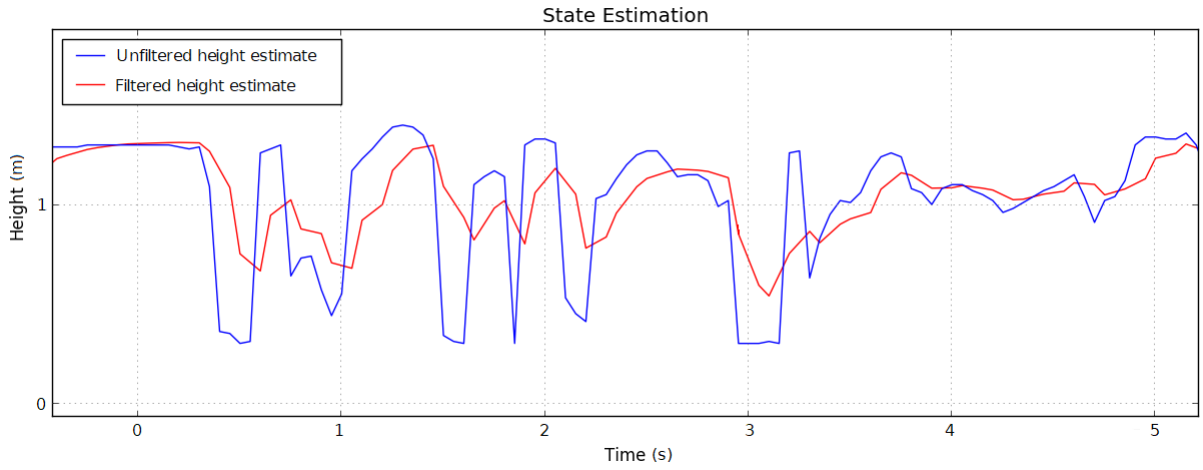


Figure 3: Height vs Time plot for unfiltered (blue) and filtered (red) height estimate

2.2 Control System

The system uses a two-level cascade controller as shown in Figure 4:

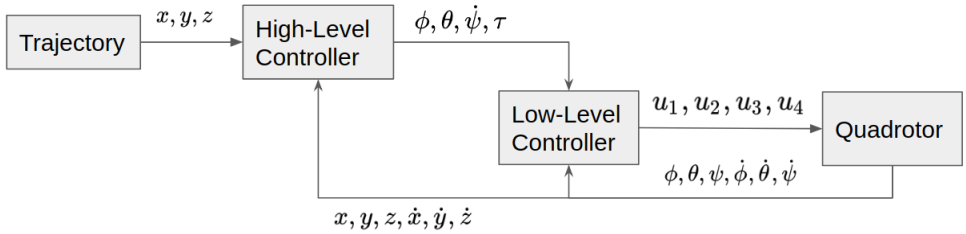


Figure 4: Control architecture of the MAV. The high-level NMPC controller takes a trajectory as input, and outputs roll (ϕ), pitch(θ), yaw-rate($\dot{\psi}$) and thrust(τ) commands. A low-level controller runs on attitude feedback and commands the motor speed.

2.2.1 Low-level PID Controller:

The low-level controller runs on an STM32F processor that provides on-board attitude estimation at a high frequency (200 Hz). It also communicates with sensors and the on-board computer through a MAVLink (11) architecture provided by PX4 (10).

2.2.2 High-level NMPC controller:

For an optimal controller, system identification is the primary requirement. After getting an estimate of the system dynamics, the controller solves a constraint optimization problem and produces the necessary control inputs. Our system uses the ACADO Toolkit (12) to solve the NMPC. We first define the following state vector:

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{p}^T & \boldsymbol{v}^T & \phi & \theta & \psi \end{pmatrix}^T, \tag{1}$$

where p is the position vector, v is the velocity vector and ϕ, θ and ψ represent the roll, pitch and yaw respectively.
and the control input vector:

$$\boldsymbol{u} = \begin{pmatrix} \phi_{cmd} & \theta_{cmd} & \dot{\psi}_{cmd} & \tau_{cmd} \end{pmatrix}^T \tag{2}$$

Now, we can define the Optimal Control Problem (the optimization problem that is solved by the controller) as follows:

$$\begin{aligned} \min_{\boldsymbol{U},\boldsymbol{X}} \int_{t=0}^{T} \Big(\|\boldsymbol{x}(t) - \boldsymbol{x}_{ref}(t)\|^2_{\boldsymbol{Q}_x} + \|\boldsymbol{u}(t) - \boldsymbol{u}_{ref}(t)\|^2_{\boldsymbol{R}_u} \, dt \Big) + \|\boldsymbol{x}(T) - \boldsymbol{x}_{ref}(T))\|^2_{\boldsymbol{P}} \\ \text{subject to} \quad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}); \\ \boldsymbol{u}(t) \in \mathbb{U} \\ \boldsymbol{x}(0) = \boldsymbol{x}\,(t_0)\,. \end{aligned} \tag{3}$$

where $\boldsymbol{f}(x, u)$ is the nonlinear system dynamics obtained via system identification,
$Q_x \succeq 0$, $R_u \succeq 0$ and $P \succeq 0$ are the respective penalties on the errors for the state, control input and final state,
x_{ref}, and u_{ref} are the target state vector and target control input at time t,
U is the control input constraint given by $U \in \{-\bar{u}, \bar{u}\}$, and
T is the prediction horizon for the given cost function.

The controller is implemented in a receding horizon fashion as explained in (1), where the aforementioned optimization problem needs to be solved at every time step and only the first control input is actually applied to the system.

3 Approach and Module-wise Division of Tasks

The complete outdoor problem statement would be accomplished using three MAVs. Initially after autonomous takeoff, two of the MAVs would execute a predefined trajectory in search of the lost objects and mailboxes. The third MAV would simultaneously fly at a higher altitude to capture images for mapping and detection of the house. After capturing the required images, if all the objects/mailboxes have not been found, the third MAV will join the other two MAVs in exploring the mission area. After exploration, the detected object poses would be published on a ROS topic and each of the MAVs will go to a mailbox and perform the delivery task, then land autonomously in the landing zone. The fragile object delivery module is yet to be integrated.

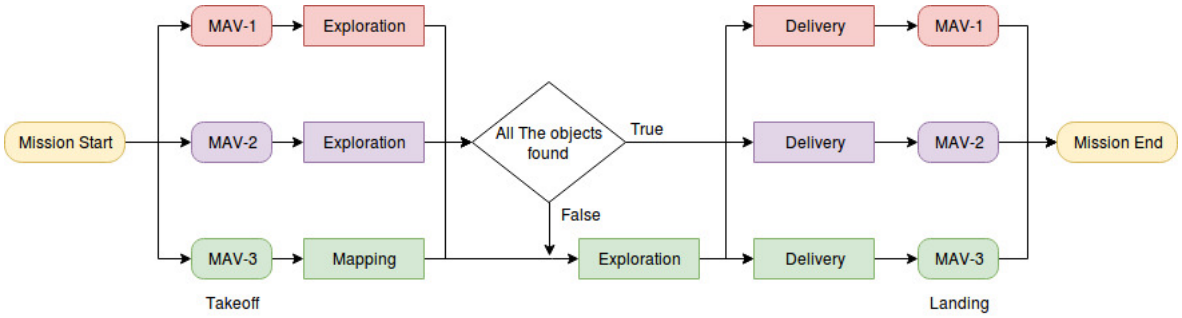


Figure 5: The Mission Approach

The complete problem statement is divided into discrete submodules, which are then integrated into a complete end-to-end solution with a finite state machine which is written using the MSM library (one of the Boost C++ libraries). The finite state machine serves as the bridge between all the different modules, by providing transition functions to switch between modules, action functions to execute the corresponding modules within each state and guard functions to prevent transition between modules in case of any errors. At any given time during the execution of the mission, the MAV can be in any one of the following states:

Rest: The MAV is in a state of rest - disarmed and on the ground.

Hover: The MAV is hovering over a fixed point at a fixed height.

Exploring: The MAV is exploring the mission area and is collecting the position data of all the objects.

ReachMailbox: The MAV is enroute to a mailbox.

Drop: The MAV hovers at a fixed point just above the ground. In this state the servo is triggered to release the package.

ReachLZ: The MAV is enroute to the Landing Zone (LZ).

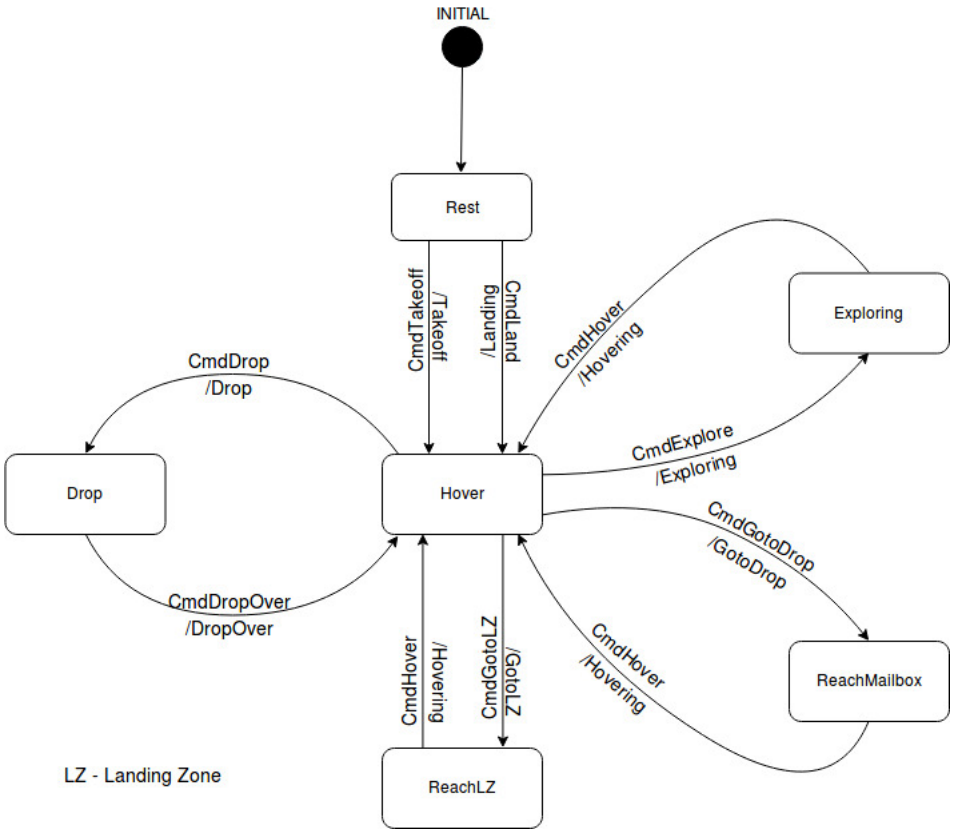


Figure 6: State Machine Diagram

A brief description of all the submodules is as follows:

3.1 Autonomous Takeoff and Landing

This module handles the autonomous takeoff procedure, once the MAVs are loaded with the packages. At the start of the mission, high-level commands are sent to the controller to achieve takeoff.

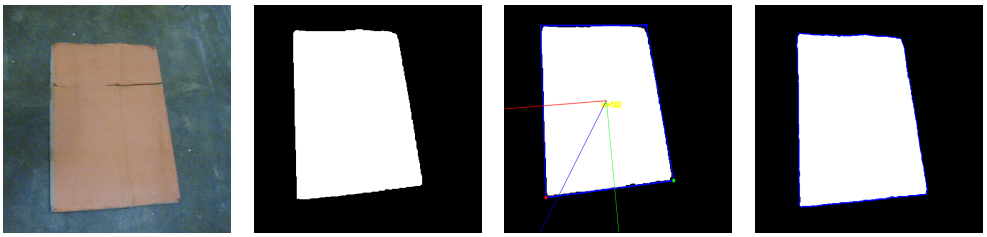
At the time of landing, the module will run a detection algorithm to detect and track the 'H' sign, estimating its pose and using it to send the required high-level commands to move the MAV to that position with an exponential decay in altitude.

3.2 Exploration

This module includes the commands and trajectories that the MAVs would be following in order to collect data for mapping and object detection. Once this state is activated after the takeoff, the MAVs will follow a pre-optimized trajectory to survey the arena, and publish the location of the found objects during the same. The trajectories are deployed to the systems via QGroundControl (13).
The trajectory is optimized to capture maximum area in minimum distance and time. The field of view of the camera is taken into account for optimization of the height and distance between adjacent passes. The complete geometric area of the arena is divided into polynomial segments (14) for each MAV in a collision free manner.

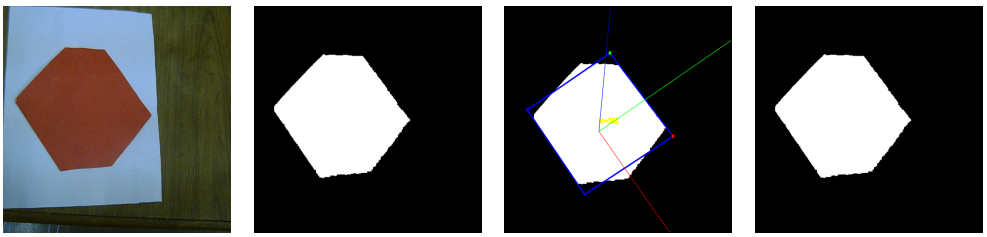
3.3 Object Detection

The purview of this module extends to detection of the objects (i.e. mailboxes and lost packages) seen by the MAVs during the execution of their respective mission trajectories. This module takes live image streams from the camera mounted on the platforms as its input, and outputs custom ROS messages that contain the information about the type, colour, size, position (with respect to various reference frames) and the GPS coordinates of the detected objects.
Our first attempt at object detection involved the use of the ArUco library with some modifications - a new marker definition was added that would only detect quadrilateral segments from the thresholded image. However, this approach did not prove to be accurate as it was unable to filter out noise elements or detect rectangular segments properly. Multiple objects of different colours could also not be detected simultaneously due to the requirement of a separate thresholded image for each colour. Hence we applied a floodfill-based segmentation algorithm(15) to overcome these issues, which is detailed below:



(a) Raw Image (b) Thresholded (c) ArUco Based (d) Floodfill based

Figure 7: Detection of a rectangular object, positive from both algorithms.



(a) Raw Image (b) Thresholded (c) ArUco Based (d) Floodfill based

Figure 8: False Positive detection from ArUco-based algorithm since it detected a non rectangular object, whereas the floodfill algorithm did not detect the object.

3.3.1 Segmentation

Since the image obtained from the fish-eye camera is distorted, it is passed through an undistort function that takes the image and the camera model as the input, and outputs an undistorted image. A floodfill-based segmentation Algorithm 3 is then applied on the undistorted image. The algorithm uses an RGB grid to consecutively check whether an image pixel may be part of a potential object. Once an object pixel is detected, a queue-based floodfill Algorithm 2 is initiated to find an object segment around it. The queue used in Algorithm 2 contains the coordinates of the object segment's pixels in the image frame, along with two pointers $\mathtt{q_{start}}$ and $\mathtt{q_{end}}$. At the end of the floodfill algorithm, the queue contains the positions of all the pixels in the segment which is then further processed.

Algorithm 2 Floodfill algorithm

Input : $(p, rgb\_grid, object\_class)$: p - starting pixel position; rgb_grid - RGB lookup grid; $object\_class$ - searched segment label

Output: $(u, v,$ queue, valid$)$: (u, v) - bounding box; queue - pixel positions; valid - segment validity

$s_{id} \leftarrow s_{id} + 1$; $q_{end} \leftarrow q_{start} \leftarrow 0$ // s_{id} stores segment id

$u_{max} \leftarrow u_{min} \leftarrow 0$; $v_{max} \leftarrow v_{min} \leftarrow 0$

pixelLabel$[p] \leftarrow$ s$_{id}$ // label starting pixel

queue$[q_{end}++] \leftarrow$ p

while $q_{end} < q_{start}$ **do** // until end of queue is reached

  $q \leftarrow$ queue$[q_{start}++]$ // get pixel from queue

  foreach $offset \in \{+w, -w, +1, -1\}$ **do** // iterate over pixel neighbours

    $r \leftarrow q + offset$

    if pixelLabel$[r] = unknown$ **then** // get pixel label

      pixelLabel$[r] \leftarrow$ rgb_grid[Image(r)]

    if pixelLabel$[r] =$ object_class **then** // if pixel is of same object

      queue$[q_{end}++] \leftarrow r$ // add current pixel to queue

      pixelLabel$[r] \leftarrow s_{id}$ // mark pixel as processed

      $u_{min} \leftarrow \min(u_{min}, r_u)$; $u_{max} \leftarrow \max(u_{max}, r_u)$ // update bounding box

      $v_{min} \leftarrow \min(v_{min}, r_v)$; $v_{max} \leftarrow \min(v_{max}, r_v)$

valid $\leftarrow true$; $s \leftarrow q_{end}$

if $s < min\_size$ **then** // check queue size

  valid $\leftarrow false$

Algorithm 3 Segmentation algorithm

Input : (p_0, *rgb_grid*, *mode*, Image): p_0 - search start position; *rgb_grid* - RGB lookup grid; Image- image to be processed

Output: *segments*: *segments* - a vector of data structures containing information of all valid segments

```
s_id ⟵ 0; p ⟵ p_0                                          // initialize segment id
repeat
    if pixelLabel[p] = unknown then                        // get current pixel label
        if rgb_grid[Image(p)] = object_class then
            pixelLabel[p] = object_class
    if pixelLabel[p] = object_class then                   // if pixel is of object type
        c ← floodfill(p, object_class)                     // execute floodfill
        if valid(c) then                                   // if pixel queue is valid
            compute more segment data
            valid ← rectangleCheck(c)
            // check if segment is rectangle, see Section 3.3.2
            if valid(c) then                               // if segment passed all checks
                segments = segments ∪ c
    p ← (p + 1) mod sizeof(Image)                          // move to next pixel
until p ≠ p_0
```

3.3.2 Pixel Queue Processing

Once the floodfill algorithm finishes an iteration, the output queue is checked for a minimal size in terms of the number of pixels in the segment. If the queue does not pass this check it is not processed further, and the segmentation algorithm resumes searching the image for more object segments. If the queue exceeds the minimum size, it is further processed.

Using the pixel queue, the center of the segment is found. Then the corners are found by calculating the positions of the four most farthest points from the centre. The `rectangleCheck` algorithm is used to check whether the segment could be a rectangle. It consists of the following steps:

- The first check involves calculating the area of the segment using the coordinates of the corners, and then comparing it with the queue size (the two values should almost be equal).
- The second check involves the use of Principal Component Analysis (PCA) to determine the length and width of the segment according to the pixel spread. PCA (16) is a statistical procedure that converts a set of observations of possibly correlated variables - in this case the pixel queue - into a set of values of linearly uncorrelated variables - in this case the length and the width. The area is calculated using this length and width, and then compared with the queue size.
- The final check involves comparing the lengths of both the diagonals of the segment (for a rectangle, these lengths should be equal).

Pixel queues that are valid after these checks are added to a vector of valid segments, which is the final output of the segmentation algorithm.

3.3.3 Pose Estimation

Once the segmentation algorithm produces a vector containing all the data of valid segments, the height data from the LiDaR sensor and the camera model is used to calculate matrices that transform coordinates from the image frame into the camera frame, then into the MAV frame and finally into the global frame. After the global coordinates are obtained for the detected segments, their sizes are compared with the actual size of the objects - this eliminates any remaining false positives.

3.4 Object Gripping and Delivery

After reaching the mailbox location, the MAV descends to a specific height, correcting its position using the relative pose of the detected object. Once the target height is reached, the control message for the actuator is published. The low power actuator connected to the system is triggered by this control message and releases the package. This module also includes the design aspect of the gripper as well as the picking mechanism.

3.5 DNN-Based Crashed MAV and House detection

Our system is equipped with an NVIDIA Jetson TX2[1]. The pre-trained models of the crashed MAV and house are deployed on the same.

- **Crashed MAV:** The crashed MAV is detected using a Convolutional Neural Network (CNN) trained on the DarkNet YOLOv3 framework. The network was trained using transfer learning on a dataset of approx. 3000 images of quadrotors. The predictions of the network are integrated into the ROS framework using the DarkNet ROS package (17). Once the bounding boxes are obtained in the image frame, it is converted into the global frame via a series of transformations and stored in the form of GPS coordinates.

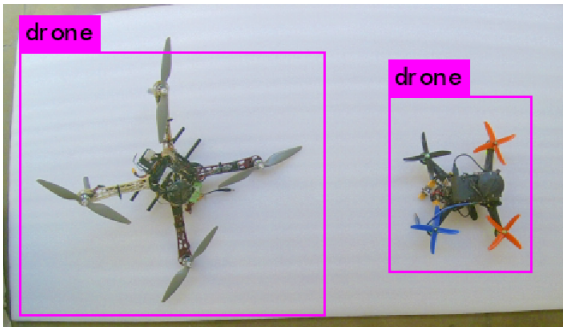


Figure 9: Images of detected MAV. The left bounding box is marked with a confidence of 97% while the right bounding box is marked with a confidence of 83%

- **House detection:** The house would be detected using a CNN that would be trained to segment roofs. The model's predictions in the image would then be accordingly converted into global coordinates and geo-stamped on the generated 2D map.

[1]Provided to the team under the NVIDIA GPU grant program

3.6 Mapping

Following are the approaches the MAVs would follow for the mapping task:

- **Open Drone Map (ODM):** The ODM (18) library would be used to generate the map of the mission area using the geo-stamped RGB images. ODM initially converts the images into a point cloud. The dots in the point cloud are then connected to create a 3D model with a continuous surface. For generating the map, a projected image of the top view from the 3D model is used (i.e. the orthophoto).

Figure 10: 2D RGB map using ODM, this map was created in 45 seconds using 8 individual images

- **OpenCV Mosaic:** In this method, the ORB (Oriented FAST and Rotated BRIEF) (19) features are used to extract keypoints from images. Using the similar keypoints present in consecutive images, a transformation matrix is calculated. The current image is then transformed into the previous image frame by superimposing the similarly extracted keypoints. As a result, an image is created by stitching the consecutive images. This process is iterated over until the images continue to stream. Thus a real-time 2D RGB map is generated.

4 Architecture

4.1 Hardware

All the MAVs have a custom carbon fibre symmetric 'X' quadrotor configuration frame with the following components (and other basic flight hardware):

- **Computational Units:** The MAVs will be mounted with Odroid XU4, Intel NUC and NVIDIA Jetson TX2 according to the increasing level of computational requirements for tasks such as mapping and DNN based object detection.
- **Gripping:** For gripping tasks, each MAV will be equipped with servo-based low-powered grippers having heavy load ($\approx 1kg$) carrying capabilities.
- **Vision System:** The MAVs will be mounted with high resolution (2.4K) Mobius Maxi Action Cameras for object detection and mapping.
- **Flight Controller:** Each system includes a Pixhawk 2.1 Cube Flight Controller.

- **Sensors:** The other sensors mounted on the MAVs include the Here+ GPS with RTK capabilities and the Benewake TFmini 1-D LiDaR for height estimation.

Please refer to Figure 11 for a complete hardware overview.

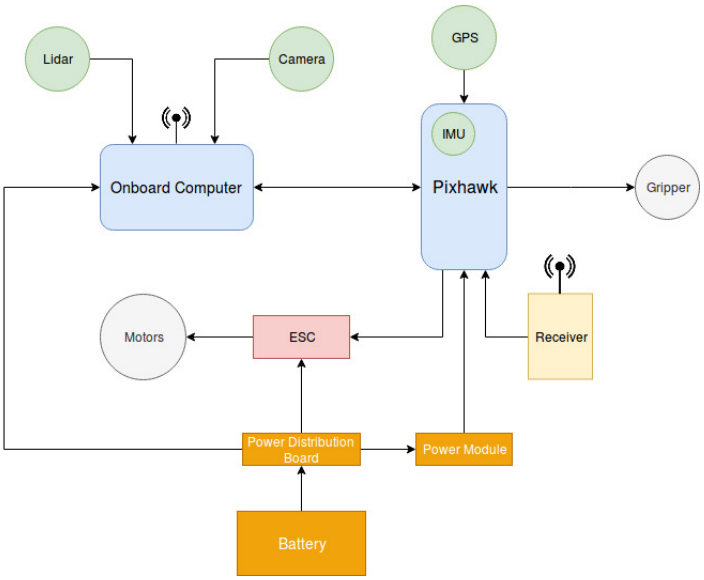


Figure 11: The Complete Hardware Architecture

4.2 Software

The software architecture on the Odroid XU4 and Intel NUC includes data processing from the sensors, the finite state machine and the high level controller. This collectively provide the system object detection and payload delivery capabilities. For computation intensive tasks (DNN based detection and mapping), our system would use the NVIDIA Jetson TX2.
The NMPC requires state estimation (provided by GPS, IMU and LiDaR) and the desired trajectory as inputs to provide high-level control commands as outputs. These commands are then sent to the low level controller, which then runs a PID controller to follow these commands.
The nadir camera provides the images to the object detection and DNN pipeline which outputs the pose of the detected object, crashed MAVs and the house. The servo motor based gripper is work in two states:- on and off. All these states of the system are managed by the finite state machine. Please refer to Figure 12 for a complete software overview.

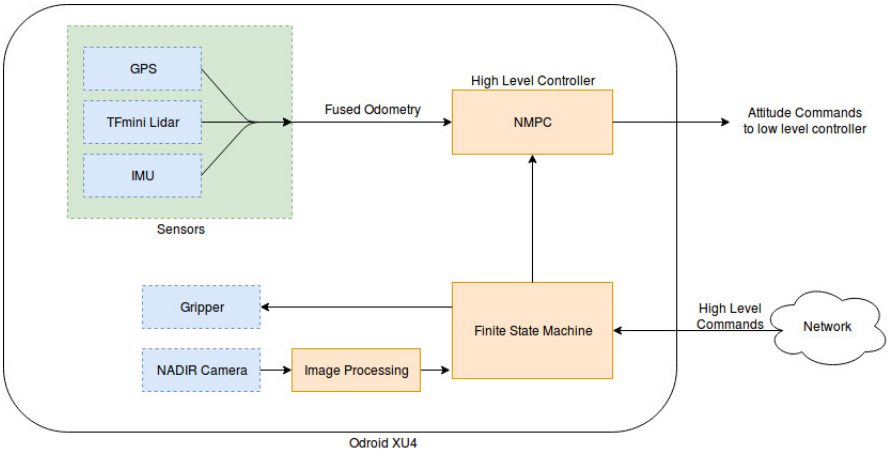


Figure 12: The complete software architecture

4.3 Integration

The complete framework is integrated by the Robot Operating System (ROS). All the three MAVs are interconnected in the ROS network with Ground station by the use of FKIE Multi-master (20) setup, which provides a method to run independent ROS masters on each of the systems in contrast to the single master systems, keeping other systems intact in case of a network failure. Figure 13 shows the ROS network of the system.

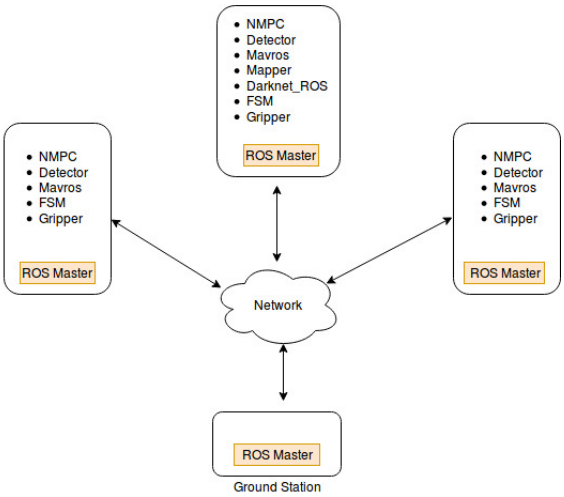


Figure 13: ROS Integration

Bibliography

[1] Mina Kamel, Michael Burri, and Roland Siegwart. Linear vs nonlinear MPC for trajectory tracking applied to rotary wing micro aerial vehicles. *CoRR*, abs/1611.09240, 2016.

[2] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 298–304, Sep. 2015.

[3] VICON motion capture system. Website. [Online] `http://www.vicon.com/`.

[4] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016.

[5] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, Sep. 2004.

[6] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[7] PAL Robotics. ArUco ROS: Software package and ros wrappers of the aruco augmented reality marker detector library. `https://github.com/pal-robotics/aruco_ros`, 2014–2018.

[8] M. Bhargavapuri, J. Patrikar, S. R. Sahoo, and M. Kothari. A low-cost tilt-augmented quadrotor helicopter : Modeling and control. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 186–194, June 2018.

[9] Krishnraj S. Gaur, Hardik Parwana, Ajay Bhatt, Gaurav Pandey, and Mangal Kothari. *Low Cost Solution for Pose Estimation of Quadrotor*.

[10] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6235–6240, 2015.

[11] MAVlink (2014) MAVlink micro air vehicle protocol. Website. [Online] `http://mavlink.org`.

[12] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. ACADO toolkit-an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, may 2010.

[13] Lorenz Meier and MAVLink developer team. QGroundControl: Cross-platform ground control station for drones. `http://qgroundcontrol.io/`, 2014–2019.

[14] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016.

[15] Petr Štěpán, Tomáš Krajník, Matěj Petrlík, and Martin Saska. Vision techniques for on-board detection, following, and mapping of moving targets. *Journal of Field Robotics*, 36(1):252–269, 2019.

[16] HAROLD HOTELLING. RELATIONS BETWEEN TWO SETS OF VARIATES*. *Biometrika*, 28(3-4):321–377, 12 1936.

[17] Marko Bjelonic. YOLO ROS: Real-time object detection for ROS. `https://github.com/leggedrobotics/darknet_ros`, 2016–2018.

[18] Open drone map: A command line toolkit to generate maps from drones. Website. [Online] `https://github.com/OpenDroneMap/ODM`.

[19] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, Nov 2011.

[20] Alexander Tiderko, Frank Hoeller, and Timo Röhling. *The ROS Multimaster Extension for Simplified Deployment of Multi-Robot Systems*, pages 629–650. Springer International Publishing, Cham, 2016.