Indian Institute of Technology, Kanpur

Proposal for D.R.D.O. SASE's UAV Fleet Challenge
Inter IIT Tech Meet 2019

November 30, 2019

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. We have participate at various national/international aerial robotics competitions, few of our past achievements include the 6th and 7th Inter IIT Tech Meet and the International Micro Aerial Vehicle (IMAV) Outdoor Challenge held in October 2019 in Madrid, Spain. The IMAV problem statement involved three major components: (a)Detection, (b)Delivery and (c)Mapping. We successfully implemented swarm technology to achieve this using three MAVs for an area of 30,000 m^2. Apart from this the team works towards contributing to the global open source robotics community and implement various industrial challenges.

Past Work[1]

Our team has worked with state-of-the-art techniques in the domain of aerial vehicles, including the field of controls, localization and mapping. Few of our key works include:

- **Controllers**
  - PID Controller
  - Nonlinear Model Predictive Controller (NMPC)[2]
- **Localization**
  - Visual-Inertial Odometry (VIO): We used the Intel Real Sense Camera for feature detection and subsequent indoor localization using the ROVIO[3] algorithm.
- **Swarm Communication**
  - We implemented a long-range communication link (2 km omni-directional range), to achieve Wi-Fi connectivity among the quadcopters which shared data via the Multimaster setup.

[1] 1. GitHub 2. YouTube 3. GitBook Documentation

[2] Kamel, Burri, and Siegwart, Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles. 2016 CoRR.

[3] Bloesch et al., Robust visual inertial odometry using a direct EKF-based approach. 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

1 System Description

(a) MAV-1: A system with heavy payload carrying capabilities, equipped with Nvidia Jetson TX2 and VIO sensor

(b) MAV-2: A system capable of high speed surveying equipped with RTK-GPS and an OCam Camera

(c) MAV-3: A system capable of detection and delivery, equipped with Nvidia Jetson TX2, OCam and servo grippers

Figure 1: Prototype Systems

1.1 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 this challenge. The MAVs use GPS for horizontal position estimation and a LiDaR for height estimation. For velocity estimation, the MAVs use fusion of attitude (provided from the on-board IMU) and position data. For rotational degrees of freedom, the MAVs use on-board IMUs with sensor fusion using an Extended Kalman Filter (EKF) algorithm provided by the PX4 autopilot[4].

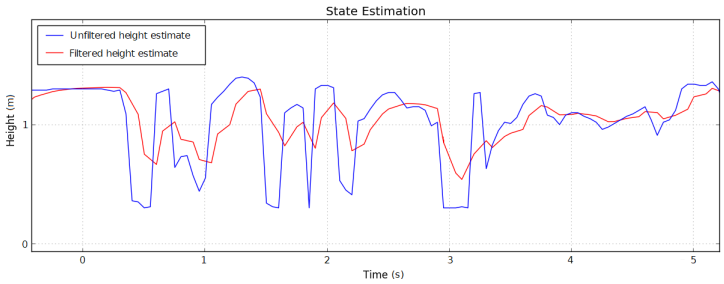

Figure 2: Height vs Time plot for unfiltered (blue) and filtered (red) height estimate after applying the EKF

1.2 Control System

The system uses a two-level cascade controller as shown in Figure 3:

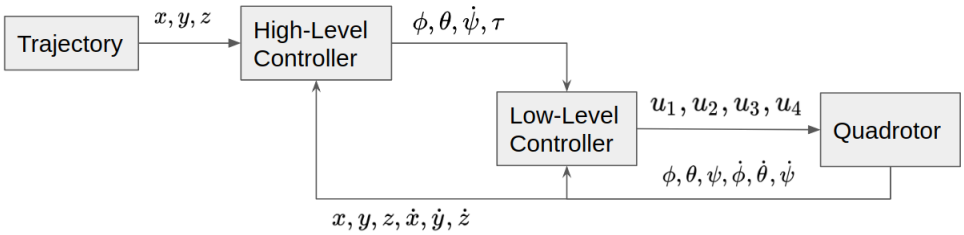

Figure 3: Control architecture of the MAV. The high-level NMPC controller takes a trajectory as input, and outputs roll (ϕ), pitch(θ), yaw-rate($\dot{\psi}$) and thrust(τ) commands. A low-level controller runs on attitude feedback and commands the motor speed.

[4]Meier, Honegger, and Pollefeys, PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. 2015 IEEE International Conference on Robotics and Automation (ICRA).

1.2.1 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[5] architecture provided by PX4.

1.2.2 High-level NMPC controller:

If the need arises, in order to handle robust and complex maneuvers, a Non-Linear Model Predictive Controller will be used. 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[6] to solve the NMPC. The controller is implemented in a receding horizon fashion, where the optimization problem needs to be solved at every time step and only the first control input is actually applied to the system.

2 Approach and Module-wise Division of Tasks

The complete problem statement would be accomplished using three MAVs. Initially after autonomous takeoff, all of the MAVs execute a predefined trajectory in search of the boxes. Each MAV detects the boxes independently and relays the box coordinates to the ground station. Count of detected objects will be shared with each MAV. Once four boxes are detected, the MAVs initiate landing. The whole task 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[7] (one of the Boost C++ libraries).
A brief description of all the submodules is as follows:

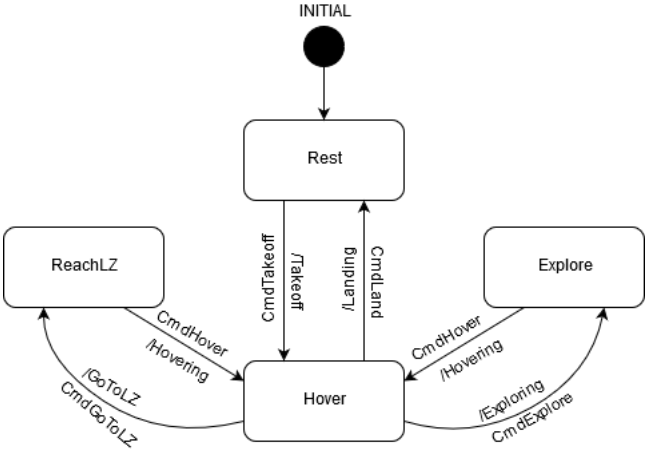

Figure 4: State Machine Diagram

2.1 Autonomous Takeoff and Landing

This module handles the autonomous takeoff procedure. At the start of the mission, high-level commands are sent to the controller to achieve takeoff. At the time of landing, the module achieves controlled descent by sending required high-level commands using height estimate as feedback.

2.2 Exploration

This module includes the commands and trajectories that the MAVs would be following in order to collect data for object detection. Once this state is activated after the takeoff, the MAVs 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[8], and are optimized to capture maximum area in minimum distance and time. The field of view of the camera is taken into account for the height and distance between adjacent passes. The complete geometric area of the arena is divided into polynomial segments[9] for each MAV in a collision free manner.

2.3 Object Detection

The purview of this module extends to the detection of boxes seen by the MAVs during the exploration of the arena. This module takes live image streams from the camera mounted on the MAVs and processes it to find potential candidates for the target boxes. Then, a series of checks are used to narrow it down to the

[5]MAVLink (2014) MAVLink messenger protocol. Website.
[6]Houska, Ferreau, and Diehl, ACADO toolkit - An open-source framework for automatic control and dynamic optimization. 2010 Optimal Control Applications and Methods.
[7]Boost Meta State Machine (MSM) Library. Website.
[8]Meier and team, QGroundControl: Cross-platform ground control station for drones. Website.
[9]Richter, Bry, and Roy, Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. 2016 Robotics Research, Springer.

target boxes alone. To prevent re-detection of same boxes, the MAVs compare the GPS coordinates of the new objects with those of the previously detected boxes.

2.3.1 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. The image is then converted to the HSV color space and a thresholding algorithm is applied to obtain only the pixels of a certain color. Contours are then extracted from the binarized image and further processed.

2.3.2 Contour Processing

The contours previously obtained are first checked for a minimum pixel count. Contours smaller than this minimum are discarded. The centre of the contour is obtained and then the corners are found by computing the convex hull of the contour. Firstly, we compare the area enclosed by the hull and the total area of the segment (For a convex polygon these must be almost equal). Secondly, since dimensions of the boxes are much smaller than the height from which they are viewed, the boxes will appear as quadrilaterals (hexagons at most). So we compare the lengths of the diagonals of the segment, for a regular polygon these must almost be equal.
Contours that are valid after these checks are added to a vector of valid objects, which is the final output of the object detection algorithm.

2.3.3 Location Estimation

Once the object detection algorithm produces a vector containing all the data of valid objects, 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. These coordinates are then relayed to the ground station for map generation and also relayed to the other MAVs.

2.3.4 Results

After extensive testing we found that the approach is reliable and gives practically usable data. The only issues being the thresholding which is sensitive to changes in HSV Values due to change in lighting conditions, exposure of camera, white balance and other camera parameters. We have been testing the location estimates with different algorithms and their relative detection. However, no need as of now has arised to move towards learning approaches. But on the later rigorous tests, we do have the room for the same using the Nvidia Jetson TX2. Figure 5 shows results of the algorithm applied to a green object.

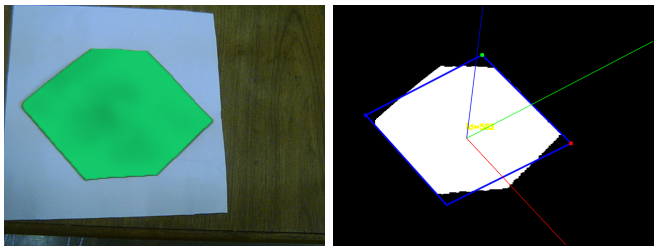

Figure 5: Captured Image of a green object along with estimated pose of camera

2.4 Swarm Technology

All the MAVs are connected to a WiFi network with independent ROS[12] Masters running onboard each MAV. All the three MAVs are interconnected in the ROS network with the ground station by the use of the FKIE Multimaster[13] setup. The count of the boxes detected is published on a ROS topic continuously. This data is read by the other MAVs and ground station. Apart from the objects, each MAV continuously shares its own position which is estimated in local coordinate frame by other UAVs and if probability of collision arises, the MAVs change their own trajectories to safely avoid the same. Apart from these data any other crucial information can be shared as per the requirement.

3 Architecture

3.1 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 are mounted with Odroid XU4 and NVIDIA Jetson TX2 according to the computational requirements.
Vision System: The MAVs are mounted with Bluefox or OCam Cameras for object detection.
Flight Controller: Each system includes a Pixhawk 2.1 Cube Flight Controller.
Sensors: The MAVs are also equipped with the Here+ GPS with RTK capabilities and the Benewake TFmini 1-D LiDaR for height estimation.

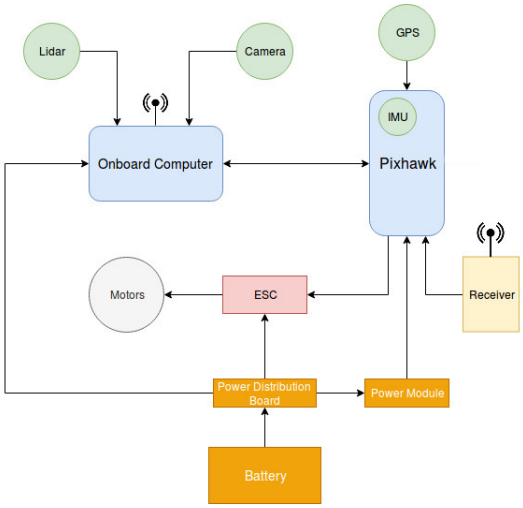

Figure 6: Complete Hardware Architecture

3.2 Software

The software architecture on the Odroid XU4 and Nvidia Jetson TX2 includes data processing from the sensors, the finite state machine and the high level controller. 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 box detection pipeline which outputs the box coordinates. All these states of the system are managed by the finite state machine. Please refer to Figure 6a for a complete software overview.

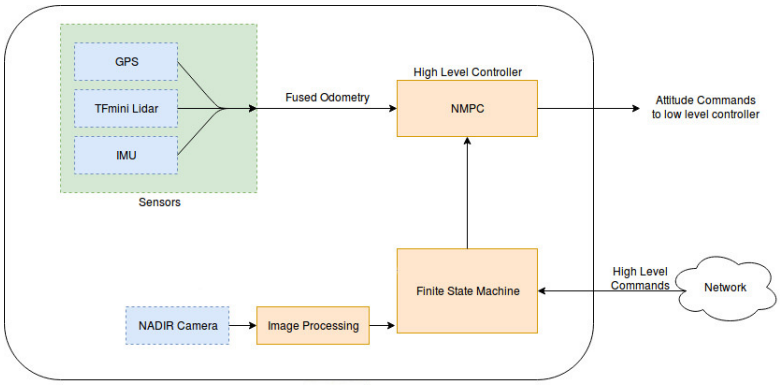

Figure 7: (a) Complete Software Architecture

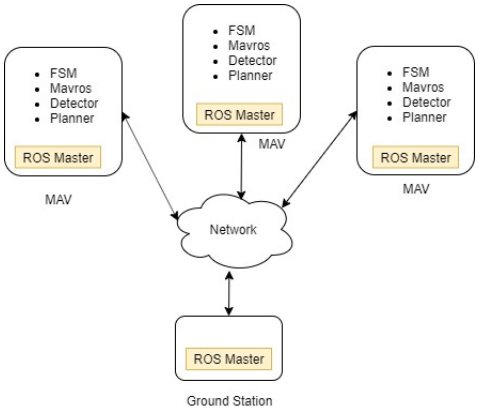

Figure 7: (b) ROS Integration

3.3 Integration

The complete framework is integrated by the Robot Operating System (ROS)[10]. All the three MAVs are interconnected in the ROS network with Ground station by the use of FKIE Multimaster[11] 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 7b shows the ROS network of the system. A basic GUI is implemented based on RQt which displays all the information relevant to the mission such as the number and coordinates of boxes detected.

[10]Quigley et al., ROS: an open-source Robot Operating System. 2009 Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics.
[11]Tiderko, Hoeller, and Röhling, The ROS Multimaster Extension for Simplified Deployment of Multi-Robot Systems. 2016 Robot Operating System (ROS): The Complete Reference (Volume 1), Springer.