Team Aerial Robotics

Indian Institute of Technology, Kanpur

D.R.D.O. SASE's UAV Fleet Challenge

Technical Report

Date of Submission:

January 10, 2020

Contents

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.
We have participated 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.
Apart from this, the team works towards contributing to the global open source robotics community and implementing various industrial challenges.

1.1 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. Few of our major projects include:

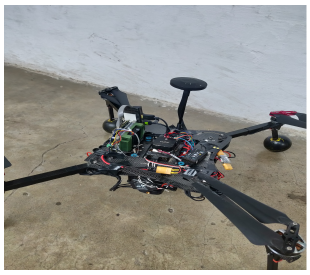
- **Controllers**
  - PID Controller
  - Nonlinear Model Predictive Controller (NMPC) (1)
- **Localization**
  - Visual-Inertial Odometry (VIO): We used the Intel Real Sense Camera for feature detection and subsequent indoor localization using the ROVIO (2) algorithm.
- **Swarm Communication**
  - We 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.

1.2 Past Competitions

- 3rd Position, Warehouse Inventory Check :: 6th Inter IIT Tech Meet, IIT Madras (3) (4)
- 2nd Position, PlutoX Hackathon :: 7th Inter IIT Tech Meet, IIT Bombay
- International Micro Aerial Vehicle (IMAV) Competition 2019 :: Outdoor Challenge, Madrid, Spain: The problem statement involved three major components: (a)Detection, (b)Delivery and (c)Mapping. Swarm technology was to be employed to achieve this in an area of 30,000 m^2.

2 System Description

(a) MAV-1: A system with heavy payload carrying capabilities, equipped with NVIDIA Jetson TX2


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

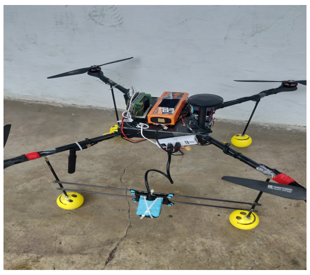

(c) MAV-3: A system capable of detection and delivery, equipped with NVIDIA Jetson TX2, OCam and servo grippers

Figure 1: Prototype Systems

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 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 (5).

2.2 Control System

The system uses a two-level cascade controller as shown in Figure 2:

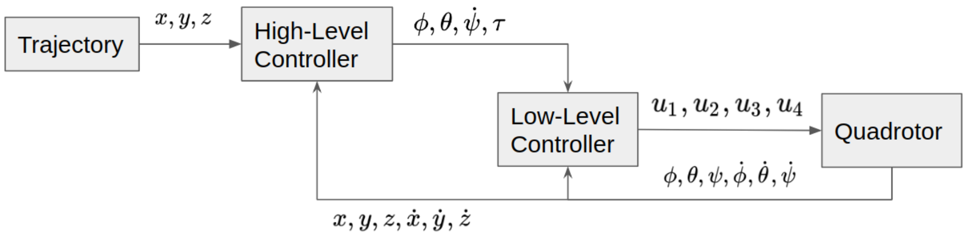

Trajectory
x, y, z
High-Level Controller
$\phi, \theta, \dot{\psi}, \tau$
Low-Level Controller
u_1, u_2, u_3, u_4
Quadrotor
$\phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}$
$x, y, z, \dot{x}, \dot{y}, \dot{z}$

Figure 2: 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 (6) architecture provided by PX4.

2.2.2 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 (7) 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:

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

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 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 (8) (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:

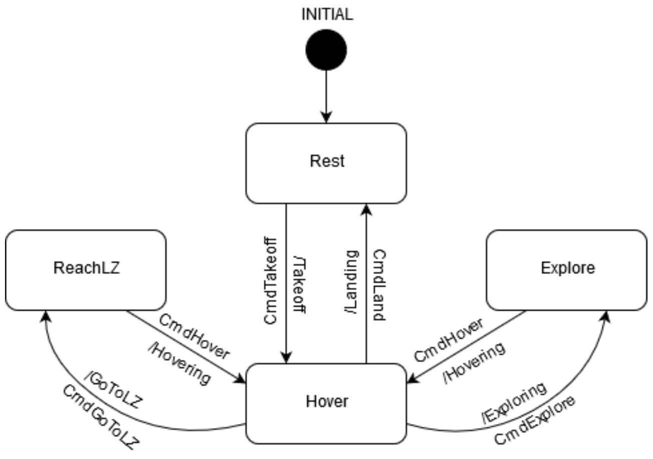


Figure 3: State Machine Diagram

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.

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

A brief description of all the submodules is as follows:

3.1 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.

3.2 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 (9), 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 (10) for each MAV in a collision free manner.

3.3 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 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.

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. 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.

3.3.2 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 as in Algorithm 1 and then removing the outliers as specified in Algorithm 3. 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.

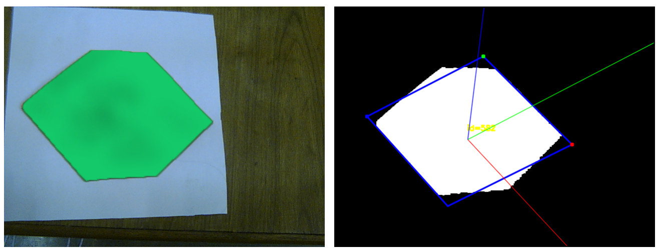


Figure 4: Captured Image of a green object along with estimated pose of camera

Algorithm 1 Object Detection Algorithm

```
Input   : src - Input image
Output: hulls - Array of detected object centres
thresh_img = HSVthreshold(src)    // The input image is thresholded using the HSV parameters of required target to obtain a binary image
thresh_img = MorphOpen(thresh_img)  // A morphological opening operation is applied on the binary image to eliminate noise
list_contours = findContours(thresh_img) // The contours are then extracted
initialize hulls with size = contour.size() and all values = 0
for $i = 0$ to $i = contour.size() - 1$ do
  if $contourArea(list\_contours.at(\text{i})) > MinContourArea$ then
      hulls.at(i) = ConvexHull(list_contours.at(i))// The ConvexHull returns indices of detected hull points
      corners = OutlierFilter(list_contours.at(i), hulls.at(i))
      if corners.size()==4 then  return corners; // Only allow quadrilaterals
    end if
  else
    skip contour
  end if
end for
```

Algorithm 2 Curvature

```
Input   : contour - contour, point_index - index of the point in the contour whose curvature is being calculated
Output: curve - Measure of curvature at a point used to characterize corners
$\delta = contour.size()/8$ // For a square
$left\_vec = contour.at(point\_index) - contour.at(point\_index - \delta)$
$right\_vec = contour.at(point\_index) - contour.at(point\_index + \delta)$
$curve\_vec = contour.at(point\_index - \delta) - contour.at(point\_index + \delta)$
$curve = MagnitudeofCrossProduct(left\_vec, right\_vec)$
$curve = curve/Norm(curve\_vec))$
return curve;
```

Algorithm 3 Outlier Filter

Input : *hull* - detected hull, *contour* - detected contours

Output: *corners* - corners after exclusion of false detected hull points

```
$x_{max} = 0$
$y_{max} = 0$
initialize $curvature$ with size = contour.size() and all values = 0
initialize $max$ with size = contour.size() and all values = 0
$\delta = contour.size()/8$ //For a square
  for $i = 0$ to $i = hull.size() - 1$ do
    $curvature.at(hull.at(i)) = \text{Curvature}(contour.at(hull.at(i)))$
  end for
  for $i = 0$ to $i = contour.size() - 1$ do
    if $curvature.at(i) > curvature.at(i+1) \&\& curvature.at(i) > curvature.at(i-1)$ then
      $max.at(i) = curvature.at(i)$
    end if
  end for
  for $i = 0$ to $i = contour.size() - 1$ do
    for $j = i - \delta$ to $j = i + \delta$ do
      if $max.at(j) == 0$ then
        continue;
      end if
      if $max.at(j) > y_{max}$ then
        $x_{max} = j$
        $y_{max} = Curvature(contour.at(j))$
      end if
    end for
  end for
  for $i = 0$ to $i = max.size() - 1$ do
    if $max.at(i) != 0$ then
      $corners.push\_back(contour.at(i)$
    end if
  end for
  return corners;
```

3.3.3 Location Estimation

Once the object detection algorithm produces a vector containing all the data (i.e 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.

3.4 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.

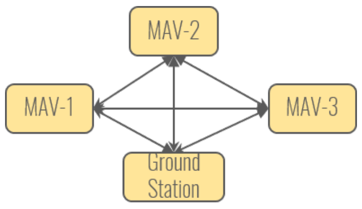


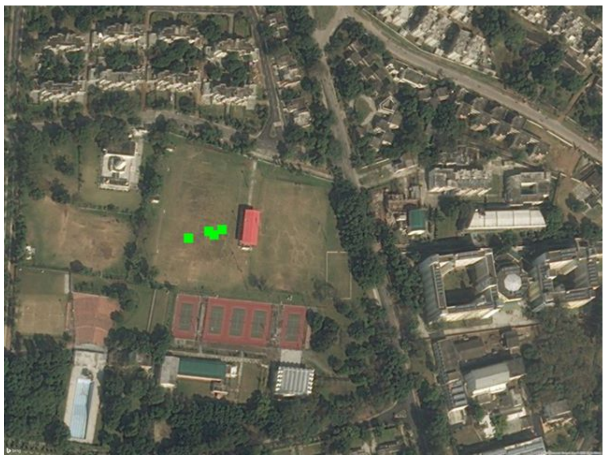
Figure 5: Swarm Network Diagram

3.5 Map Generation

For dynamic map generation, the Bing Maps API© is used to generate an aerial image of the field. An empty map image is stored locally before the mission. The router table is constantly queried by a ROS node during the mission to check for coordinates of new objects. As soon as an object is detected, the ROS node marks the point on the map. Using the center coordinates of the map, the scale, and some geodesic calculations, the pixel coordinates of the point to be marked is obtained. Then the node updates the map locally, and calls a service to another node which then loads the new image and publishes it

For static map generation, the python module named **folium** is used to generate a web page with the map and the markers denoting the locations and GPS coordinates of the objects, once the mission is over.

(a) Map stored offline at the ground station


(b) Detected objects marked on the map

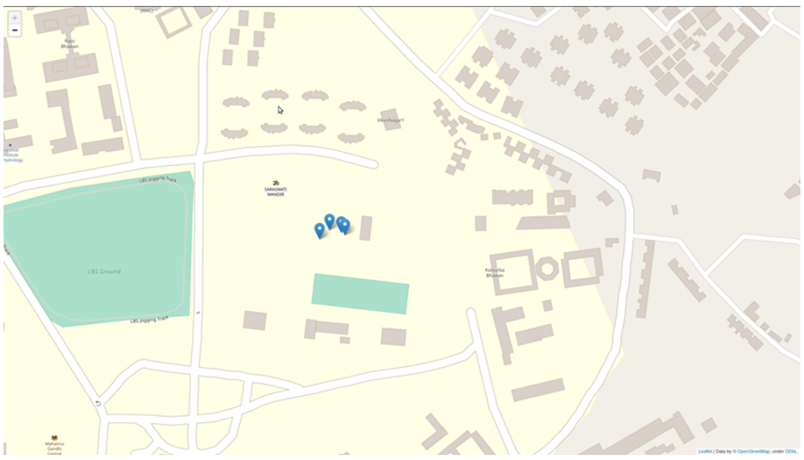


Figure 7: Detected objects marked on the map, generated using *Folium*

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 are mounted with NVIDIA Jetson TX2.
Vision System: The MAVs are mounted with OCam 5MP 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 a Rangefinder for Height estimation.
Propulsion System: TMotor high efficiency Propulsion system.

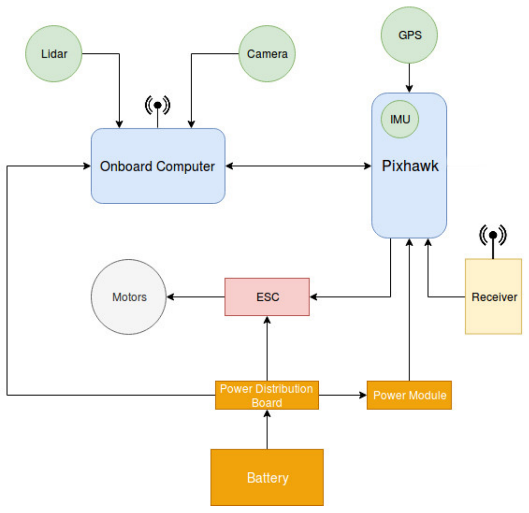
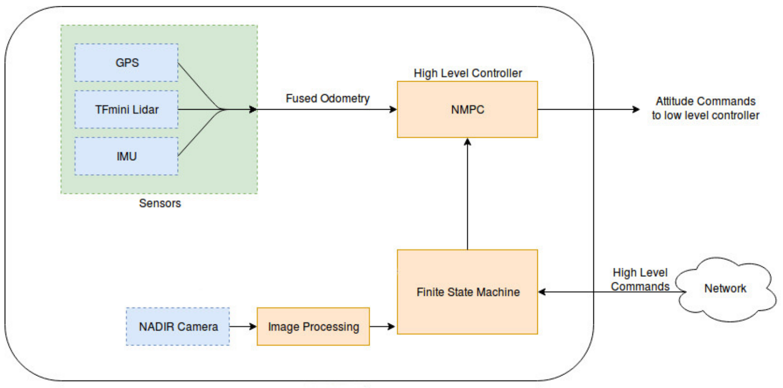


Figure 8: Complete Hardware Architecture

4.2 Software

The software architecture on the NVIDIA Jetson TX2 includes data processing from the sensors, the finite state machine and the computer vision algorithms. The commands for the mission mode are sent to the low level PID controller, which then executes the command. 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 9a for a complete software overview.



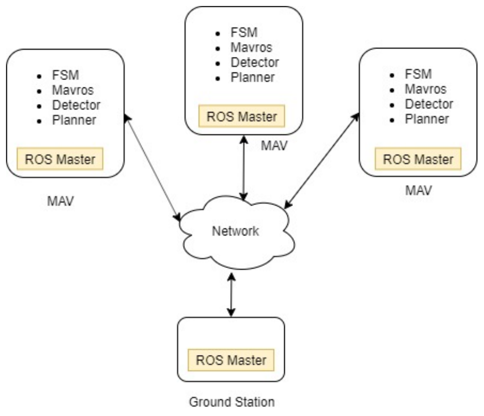


Figure 9: (a) Complete Software Architecture

Figure 9: (b) ROS Integration

4.3 Integration

The complete framework is integrated by the Robot Operating System (ROS) (11). All the three MAVs and the Ground station are interconnected in the ROS network by the use of FKIE Multimaster (12) 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 9b shows the ROS network of the system.

5 Supplementary Material and Contact Details

Feel free to contact us! We will be really happy to hear from you, be it about discussion of some new idea for a technology transform or about your support that can immensely help us.

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

Contact Us:

Pence Mataria
(Team Head)
9727799001
pencem@iitk.ac.in
GitHub: https://github.com/PenceMataria

Bibliography

[1] Mina Kamel, Michael Burri, and Roland Siegwart. Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles. 2016 corr. *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. 2015 ieee/rsj international conference on intelligent robots and systems (iros). In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 298–304, Sep. 2015.

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

[4] Krishnraj S. Gaur, Hardik Parwana, Ajay Bhatt, Gaurav Pandey, and Mangal Kothari. *Low Cost Solution for Pose Estimation of Quadrotor. 2018 AIAA Information Systems-AIAA Infotech @ Aerospace.*

[5] 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). *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6235–6240, 2015.

[6] MAVlink (2014) MAVlink messenger protocol. Website. Website. [Online] `http://mavlink.org`.

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

[8] Boost meta state machine (msm) library. Website. Website. [Online] `https://www.boost.org/doc/libs/1_64_0/libs/msm/doc/HTML/index.html`.

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

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

[11] 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. 2009 proc. of the ieee intl. conf. on robotics and automation (icra) workshop on open source robotics. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[12] Alexander Tiderko, Frank Hoeller, and Timo Röhling. *The ROS Multimaster Extension for Simplified Deployment of Multi-Robot Systems. 2016 Robot Operating System (ROS): The Complete Reference (Volume 1), Springer*, pages 629–650. Springer International Publishing, Cham, 2016.