Autonomous Quadcopter Project - Building A Level 4 Autonomous UAV

Quadcopter Team - Team Robocon

Birla Institute of Technology and Science, Pilani

Abstract

This paper proposes a Level 4 Autonomous Quadcopter UAV (Unmanned Aerial Vehicle) capable of navigating in GPS (Global Positioning System) denied environments, relying purely on its onboard sensors consisting of a 3D LiDAR (Light Detection and Ranging), an RGB-D camera and an accurate IMU (Inertial Measurement Unit) for all its sensing requirements. It will be equipped with a strong onboard computer for handling real-time processing applications. In this project we will develop a software framework based on the Robot Operating System (ROS) for the required autonomy level. Furthermore, we will present the preferred hardware integrations and a selection of relevant APIs. The source code for this project will be made open source to promote research in the domain and encourage the worldwide community of developers to suggest improvements and collaborate. The whole implementation will be tested rigorously at various levels. We have divided the whole software stack into 3 main subsystems namely - State Estimation and Localization, Visual Perception, and Motion Planning. The final setup of the quadcopter consists of an Nvidia Xavier NX onboard computer with the sensor package consisting of a Velodyne Puck LITE LiDAR, a ZED 2 stereo camera, and a Vectornav VN 100 IMU. Pixhawk 4 flight controller will be used as the main controller unit for mapping the trajectory waypoints and the velocity profile to the motor channel inputs. The State Estimation subsystem will cater to the task of Localization in 3D space by fusing the data from different sensors and use it to generate a pose estimate. Visual Perception will use advanced Computer Vision algorithms to get intricate details of the environment scenario by Mapping the environment (static obstacle), detecting dynamic obstacles and estimating their pose and heading. Finally, the Motion Planning subsystem will take the estimated state from the State Estimation subsystem and the obstacle set (static and dynamic) from Visual Perception and will fuse them to generate a trajectory and velocity profile based on set constraints and user inputs for source and destination points. Further specifics related to these subsystems will be discussed in detail in the subsequent sections.

Introduction

The Unmanned Aerial Vehicle (UAV) sector has become a multi-billion-dollar industry since its inception and has gained a lot of attention by both academia and industry, especially in the last decade. It is by far one of the most flourishing fields in terms of innovation in the 21st century. There has been a sudden explosion of research opportunities in the sector related to autonomous aerial systems. Autonomous UAVs have pushed the existing cutting-edge technologies and algorithms to their limits making room for evolution of the current paradigms in robotics. Corporate companies have started to explore UAVs for various applications like autonomous logistics, surveillance and even military assessments for example. E-Commerce giants like Amazon have established spin-offs like Prime Air with the objective of achieving rapid last mile delivery using high level autonomy drones. Therefore, this project provides an opportunity for the students and the institute to make a significant contribution to both industry and academia.
In July 2018, the Federation of Indian Chambers in Commerce and Industry (FICCI) organized an event "Make in India for Unmanned Aircraft Systems (UAS)" based on which they later released a report by the same name. Therein the committee recognized the direct application of autonomous and manual aerial systems in the following sectors: Power and Utility, Agriculture, Highways, Mining, and Railways, among others. All of these are product categories of preference as declared by the Public Procurement of the Make in India scheme. Furthermore Aerospace and Defense, Railways and construction are among the 27 champion sectors recognized by the Department of Promotion of Industry and Internal Trade (DIPP, India). Hence, there is a high probability that this project will receive great support from the concerned government departments if successful, at a later stage.

According to the industrially accepted scale of the 5 levels of drone autonomy[1], L4 (level 4) autonomy implies that the drone should have backup systems for most of its hardware stack, so that if one fails the platform will still be operational enough to execute a smooth and planned fail safe maneuver thus avoiding any damage to property and life. Also, L4 drones are capable enough to plan their own mission given start and end points in a pre-flight developed map as user input. Pilot's intervention is not required at any stage in a L4 drone. If development of an L4 drone is successful then it's only a matter of time before it is extended to L5, because once the required framework is set L4 to L5 transition will only require replacement of certain decision making algorithms with evolving AI related optimization techniques. Hence, this project is a first step towards attaining fully autonomous intelligent aerial systems.

Such a L4 autonomous UAV has large-scale applications as recognized in the report by FICCI and accounts by Economic Times. National Highway Authority of India (NHAI) project consultants use drones for 3D digital mapping of highways. Drones were used on a 250 km stretch of the Nagpur-Mumbai Highway in 2016. This process can be fully automated by a small extension of our proposed UAV system for adapting to outdoor environments. Drones give accurate data which help access the progress of work and comparison with ground reports unlike manual surveys. As a result, the National Railway System used drones for videography inspection of 37,000 km of railway tracks and 3-D mapping of a 3,360 km corridor, a direct application of our project. FICCI's report projects that at the current rate of increasing adoption of UAS in

[1] The 5 levels of drone autonomy for non-military applications as accepted by drone industries defines the levels as follows:

Level 0 - No Automation: The drone is 100% manual.; Level 1 - Low Automation: Pilot remains in control. Drone has control of at least one vital function.; Level 2 - Partial Automation: Pilot remains responsible for safe operation. Drone can take over heading, altitude under certain conditions.; Level 3 - Conditional Automation: Pilot acts as fail-back system. Drone can perform all functions given certain conditions.; Level 4 - High Automation: Pilot is out of the loop. Drone has backup systems so that if one fails, the platform will still be operational.; Level 5 - Full Automation: Drones will be able to use AI tools to plan their flights as autonomous learning systems.

India, by 2021, the value of industry and market would be around US$ 885.7 million, while global market size will touch US$ 21.47 billion. Hence, innovation in the domain will be highly rewarding and the project is worth investing into.

Furthermore, this project will be the first of its kind in the history of BITS Pilani, Pilani campus. It is our humble expectation as a student tech team that this project will set up a new standard for tech teams and will encourage the upcoming generations of students to explore robotics to a whole new level. This project, owing to its interdisciplinary nature, will attract students from all disciplines to join technical research.

In this paper henceforth, we provide a novel approach to make an Autonomous Quadcopter UAV. We will start with an assessment of existing systems under research both within and outside the country, and then we will present an analysis of the finalized system architecture and delve a bit deeper into the subsystem specific details based on the research done by the team so far. Note that all of the proposed algorithms have been decided by a careful assessment of the pros and cons of the existing frameworks and the time required for implementation and hence are subject to change if a better option is explored in the future.

Related Work

Autonomous UAVs provide significant benefits to the above-mentioned sectors as well as in applications related to high performance 3D structure inspection - risk assessment and rapid logistics. Consequently, they have been an important subject of research in the last few years.
The GRASP lab at the University of Pennsylvania developed an autonomous drone swarm which relies on Visual Odometry from a VGA stereo pair and a 4K video camera for navigation and uses an IMU for localization. The swarm was scalable, i.e. new drones could be added without sacrificing performance. The flight controller was made using the Qualcomm Snapdragon Flight development board consisting of a quad-core processor. Only one or two cores, running ROS, were used for flight. The controller they developed was robust enough to handle 18-mph gusts and was able to localize in low light conditions. However, they did not propose a formal framework for indoor navigation in 3D, nor did they develop the system for all environment scenarios. We aim to consider major maneuver scenarios and implement relevant behaviors.

GRASP lab is also working on an ongoing project where they aim to develop Autonomous Micro UAVs for first response and precision farming. They also made a large quadrotor weighing almost 2kg which is claimed to be capable of autonomous navigation indoors or outdoors, through forests, buildings, and farms.

Tomic et al. (2012) [54] designed an autonomous UAV for search and rescue missions targeting both indoor and outdoor environments. Their system has many striking similarities to the one proposed here. They proposed the use of a 2D LiDAR, a camera for object detection and an IMU for pose estimation along with IPC for localization from point cloud data. However, their system did not build a geometric map of the environment, instead it uses known landmarks to correct the drift in position estimates. They used several gumstix boards for computation of high level, low level and sensing tasks. They incorporated several state machines for sequential task planning for entering and leaving houses through windows autonomously. Their behavior planner was also implemented using a finite state machine and serves as an excellent reference for our own behavior planner.

In India, the DRDO Netra project is aimed towards making an autonomous UAV for surveillance and reconnaissance applications. IISc Bangalore's Robert Bosh Centre for Cyber – Physical Systems is working on autonomous outdoor drones and drone swarms for coordinated autonomous flights, while also working on Ultra-wide band transceivers for smart infrastructure to aid the SLAM algorithm. IIIT Hyderabad's Robotic Research Center has worked on fast motion planning on monocular quadcopters (K. Bipin et al.

2015) [58]). Several drone startups like Skylar drones, Bangalore have started working on autonomous UAV solutions.

Technical Details

The Robot Operating System (ROS) [13]

ROS is a software framework for designing and writing robot software. It is a collection of tools, libraries and conventions which aim to simplify the task of modelling robotic behavior and accepts a wide variety of robotic platforms. In a robot there are many sensors and computing units which make up the whole system. Routing and managing the exchange of data (messages) between these sensors and computing units is quite challenging considering that all these nodes (sensors and computers) operate at different frequencies and hence transmit data at different timestamps. Such an asynchronous system requires a set of defined protocols and data routing techniques. This is where ROS comes into the picture. ROS introduces a peer-to-peer topology of message exchanging and provides a variety of protocols relevant to a robotic application. Furthermore, ROS uses a tool-based approach wherein tools are implemented as collections of libraries that perform various tasks, e.g. navigating the source code, get and set configuration parameters, visualize the peer-to-peer connection topology, graphically plot message data, and so on. The biggest power of ROS lies in the fact that it is Open-Source thus allowing to collaborate all the brilliant minds working in the domain to add to the existing set of tools and thus host well documented packages of their own. As a result, ROS re-uses code from numerous other open-source projects like OpenCV, planning algorithms like OpenRAVE, etc. Hence, because of being open-source ROS contains the biggest set of libraries and tools and supports a majority of sensors and simulator software. Versatility, a huge community of researchers and detailed documentation make ROS the best choice when compared to other existing frameworks.

State Estimation and Localization

The main purpose of the State Estimation and Localization subsystem is to accurately measure the position of the quadcopter in each map with the help of the readings obtained from various sensors. The sensors used in this project will be the LiDAR, depth tracking cameras in addition to IMU, barometer, and the GPS (when available) that are inbuilt into the flight controller. As each sensor has some error associated with its readings, it becomes important to take the readings of each sensor and integrate them to get the most accurate state of the quadcopter possible.

Kalman filters are a technique to fuse information, originally introduced in [1] by Kalman.(1960). It takes the output from the available sensors to produce a final estimate of the position of the quadcopter on a given map. It considers the uncertainty in motion and the measurements. This filter takes the probabilistic measure of the state of a system and updates in real-time using two steps- prediction and correction. The various states are expressed as random variables that are specified with their mean values and covariances. The standard Kalman filter estimates the state for linear systems. However, the situation the quadcopter will be dealing with will not be linear. Hence, there are some modifications made to this to include the non-linearity of a system in the calculations.

Extended Kalman Filter: This filter linearizes non-linear systems by expanding the non-linear terms using the Taylor series expansion. This converted linear system represents the deviations from the nominal trajectory of the nonlinear system. Then, the Kalman Filter is used to estimate the deviations. It does not require very high computational power and has a fair amount of success when used for autonomous vehicles, having been widely used. However, the problem arises when the error is highly non-linear, and

the linear approximation leads to large accumulated errors. Hence, this is not a very safe option to use for our requirements. The use of EKF for UAV has been presented in [2] by Mao, Drake & Anderson. (2007)

Unscented Kalman Filter: The UKF solves the linear approximation issue faced when using the EKF. The state distribution is shown by a Gaussian random variable and is specified using a minimal set of chosen sample points that completely capture the true mean and variance of the variable. This filter has been found to achieve a better level of accuracy at a similar level of complexity when compared to the EKF. Hence, this is preferred over the EKF. The UKF is explained in [3] by Wan, Eric & Merwe, Ronell. (2000). As shown in [4] by Julier & Uhlmann(2004), the UKF predicts the state with more accuracy and is also easier to implement than the EKF.

Error State Extended Kalman Filter: The ES-EKF is a modification of the EKF. The state of the quadcopter would have two terms here - the nominal term and the error term. The KF is applied to just the error state. This avoids filtering over any non-linear behaviors. It performs better than the vanilla EKF as the evolution of the error tends to be more linear. This error state formulation makes it easier to work with constrained quantities like rotations in 3D space. It is explained in [10] in Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches.

Particle Filter: This method to estimate the state does not depend on the probability density function of the measurements. It makes no assumptions about the distribution of the noise and, thus, gives the most accurate prediction of the state when compared to the other discussed filters. A drawback of the particle filter is that after a certain number of iterations, almost all the weights become zero. And after some time, the particles that are generated to help measure the state lose their effectiveness. Additionally, it is very computationally expensive, especially for creating 3D maps for localization, and may not give the state within the time required to meet safety requirements. A modified version of the Particle Filter for UAVs was presented in [5] by Braga, Shinguemori & Campos Velho.(2018). The use of Particle Filter for map-matching to estimate the position is shown in [6] by Gustafsson et al.(2002).

A detailed comparison between these filters is given in [7] by Rigatos, Gerasimos. (2012). There are other related papers that use a combination of these filters to improve the tracking accuracy. A solution using the Ensemble Kalman Filter is presented in [8] by Pornasrayouth, Wingsaisuwan & Yamakita.(2011). Another solution combining the Particle Filter and the Unscented Kalman Filter for accurately landing UAVs is shown in [9] by Pessanha Santos, et al. (2015)

Conclusion: As of now, we have decided to implement the ES-EKF in our quadcopter for fusing the sensor data. It performs almost as well as the UKF and requires less computational power. However, if this does not provide satisfactory results, then we will try to use an ensemble EKF, which combines multiple Kalman Filters that shall boost their gains. Another option would be using a KF and PF combination, which will provide higher accuracy as PF performs better in highly non-linear models. The most robust filter will be decided based on the results of the simulations that we shall perform and test each of these filters on.

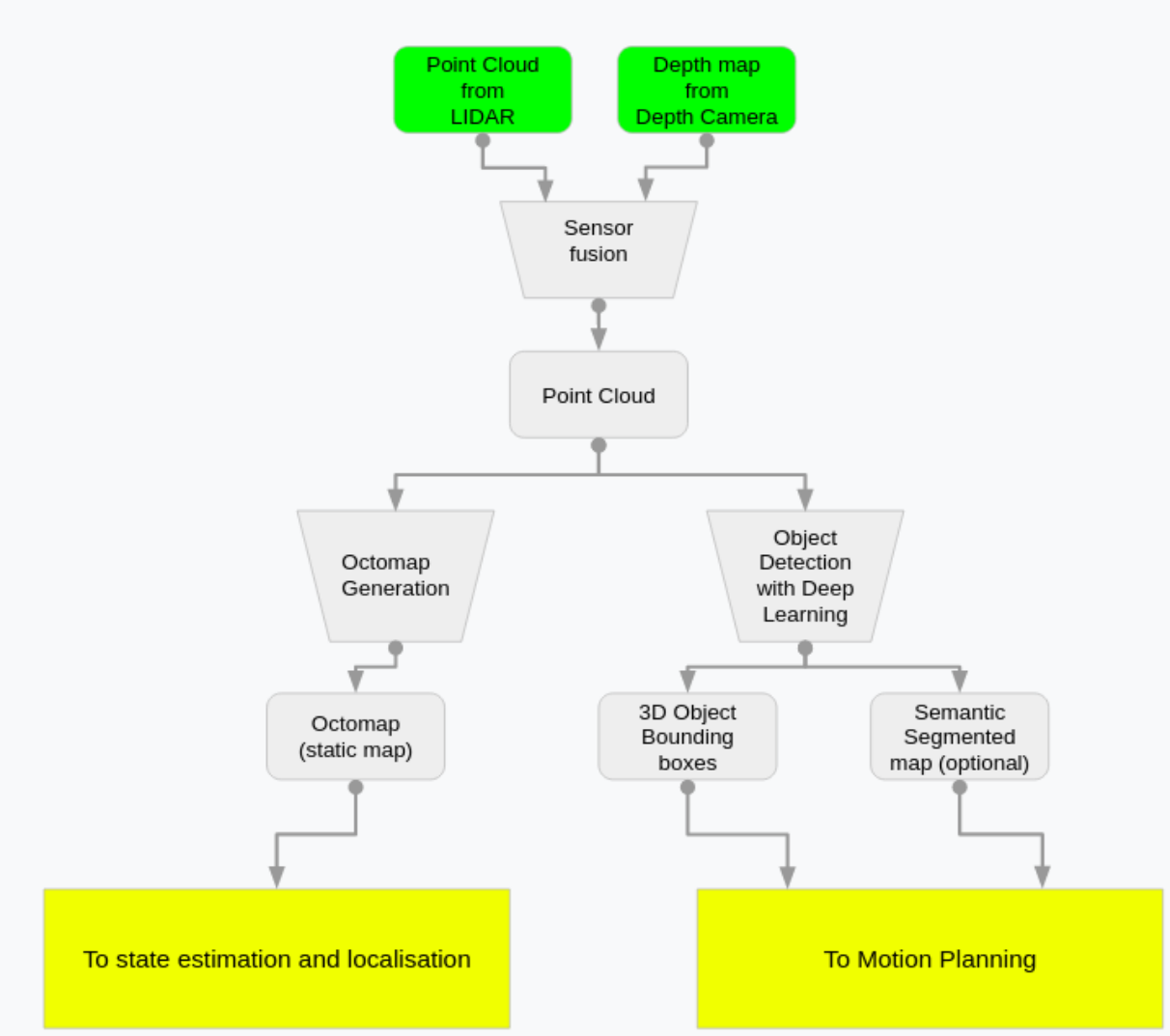
The prediction loop of a filter generally requires a high frequency sensor, which calls for an IMU. An IMU contains an array of Gyroscope, Accelerometer, Barometer sensors which provide accurate rates of change of pose in 6 degrees of freedom and give updates at a frequency of 70-90Hz which is a very good range for prediction, especially for fast moving systems like quadcopters. The problem with such sensors is that, with time, due to MEMS related intricacies (flicker noise in electronics and other effects) a bias (drift from the mean value of the output rate) is introduced in the readings of the sensors. Since the outputs of the gyros and accelerometer are integrated over time the noise magnifies and hence the uncertainty increases over time. This is exactly the reason why an IMU will never suffice as a standalone sensor for state estimation, it must be complemented with a Visual Perception based Localization technique or GNSS. Now since

majority of updates are due to the IMU we need to make sure that the IMU is precise enough. There are two ways to do that. One is to use a FOG (Fiber optics gyro) IMU, which works on Sagnac Effect in light interferometry and is hence very precise and has negligible bias. However, FOG IMUs are impractically large in the context of drones and therefore are out of question. That leaves us with the last option, using a highly accurate MEMS IMU with a low bias. Given below is a table comparing two of the available IMUs, the Vectornav VN-100 and the Ellipse 2 Micro INS.

	Vectornav VN 100 (SMD)	Ellipse 2 Micro INS
Size	24 x 22 x 3 mm	26.8 x 18.8 x 9.5 mm
Weight	3.5 g	10 g
Input Voltage	3.2 to 5.5 V	4 – 15 V
Current Draw	45 mA @ 3.3 V	-
Power	185 mW	400 mW
Operating Temp.	-40° to +85° C	-40 to 85 °C
Accel Range	±16 g	± 16 g
Accel In-Run Bias	< 0.04 mg	± 5 mg
Accel Noise Density	0.14 mg/√Hz	57 µg/√Hz
Gyro Range	±2,000°/s	± 450 °/s
Gyro In-Run Bias	5-7°/hr typ.	± 0.2 °/s
Gyro Noise Density	0.0035 °/s /√Hz	0.15 °/√hr
Magnetic Heading	2.0° RMS	0. 8 °
Heave	5 % or 5 cm	5 cm
Onboard Filter Update Rate	400 Hz (ES-EKF)	Runs an EKF
IMU Data frequency	800 Hz	200 Hz
Angular Resolution	0.001°	
Pitch/Roll Accuracy	0.5°	0.1 °
Documentation	Vectornav	Ellipse 2 Micro Series

Based on the above comparison, we have decided to use the Vectornav VN-100. It has industrial grade MEMS sensors which have a very low bias (5-7o/hr for gyro and < 0.04mg for Accel-in Run bias). It also has lower weight, power consumption, gyro in-run bias, gyro noise density, better magnetic heading, higher frequency and better angular resolution when compared to the Ellipse 2 Micro INS. It also fares much better in these aspects when compared to many of its counterparts. Vectornav VN-100 has a history of being used in a lot of high precision robotics systems such as the MIT Mini Cheetah.
With the localization problem solved we now look at the Visual Perception software stack.

Visual Perception



The Visual Perception subsystem involves capturing the image and the depth data from different sensors, processing it for easy access by other subsystems, and manipulating it to perform computer vision tasks like fusing sensor data to generate a 3d point cloud, object detection, semantic segmentation, visual odometry and mapping. The abstracted pipeline is shown in above figure.

Sensors for perception

The drone needs cameras and sensors to perceive the environment and to do any of the tasks like visual odometry or object detection. To ensure the safety of any being and the drone itself, we plan to use multiple camera modules and lidar. This would be a safeguard against sensor-failure and multiple sensor fusion would also enrich the maps the drone generates.

The following table gives a comparison of the different types of sensors available and their advantages over one another:

LIDAR	***Stereo Camera***	***RGBD***
Wide FOV (360deg)	Better FOV than RGBD, but worse than the LIDAR	Narrow FOV
Lower resolution	High resolution	High resolution
Easier to integrate	Very computationally expensive. Need to process the two different images to get the depth data.	Computationally expensive (need to process the IR projections to generate the depth map.)
Works in most lighting conditions but does not work with reflective surfaces.	Works better in well-lit or outdoor environments.	Works better in dim light
Long range	Medium range	Short range

We plan to use a LIDAR, and have to finalize between a stereo camera or a depth camera, in order to take benefit of a high resolution map of the front generated by the depth camera or the stereo camera. The wider FOV of LIDARs would allow the drone to a 360degree view as objects approaching from behind or sideways would otherwise be undetectable / partially visible with a narrow depth map.
TOF cameras are also an option, but these are not considered here since they are usually worse in all the above-mentioned fields than all the options mentioned above.

Finally, these sensors were selected:
Depth cam: Realsense D435
Stereo cam: Zed 2
LiDAR: Puck LITE

The following table compares the aforementioned sensors:

SENSOR	**Puck Lite (LIDAR)**	**ZED 2(stereo camera)**	**Intel Realsense D435(depth camera)**
Depth sensor	N.A.	N.A.	Depth Field of View: 86° × 57° (±3°) 1280x720@ 90 fps
Depth range	100 m	20 m	10m
Field of View	FOV(H x V): 360° x +15.0° to -15.0° (30°)	FOV(H x V x D): 110° x 70°x 120°	RGB Sensor FOV (H × V × D):69.4° × 42.5° × 77° (± 3°)
Resolution and refresh rate	Rotation Rate: 5 Hz – 20 Hz	Side by Side	RGB Sensor Resolution:

		2x (2208x1242) @15fps Or 2x (1920x1080) @30fps Or 2x (1280x720) @60fps Or 2x (672x376) @100fps	1920 × 1080 @30 fps
Dimensions	71.7mm height ; 103.3mm diameter	175 x 30 x 33 mm	90 mm × 25 mm × 25 mm
Weight	~590 g (without Cabling and Interface Box)	166g	72g
Power consumption	8W (9-18V)	1.9W (5V)	2W (5V)

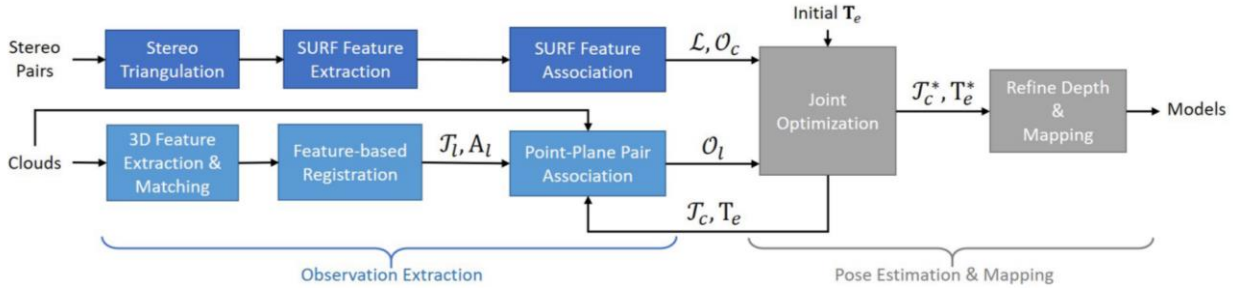
We prefer the Stereo Camera over the Depth camera, because of its higher refresh rate, better resolution, and better FOV.

Fusing the Data from different sensors

Since we plan to make use of multiple sensors for capturing surrounding data, we need to fuse the data that shall be obtained from the sensors at different frequencies and resolutions to get the best point cloud dataset and noise reduction as possible.

The ICP (Iterative Closest Point) algorithms are very popular to fuse LiDAR and Cameras data. The ICP algorithm was first introduced by Chen and Medioni, 1991 [14] and Besl and McKay, 1992 [15]

It is the most widely used algorithms due to broad variety of reasons as can be inferred from Weikun Zhen et al., 2019 [16] (below is an abstract flowchart from the paper summarizing the algorithm)



In recent times multiple optimizations have been made to ICP algorithm. **PLICP (Point to Line ICP)** algorithm is one such big optimization which converges the points quadratically and in finite number of steps (Andrea Censi , 2008 [17]). Another noticeable optimization is the use of **Anderson acceleration** instead of Picard acceleration as rightly explained by A. L. Pavlov et al., 2017 [18].

Besides ICP and PLICP algorithms there are also few other efficient alternatives such as the YOLO algorithm which was proposed by Jian Han et al., 2018[19].

The primary purpose of ICP based algorithms is to be able to estimate the rotation and the translation matrices based on point cloud data from successive iterations of the LiDAR as the quadcopter traverses in the 3D space. These matrices are then sent over to the State Estimation and Localization subsystem for the correction algorithm of the filter used. Since this is a Vision algorithm it does not suffer from electromagnetic and mechanical biases like the IMU does. Therefore, even though the output is probabilistic

it has lower uncertainties than the IMU data and, hence the LiDAR complements the high frequency updates of the IMU by removing the accumulated uncertainty and maintaining accurate state updates.

Feature extraction

To aid motion planning, we plan to perform Visual Odometry, in which we would use the difference between successive frames captured by the camera to estimate its motion. To do this, we must do feature extraction which refers to the process of extracting useful information referred to as features from an input image. The extracted features must be representative in nature, carrying important and unique attributes of the image. It is a fundamental step in any object recognition algorithm.

The widely used algorithms for feature extraction include **SIFT** (Scale Invariant Feature Extraction) algorithm (proposed by Lowe, David G., 1999[20]), **SURF** (Speed Up Robust Feature) algorithm (proposed by Herbert Bay et al., 2006[21]) and the **ORB** (Oriented fast and Rotated Brief) algorithm (proposed by Rublee, Ethan et al, 2011[22]). This sub-topic is still being researched by us and we have not finalized the feature extractor to be used. Right now ORB feature extractor seems to be the most favorable.

In **SIFT** algorithm key points of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database. From the full set of matches, subsets of key points that agree on the object and its location, scale, and orientation in the new image are identified to filter out good matches.

Object detection using **SURF** is composed of three steps: feature extraction, feature description, and feature matching. The main interest of the SURF approach lies in its fast computation of operators using box filters, thus enabling real-time applications such as tracking and object recognition.

ORB performs as well as SIFT on the task of feature detection (and is better than SURF) while being almost two orders of magnitude faster. ORB builds on the well-known FAST key point detector and the BRIEF descriptor. Both of these techniques are attractive because of their good performance and low cost.

Object detection

In this sub-topic we aim to successfully locate, classify and label the objects being detected from 3D point cloud data (there are many advantages in using 3D data for object detection [23]) or from a 2D image using ConvNets [24][25][26][27].

Since Deep Learning is a very active research field, new, better and more optimized solutions keep emerging. There are multiple feasible options available and this domain is currently still in research phase for the project and we have not fully finalized a method. Still, the following approaches look viable

- We can integrate the pre-built models in *librealsense* (the official library provided along with realsense cameras by Intel). These built-in model can be optimized and extended as per our need using the Realsense SDK. There are few open source custom models available.
- The challenges involved with object detection using 3d point cloud data and the solutions that can be used to solve the same have been listed briefly by Shuran Song and Jianxiong Xiao, 2015 [28]. The demo video of their work can be viewed on their university's website. It might be slower as compared to other algorithms as it first takes in a 3D cad model to generate an ensemble of SVMs which are then used to find the object in the scene **(on depth map).**

- NVIDIA's research for object detection on point cloud [29] . NVIDIA provides ResNet based Deep Learning Models as part of its ISAAC SDK, called DetectNetv2. These can be directly used for object detection or tweaked to better fit our needs.

We plan to include sematic segmentation in the visual perception subsystem as we would have more data on the classification of points rather than binary data. A possible approach would be to convert the 3D point cloud data into 2D images (for which semantic segmentation is highly efficient). Ye Wang's presentation, 2016 [30] talks more deeply about semantic segmentation and object detection (with 3d bounding cuboids). The benefits of sematic segmentation are many in an urban environment as has been highlighted by Cordts, et al 2016 [31], Neuhold, G et al, 2017 [32] and few others [33][34].

The detected objects(as 3D bounding cuboids in the point cloud) are then used by the Motion Planning Subsystem, to plot the path whilst avoiding the obstacles.

Mapping

As the drone moves, it generates large maps which would be difficult to store with conventional point clouds since the drone has much larger range both horizontally and vertically compared to ground bots. To solve this problem, we plan to use Octomaps (Hornung et al., 2013 [35]) to store the 3D map which would also be updated periodically.

Octomaps use much less space than traditional 3D point clouds, and thus are faster to compute. Also, since the nodes in an octomap are probabilistic rather than deterministic, they are easier to integrate using multiple sources (sensors).

Octrees and Octomaps

Octrees use hierarchical trees to store the map. The entire perceptible space is divided into 8 cubes, and each of these cubes is further divided into 8 sub-cubes. This process happens recursively until the required resolution is achieved. These can then be used to store a binary map as all the nodes in the tree(smallest sub-cubes) can be set as occupied or unoccupied. Also, it is possible to get a compact representation of the map, because when all the children of a node have the same state, the child nodes can be removed.

Octomaps are similar to octrees, but instead of storing binary states in each leaf, the states are probabilistic. Also, octomaps can accommodate noise by using previous states and estimate the current state of a leaf, thus reducing the sensor noise.

Alternatives:

Point clouds: Point clouds directly store all the perceptible information and are not memory efficient. Furthermore, they do not allow to differentiate between obstacle-free and unmapped areas and provide no means of fusing multiple measurements probabilistically.

Fixed Grids with voxels

A major drawback of rigid grids (Tabak & Jain ,1989 [36]) is their large memory requirement. The grid map needs to be initialized so that it is at least as big as the bounding box of the mapped area, regardless of the actual distribution of map cells in the volume. In large-scale outdoor scenarios or when there is the need for fine resolutions, memory consumption can become prohibitive. Furthermore, the extent of the mapped area needs to be known beforehand or costly copy operations need to be performed every time the map area is expanded.

Conclusion

The probabilistic nature of octomaps is a double edged-sword. Though it helps in cleaning out sensor noise, it also makes it less responsive against dynamic changes . If a leaf was observed free for k times, then it has to be observed occupied at least k times before it is considered occupied according to the threshold (assuming that free and occupied measurements are equally likely in the sensor model) . Also, since the map only gives the coarse nature of a point(whether it is occupied or not) it becomes difficult to do object recognition.

Thus, to make the drone safer against dynamic obstacles, we plan to use both octomaps and point clouds. Octomaps would maintain a map of the entire perceived region, and the smaller point cloud at each instance would be used directly for object detection and (optionally) semantic segmentation.

Motion Planning and Collision Checking

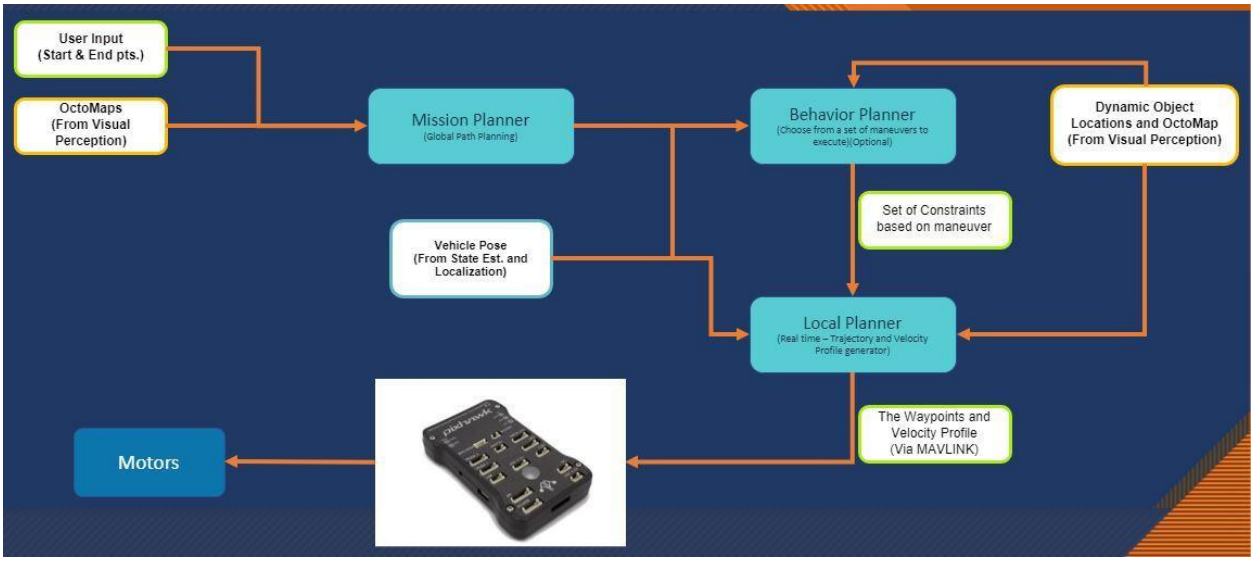


Fig. 1

As the name suggests Motion Planning subsystem will take care of planning the motion between the given start and end point in the pre-generated OctoMaps. The main advantage of OctoMap is its flexibility as the extent of the map does not have to be known as shown in [39].

For Motion Planning and Navigation, we considered using the ROS Navigation Stack [40], Tesseract [41], and MoveIt! Framework [42].

1. **ROS Navigation Stack** is a barebones implementation of SLAM, Motion Planning, and navigation inside ROS, and it is not as feature rich as MoveIt! Or Tesseract. Also, the navigation stack is geared for 2d planning and uses search-based algorithms (SBPL library) [43] to compute the optimized solutions.
2. **Tesseract** is another library for motion planning and collision checking that implements some very good motion planning algorithms like TrajOpt which has the support of convex-convex collision and Tesseract is made keeping in mind that it should be as lightweight and use as few dependencies as possible. Tesseract is still under Heavy Development and lacks very good community support.

3. **MoveIt!** is one of the most widely adopted Motion Planning, Collision checking, and navigation library. It has one of the best Community supports and we can use any of the available motion planner by simply changing the plugin. MoveIt! has pre-built support for forward and inverse kinematics, motion planning (with constraints), collision avoidance, grasp planning,

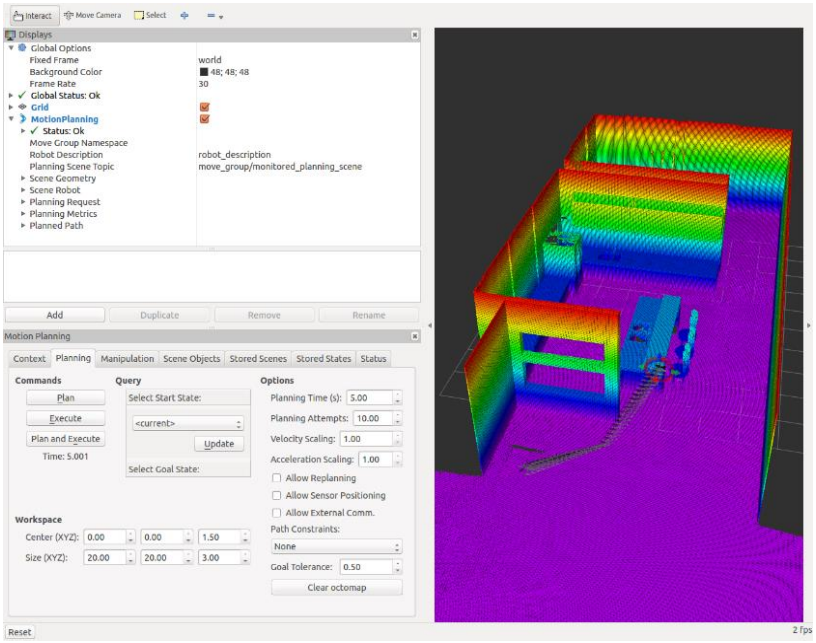


Fig. 2

We will be utilizing and making improvements on the MoveIt Motion Planning Library. Although designed for motion planning of multi-degree-of-freedom robots, MoveIt works well with UAVs. We have looked upon Open Motion Planning Library (OMPL) [44] for implementation of motion planning algorithms, but this area is further to be researched, and it might be needed that we develop our own motion planning algorithm to be used in conjunction with MoveIt!.

For the obstacle avoidance algorithm, there are a myriad of methods being applied. The use algorithms such as Dijkstra, A*, D* & other sampling-based algorithms is well established in path planning of autonomous ground vehicles. These algorithms loop through the sample space to search for valid states and then proceed to search for an optimal path. However, in high dimensional case such as a UAV path planning, these algorithms are not suitable as the enormous sample space requires high computational power: To overcome this, sampling heuristics is added into the algorithm to reduce the amount of computation required. The OMPL Library consists of various Sampling based Planners, with and without heuristics like RRT and its derivatives, PRM and its derivatives, EST, SPARS etc. The list can be found at [45]. Although, OMPL automatically selects the best fit planner for given scenario, we can also force it to use a Specific Planner. Many Research papers on autonomous UAVs have mentioned RRT [46] and its derivatives as their choice, for example [47] and [48].

MoveIt acts as an integrator by connecting OctoMap, Path Planning library and PX4 for perception, path-planning, and motion control respectively. It provides crucial robot constraints for path-planning, processes sensor data for mapping in OctoMap and sends path setpoints to PX4 for flight control.

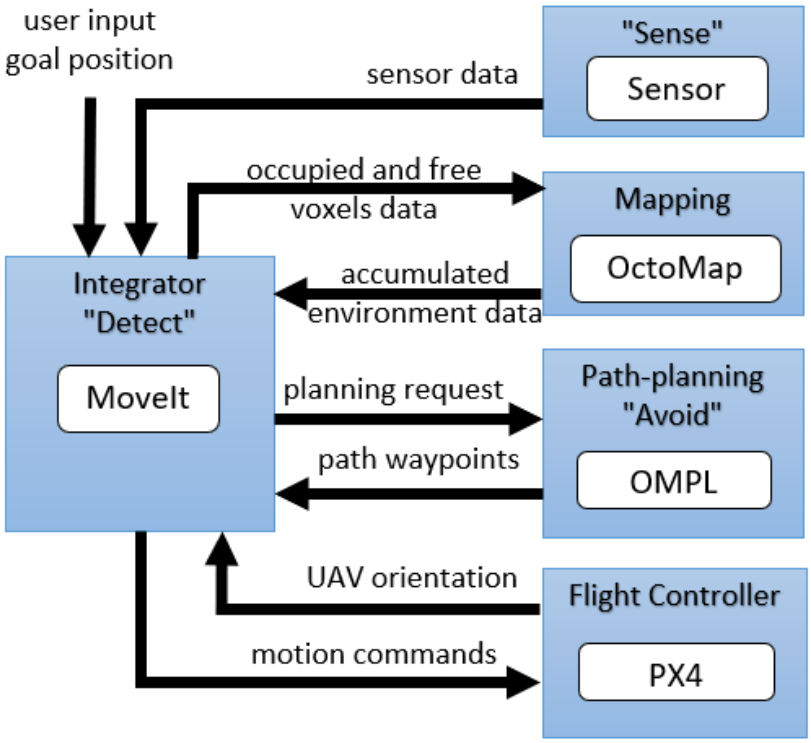


Fig. 3 | Image Credits [49]

We can mix and match different Planning algorithms available in MoveIt! for best results which suit our case. For example, OMPL can be run as a Pre-Processor for CHOMP, wherein CHOMP will get a vague path as its input from OMPL which it will further improvise upon, as suggested in [50] For MoveIt to work with UAVs, MoveIt must be configured such that the UAV is detected as a single-joint, multi-degree-of-freedom robot. Additionally, commands to yaw the UAV during flight must be calculated so that the UAV faces the direction it is moving in.

The Motion Planning Subsystem must take care of five main things:

1. **Global Path Planning**: The user will select a start and end-points in the MoveIt! interface on the pre-generated Octo-Map and then the motion planner should be able to generate a path that should be followed by the quad-rotor, assuming everything to be the same as the old map. The move_group node in MoveIt! will generate the desired trajectory that moves the quadrotor to a desired pose without crashing into any obstacles identified by the sensors or violating the velocity and acceleration constraints of the quadrotor model.
2. **Local Path Planning**: The drone can face any new obstacles in its path and should be able to make decisions based on the data that its on-board sensors are providing it. MoveIt! can do this by directly taking the raw point-cloud data or depth image from the sensors, as suggested in MoveIt! documentation [51]. The Point Cloud Data/Depth Map data needs to be explicitly updated at each iteration of program, and then the validity of our existing path should be checked against the newly generated Octo Maps. If the existing path turns out to be invalid, the Path Planning loop runs again, as done in [49] and explained in Fig. 4.

3. **Behavior Planning**: Different scenarios demand maneuvers of different nature, furthermore, depending on the environment several constraints on the quadcopter's motion and trajectory (for e.g.: max curvature, max angular acceleration, max pitch, etc.) might be relaxed or made more strict. It is the job of behavior planner to decide these set of constraints and prepare an ideal action plan for the quadcopter. Different behaviors will be incorporated as a pre-defined set of maneuvers for different environment scenarios called 'Behaviors'. The behavior planner will relay these behaviors to the local planner based on which the most relevant trajectory will be prepared. The behavior planner can be incorporated as a Rule based priority planner (Sub Sumption architecture), as a FSM (Finite State Machine) [54] or as a set of policies learnt using a Machine learning model. In MoveIt! We can easily set such constraints dynamically under the "User-specified Constraints". This is done using callback functions inside the code. [51]
4. **Collision Checking**: Collision checking is a computationally intensive task and requires proper assumptions and models to be done efficiently. In MoveIt! the CollisionWorld object is used to find collisions inside a planning scene which is using the Flexible Collision Library (FCL) [52] package as a backend. MoveIt! supports collision checking for different types of objects, such as meshes, primitive shapes such as boxes, cylinders, cones, spheres, and OctoMaps. [51]
5. **Sending Actions to Bot**: After motion planning, the generated trajectory talks to the controllers in the robot using the FollowJointTrajectoryAction interface in MoveIt!. This is an action interface in which an action server is run on the robot and move_node of MoveIt! initiates an action client that talks to this server and execute the trajectory on the real robot/Gazebo simulator. MoveIt! does not currently come with the ability to develop multi Degree of Freedom trajectories required to take full advantage of the quadrotor's dynamics. Instead, we will use a modified version of the Moveit_simple_controller_manger to handle the MultiDofFollowJointTrajectoryAction to interface with the Action Server. This action server interfaces with the MoveIt! action client with the MultiDofFollowJointTrajectory.action. AlessioTonioni/Autonomous-Flight-ROS: A simple autopilot for a quadrotor realized using MoveIt!. The system uses a simulated RGBD sensor to reconstruct the map, then OMPL for path generation. [53] This repository shows a standard way how these action files can be configured for a quadrotor. The action server receives the trajectory output from the move_group node that will successfully navigate the quadrotor from its initial pose to the desired pose. This trajectory takes the form of a trajectory_msgs/MultiDOFJointTrajectory message.

Tentative Working Flow Chart [49]

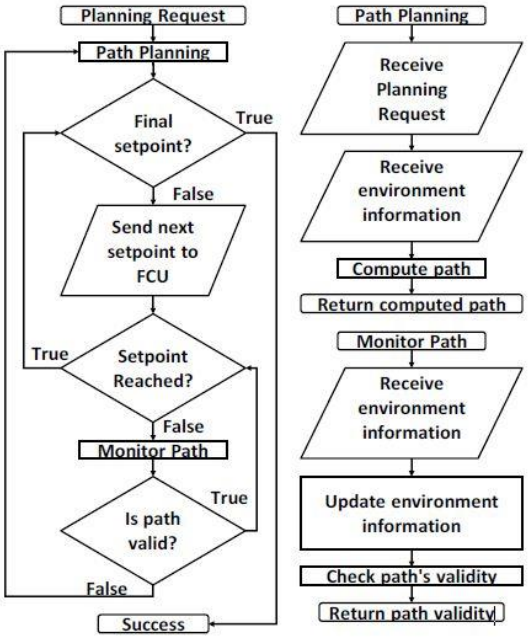


Fig. 4

Link to Simulation video

For testing the MAVROS protocol and its commands, a basic simulation was prepared where control commands were given to Pixhawk SITL (Software in the Loop) firmware to avoid colliding with an array of rectangular frames. Link

Working with MoveIt! [51]

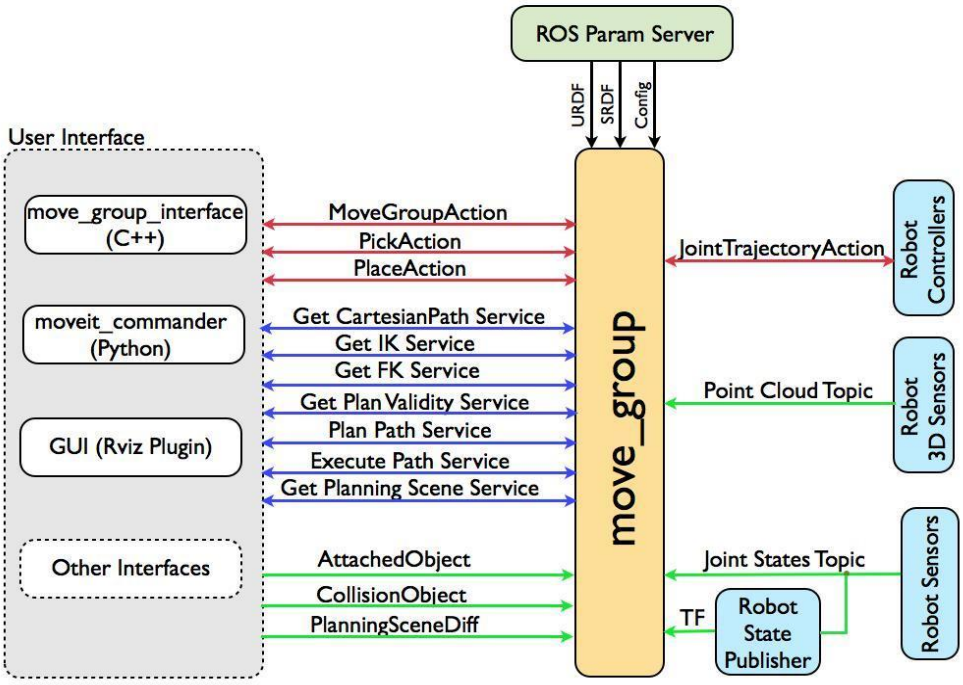


Fig. 5

The primary node used in MoveIt! is the move_group node. As shown in the system architecture image below, the move_group node integrates all of the external nodes in order to provide a set of ROS actions and services for users.

The move_group_interface package provides a C++ interface,
The moveit_commander package supports Python,
and lastly, the Motion Planning Rviz plugin provides a GUI interface.

The move_group node collects robot information such as point cloud, joint state (/joint_states topic) of the robot, and transform (TF) of the robot in the form of topics and services from the State Estimation subsystem.
Lastly, the move_group node maintains a planning scene using the quadrotor's sensors as inputs. The planning scene is representative of the quadrotor's current state and its observed environment. The term "planning scene" is used to represent the world around the robot and store the state of the robot itself. The planning scene monitor inside move_group maintains the planning scene representation.

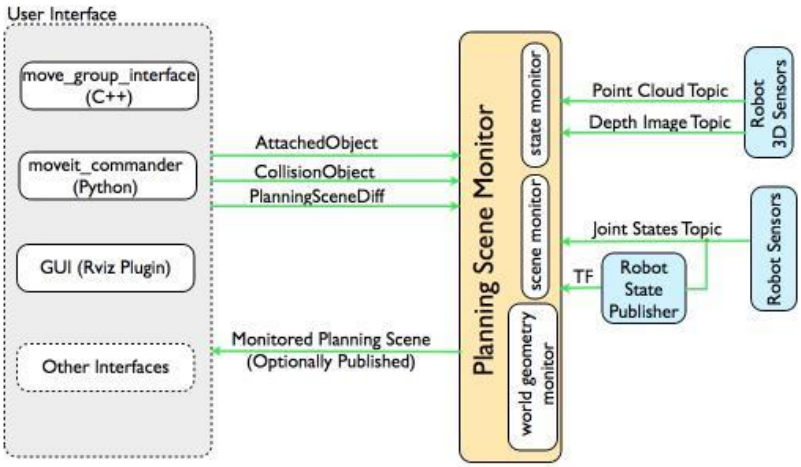


Fig. 6

The move_group node consists of another section called the world geometry monitor, which builds the world geometry from the sensors of the robot and from the user input.
The planning scene monitor reads the joint_states topic from the robot, and the sensor information and world geometry from the world geometry monitor. The world scene monitor reads from the occupancy map monitor, which uses 3D perception to build a 3D representation of the environment in OctoMaps.

Hardware Selection

Onboard Computer:

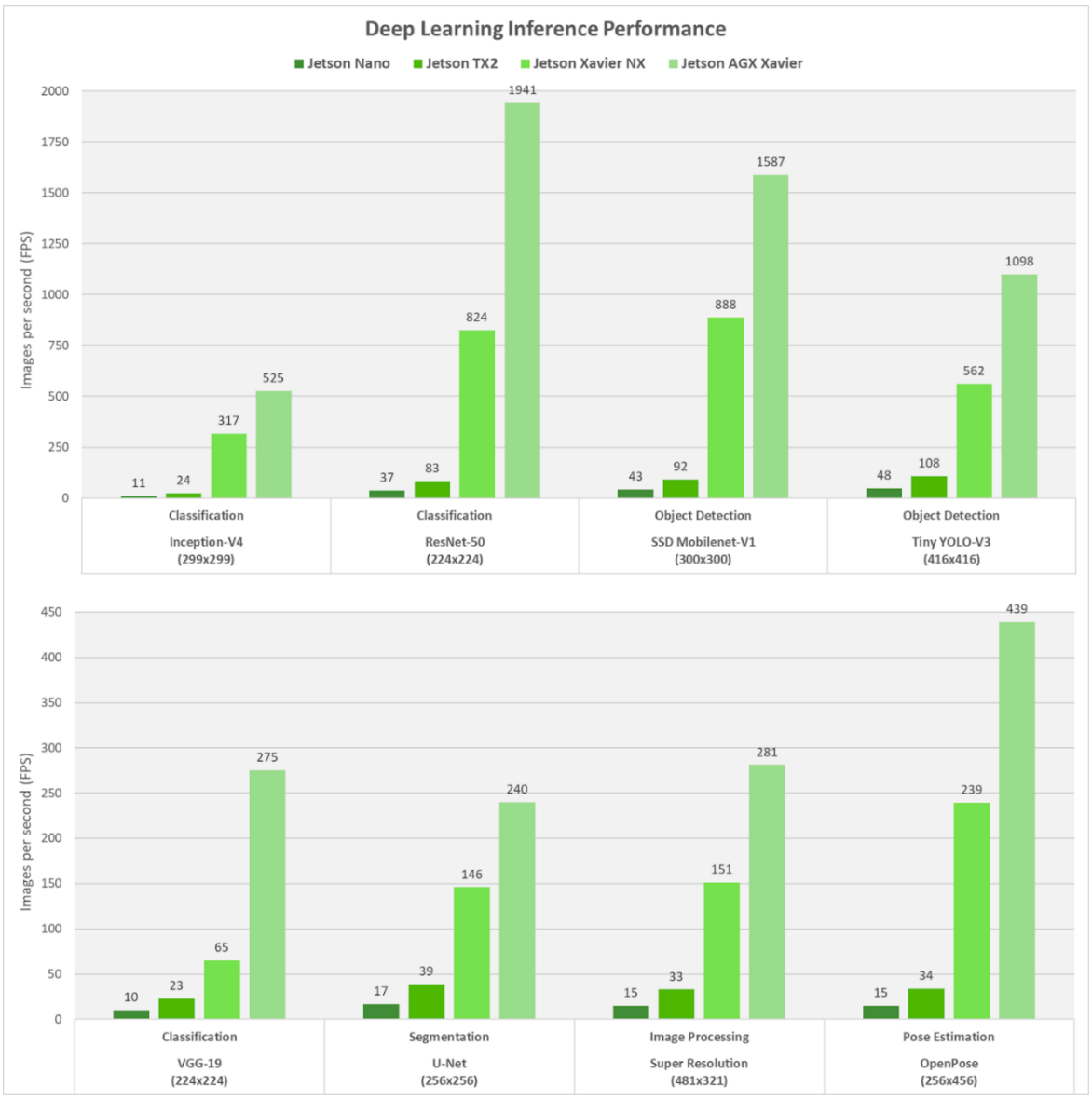
BOARDS	TX2	AGX Xavier 32 GB	AGX Xavier 16 GB	Xavier NX
Price	41,669	90,700	89,974	41,665
Voltage	5.5 V-19.6 V DC	System Voltage Input 9.0V – 20.0V 5V Input: 5.0V	5 V, 9 V~20 V	5V
Net Power	7.5W – 15 W	10W – 30W		10 W/15 W
Dimensions	50 mm x 87 mm	87 mm x 100 mm		69.6 mm x 45 mm
GPU	NVIDIA Pascal architecture with 256 CUDA cores	512-core Volta GPU with 64 Tensor Cores	512-Core Volta GPU with Tensor Cores	384-core NVIDIA Volta™ GPU with 48 Tensor Cores
CPU	Dual-core Denver 2 64-bit CPU and quad-core ARM A57 complex (total 6cores). 864KB L1 + 4MB L2(2MB + 2MB)	8-core Carmel ARM v8.2 64-bit CPU, 8MB L2 + 4MB L3	8-Core ARM v8.2 64-Bit CPU, 8 MB L2 + 4 MB L3	6-core NVIDIA Carmel ARM v8.2 64-bit CPU 6MB L2 + 4MB L3
RAM	8 GB 128-bit LPDDR4 1866MHz - 59.7 GB/s	32GB 256-Bit LPDDR4x | 136.5GB/ss	16 GB 256-Bit LPDDR4x | 137 GB/s	8 GB 128-bit LPDDR4x 51.2GB/s
AI Performance	1.3 TFLOPS* **	32 TOPs		21/14 TOPS (15W/10W)
CSI Camera	Up to 6 cameras (12 via virtual channels) 12 lanes MIPI CSI-2 D-PHY 1.2 (up to 30 Gbps) C-PHY 1.1 (up to 41Gbps)	(16x) CSI-2 Lanes (16x) MIPI CSI-2 lanes (camera header)	(16x) CSI-2 Lanes	Up to 6 cameras (24 via virtual channels) 14 lanes (3x4 or 6x2) MIPI CSI-2 D-PHY 1.2 (up to 30 Gbps)
Vision accelerator	Not available	7-Way VLIW Vision Processor		
Heatsink	Comes with a thermal transfer plate. Also 3rd party available with active cooling	Available, but not officially		No official Heat Sink. 3rd party are available

Active Cooling	3rd party are available.	One kit w/ active cooling system found		No official Active Cooling solution. 3rd party are available
Remarks related to the board		More than 20X the performance and 10X the energy efficiency of the TX2		More than 10X the performance of the TX2. It has been released very recently in March

* 1TFLOP ~ 2TOP

** Intel's UHD 620 for comparison can at max do 384 GFLOPS/

** GTX 1650 = 4 TFLOPS



Frame: A 600mm Carbon Fiber Frame was chosen of HX shape. We ensured that we keep sufficient space for mounting all the Autonomy Stack as well as Quad remains as light weight as possible. Therefore, H4 600mm Reptile Frame looks like a good option.
Motors: Motors were chosen keeping in mind that we needed flight time over 15 minutes, a thrust : weight ratio of at least 2:1, and it was ensured that it is supported by our Frame and the recommended Prop size for our Frame was taken into consideration as well. We zeroed out at two motors BrotherHobby Tornado T5 Pro 640KV (high thrust, low efficiency) and SunnySky V3508 380KV (lower thrust, high efficiency).
Propellers: 12-inch propellers are recommended with the H4 600mm Reptile Frame, and the motors we zeroed out recommended CF propeller with a pitch of 3.8-4.0. Therefore, T-Motors CF 1240 propellers were chosen.
Flight Controller: Pixhawk 4 was selected as it is open source, widely supported across ROS community and has a stable updated Firmware. The other options that were considered were DJI Naza, BetaFlight**,** I-Nav. Pixhawk 4 still seemed to be the best call.
Battery: 8200mAh 6S LiPo was considered for best Flight Time and Weight management. We looked upon 6S batteries ranging from 5200mAh up to 10,000mAh and 8200mAh seemed to be a sweet spot.
Other Drone Parts: Protecting the sensors will be extremely important. Hence, we will have to either buy such structures from a vendor or we will have to prepare a 3D model and perform a thorough stress analysis using proprietary software like ANSYS and Fusion 360. Then we will 3D print these structures using a strong material which will add further costs. Furthermore, we may need to use gimbals for retracting and moving certain parts of the quadcopter depending on the need, for e.g.: attaching a changing orientation camera.
TurtleBot: TurtleBot is the de-facto standard in autonomous ground-bots prototyping due to its low cost, ease of usage and compatibility with ROS. We chose to go with TurtleBot 3 Burger. TurtleBot 3 is the latest in the TurtleBot series, and Burger is the cheapest and most compact model. It would suffice our needs in terms of modularity, physical attributes, and features.
We plan to use the ground bot to test all our modules safely before assembling them on the drone itself.

Cost Estimates

Based on the reasoning provided in the Hardware Selection and Subsystems' sections two models have been prepared for different budget levels depending on the minimum sensing and computation requirements. The two models (packages A and B) differ in precision and computation speed. The LiDAR generated instant point clouds by default while the depth camera generates a depth map which must be converted to a point cloud to perform point cloud localization techniques which adds additional computation. Also, in package A the Vectornav VN-100 is a very high grade IMU and has an inbuilt microprocessor which deploys a Kalman Filter algorithm on the sensors' data and will operate alongside the on-board computer thus saving a lot of computational time. In package B since we removed it to cut costs, we will have to use Pixhawk 4's inbuilt IMU which is not much accurate as compared to VN-100. Further in the latter option the values must be imported through MAVLINK to the onboard computer which will introduce some protocol related communication delays. The data from the 3 IMU sensors should then be processed by the on-board computer which will again increase computational time. The tracking cam's inbuilt IMU and Visual Odometry will be used in package B to make for Pixhawk 4's shortcomings. However, the LiDAR still wins by a big margin because of its high effective accuracy relative to the point

cloud size. LiDAR point clouds are denser and more accurate than the coded light point clouds generated by Intel Realsense d435. Hence, package A offers a lot more precision and accuracy than package B. Nvidia Xavier NGX and TX2 have comparable computational power. But since the TX2 is widely used by the community around the world is well documented for Robotics applications it certainly gives an advantage, because well documented hardware ensures smooth workflow and easy debugging.

Hence, using the Package A is recommended to attain L4 autonomy. It enables real time operation of fast-moving autonomous systems, especially UAVs.

Package A

Component	Package A Unit	Weight (grams)	Cost^ (INR)	Power Rating (Watts)
Frame*	H4 600mm Reptile CF	626	15000	0
Motors*	Brother Hobby Tornado T5 Pro 640 KV	388	40000	528
Flight Controller	Pixhawk 4	16	17500	2.5
Propellers*	T-Motors CF 1240	150	25000	0
Depth Camera / Stereo Camera*	ZED2	124	66000	1.9
LIDAR/Tracking Camera	Velodyne Puck LITE	600	600000	8
IMU	Vectornav - VN 100	20	72000	0.2
On-Board Computer	Nvidia Xavier NX	467	45000	7.5
Battery*	8200mAh (LiPo)	1200	50000	0
Tx/Rx, other drone parts (Gimbals, Sensor Protection, Motor Protectors, 3D Printing, ESC)		300	100000	0.5
Ground Bot	TurtleBot 3 - Burger	NA	60000	0
Total		**3891**	**10,90,500**	**548.6 Flight Time: 19-22 min**

Package B

Component	Package B Unit	Weight (grams)	Cost^ (INR)	Power Rating (Watts)
Frame*	H4 600mm Reptile CF	626	12000	0
Motors*	SunnySky V3508 - 380KV	436	31000	420
Flight Controller	Pixhawk 4	16	17500	2.5
Propellers*	T-Motors CF 1240	150	13000	0
Depth Camera / Stereo Camera*	D435	72	52000	2
LIDAR/Tracking Camera*	T265	60	52000	1.5
IMU (Spare included)	None	0	0	2
On-Board Computer*	Nvidia Jetson Xavier NX	600	80000	10
Battery*	8200mAh (LiPo)	1200	20000	0
(Tx/Rx, other drone parts (Gimbals, Sensor Protection, Motor Protectors, 3D Printing, ESC) + Proprietary Software (ANSYS, Eagle CAD), Bench Supply)		300	250000	
Ground Bot	TurtleBot 3 - Burger	-	60000	-
Total		**3460**	**5,35,500**	**438.5** *Flight Time: 26 min*

** The quoted cost also includes spares. These spares come in handy in the development & experimentation phase. The prices estimates can be slashed in half for the respective items and the cost per unit can hence be calculated.*

^ Quoted cost is in INR without import duties and shipping charges.

References

1. Kalman, R. (1960) A New Approach to Linear Filtering and Prediction Problems. ASME Journal of Basic Engineering, 82, 35-45. http://dx.doi.org/10.1115/1.3662552
2. G. Mao, S. Drake and B. D. O. Anderson, "Design of an Extended Kalman Filter for UAV Localization," *2007 Information, Decision and Control*, Adelaide, Qld., 2007, pp. 224-229, doi: 10.1109/IDC.2007.374554.
3. Wan, Eric & Merwe, Ronell. (2000). The Unscented Kalman Filter for Nonlinear Estimation. The Unscented Kalman Filter for Nonlinear Estimation. 153-158. 153 - 158. 10.1109/ASSPCC.2000.882463.
4. S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," in *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401-422, March 2004, doi: 10.1109/JPROC.2003.823141.
5. Braga, José & Shiguemori, Elcio & Campos Velho, Haroldo. (2018). Determining the trajectory of Unmanned Aerial Vehicles by a novel approach for the Particle Filter.
6. Gustafsson, Fredrik & Gunnarsson, Fredrik & Bergman, Niclas & Forssell, Urban & Jansson, Jonas & Karlsson, Rickard & Nordlund, Per-johan. (2002). Particle Filters for Positioning, Navigation and Tracking. Signal Processing, IEEE Transactions on. 50. 425 - 437. 10.1109/78.978396.
7. G. G. Rigatos, "Sensor fusion for UAV navigation based on Derivative-free nonlinear Kalman Filtering," *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Guangzhou, 2012, pp. 890-895, doi: 10.1109/ROBIO.2012.6491081.
8. Pornsarayouth, Sirichai & Yamakita, Masaki. (2011). Ensemble Kalman filter for multisensor fusion with multistep delayed measurements. 1 - 10. 10.1109/AERO.2011.5747428.
9. Pessanha Santos, Nuno & Melicio, Fernando & Lobo, Victor & Bernardino, Alexandre. (2015). A Ground-based Vision System for UAV Pose Estimation. International Journal of Robotics and Mechatronics. 1. 138-144. 10.21535/ijrm.v1i4.180.
10. Pacheco da Luz, Eduardo & Becceneri, José & Campos Velho, Haroldo. (2008). A new multi-particle collision algorithm for optimization in a high performance environment. Journal of Computational Interdisciplinary Sciences. 1. 10.6062/jcis.2008.01.01.0001.
11. Du, Hao & Wang, Wei & Xu, Chaowen & Xiao, Ran & Sun, Changyin. (2020). Real-Time Onboard 3D State Estimation of an Unmanned Aerial Vehicle in Multi-Environments Using Multi-Sensor Data Fusion. Sensors. 20. 919. 10.3390/s20030919.
12. "The Unscented Kalman Filter." *Optimal State Estimation*, 2006, pp. 433–459., doi:10.1002/0470045345.ch14.
13. Simon, Dan. Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches. John Wiley & Sons, 2006.
14. Chen, Yang; Gerard Medioni (1991). "Object modelling by registration of multiple range images". Image Vision Comput. doi:10.1016/0262-8856(92)90066-C
15. Besl, Paul J.; N.D. McKay (1992). "A Method for Registration of 3-D Shapes". IEEE Transactions on Pattern Analysis and Machine Intelligence. doi:10.1109/34.121791
16. Zhen, Weikun, et al. "A Joint Optimization Approach of LiDAR-Camera Fusion for Accurate Dense 3-D Reconstructions." *IEEE Robotics and Automation Letters*, vol. 4, no. 4, 2019, pp. 3585–3592., doi:10.1109/lra.2019.2928261.
17. Andrea Censi , 2008 IEEE International Conference on Robotics and Automation, An ICP variant using a point-to-line metric
18. A.L. Pavlov, G.V. Ovchinnikov, D. Yu. Derbyshev, D. Tsetserukou4and I.V. Oseledets, AA-ICP: Iterative Closest Point with Anderson Acceleration, 2017
19. Han, Jian & Liao, Yaping & Zhang, Junyou & Wang, Shufeng & Li, Sixian. (2018). Target Fusion Detection of LiDAR and Camera Based on the Improved YOLO Algorithm. Mathematics. 6. 213. 10.3390/math6100213.
20. Lowe, David G. (1999). Object recognition from local scale-invariant features. Proceedings of the International Conference on Computer Vision

21. Herbert Bay, Tinne Tuytelaars, and Luc Van Gool (2006), SURF: Speeded Up Robust Features
22. Rublee, Ethan; Rabaud, Vincent; Konolige, Kurt; Bradski, Gary (2011). ORB: an efficient alternative to SIFT or SURF. IEEE International Conference on Computer Vision (ICCV).
23. Olivier Faugeras, 1993, MIT Press, Three-dimensional computer vision: a geometric viewpoint
24. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).
25. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham. https://arxiv.org/abs/1512.02325
26. Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2018). Focal loss for dense object detection. IEEE transactions on pattern analysis and machine intelligence. (State of the art)
27. Qi, C. R., Liu, W., Wu, C., Su, H., & Guibas, L. J. (2017). Frustum point nets for 3d object detection from rgb-d data. arXiv preprint arXiv:1711.08488. (3D object detection from 2D)
28. Shuran Song and Jianxiong Xiao, 2015, Sliding Shapes for 3D Object Detection in Depth Images, Princeton University
29. 3D Object Pose Estimation, Nvidia docs
30. Ye Wang, 2016, 3D Object Detection in RGB--D Images
31. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., ... & Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3213-3223). (For understanding the problem + the metrics)
32. Neuhold, G., Ollmann, T., Bulò, S. R., & Kontschieder, P. (2017, October). The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes. In ICCV (pp. 5000-5009).
33. Badrinarayanan, V., Kendall, A., & Cipolla, R. (2015). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. arXiv preprint arXiv:1511.00561.
34. Zhao, H., Shi, J., Qi, X., Wang, X., & Jia, J. (2017, July). Pyramid scene parsing network. In IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
35. Hornung A, Wurm K.H., Bennewitz M., Stachniss C.,& Burgard W. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees.
36. Roth-Tabak Y, Jain R (1989) Building an environment model using depth information. Computer 22(6)
37. Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media. (PDF available online: http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf)
38. Forsyth, D.A. and J. Ponce (2003). *Computer Vision: a modern approach* (2nd edition). New Jersey: Pearson.
39. Wurm, Kai M., et al. "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems." *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*. Vol. 2. 2010.
40. Guimarães, Rodrigo Longhi, et al. "ROS navigation: Concepts and tutorial." *Robot Operating System (ROS)*. Springer, Cham, 2016. 121-160.
41. Merkt, Wolfgang, Vladimir Ivan, and Sethu Vijayakumar. "Continuous-time collision avoidance for trajectory optimization in dynamic environments." *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.
42. I. A. Sucan and S. Chitta, "MoveIt!," [Online]. Available: http://moveit.ros.org.
43. Anonymous answer, “ROS Navigation Stack vs OMPL”, [Online]. Available: https://answers.ros.org/question/131801/ros-navigation-stack-vs-ompl/?answer=132166#post-id-132166. Accessed, Sept. 2020.
44. I. A. Șucan, M. Moll and L. E. Kavraki, "The Open Motion Planning Library," IEEE Robotics & Automation Magazine, vol. 19, no. 4, p. 72–82, December 2012.
45. Kavraki Lab, Department of Computer Science, Rice University, “Available Planners”, [Online]. Available: http://ompl.kavrakilab.org/planners.html. Accessed, Sept. 2020.

46. S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1999.
47. Aguilar, Wilbert G., et al. "RRT* GL based optimal path planning for real-time navigation of UAVs." *International Work-Conference on Artificial Neural Networks*. Springer, Cham, 2017.
48. Y. Dong, C. Fu and E. Kayacan, "RRT-based 3D path planning for formation landing of quadrotor UAVs," in 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, Thailand, 13-15 Nov, 2016.
49. Lim, Catrina & Li, Boyang & Ng, Ee & Liu, Xin & Low, Kin. (2019). Three-dimensional (3D) Dynamic Obstacle Perception in a Detect-and-Avoid Framework for Unmanned Aerial Vehicles. 996-1004. 10.1109/ICUAS.2019.8797844.
50. Chitta, Sachin. "MoveIt!: an introduction." *Robot Operating System (ROS)*. Springer, Cham, 2016. 3-27.
51. MoveIt!, MoveIt! Official Documentation, https://moveit.ros.org/documentation/concepts/. Accessed, Sept. 2020.
52. Pan, Jia, Sachin Chitta, and Dinesh Manocha. "FCL: A general purpose library for collision and proximity queries." *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012.
53. Al. Tonioni. 2015. Autonomous-Flight-ROS: A simple autopilot for a quadrotor realized using MoveIt!. The system uses a simulated RGBD sensor to reconstruct the map, then OMPL for path generation. https://github.com/AlessioTonioni/Autonomous-Flight-ROS. Github, Nov. 2015
54. Tomić, Teodor & Schmid, Korbinian & Lutz, Philipp & Dömel, Andreas & Kassecker, Michael & Mair, Elmar & Grixa, Iris & Ruess, Felix & Suppa, Michael & Burschka, Darius. (2012). Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue. Robotics Automation Magazine, IEEE. 19. 46 -56. 10.1109/MRA.2012.2206473.
55. Lim, Catrina & Li, Boyang & Ng, Ee & Liu, Xin & Low, Kin. (2019). Three-dimensional (3D) Dynamic Obstacle Perception in a Detect-and-Avoid Framework for Unmanned Aerial Vehicles. 996-1004. 10.1109/ICUAS.2019.8797844.
56. Sanchez-Lopez, J.L., Wang, M., Olivares-Mendez, M.A. *et al.* A Real-Time 3D Path Planning Solution for Collision-Free Navigation of Multirotor Aerial Robots in Dynamic Environments. *J Intell Robot Syst* 93, 33–53 (2019). https://doi.org/10.1007/s10846-018-0809-5
57. Perez-Grau, FJ, Ragel, R, Caballero, F, Viguria, A, Ollero, A. An architecture for robust UAV navigation in GPS-denied areas. *J Field Robotics*. 2018; 35: 121-145. https://doi.org/10.1002/rob.21757
58. Autonomous Navigation of Generic Monocular Quadcopter in Natural Environment. Kumar Bipin, Vishakh Duggal and K. Madhava Krishna *ICRA 2015*