System Design Report

Group: 2

Devrat Singh (981024-T134)

System Diagram

The following diagram shown in Figure 1, demonstrates how each of the components in the system interact for the drone to fly continuously.

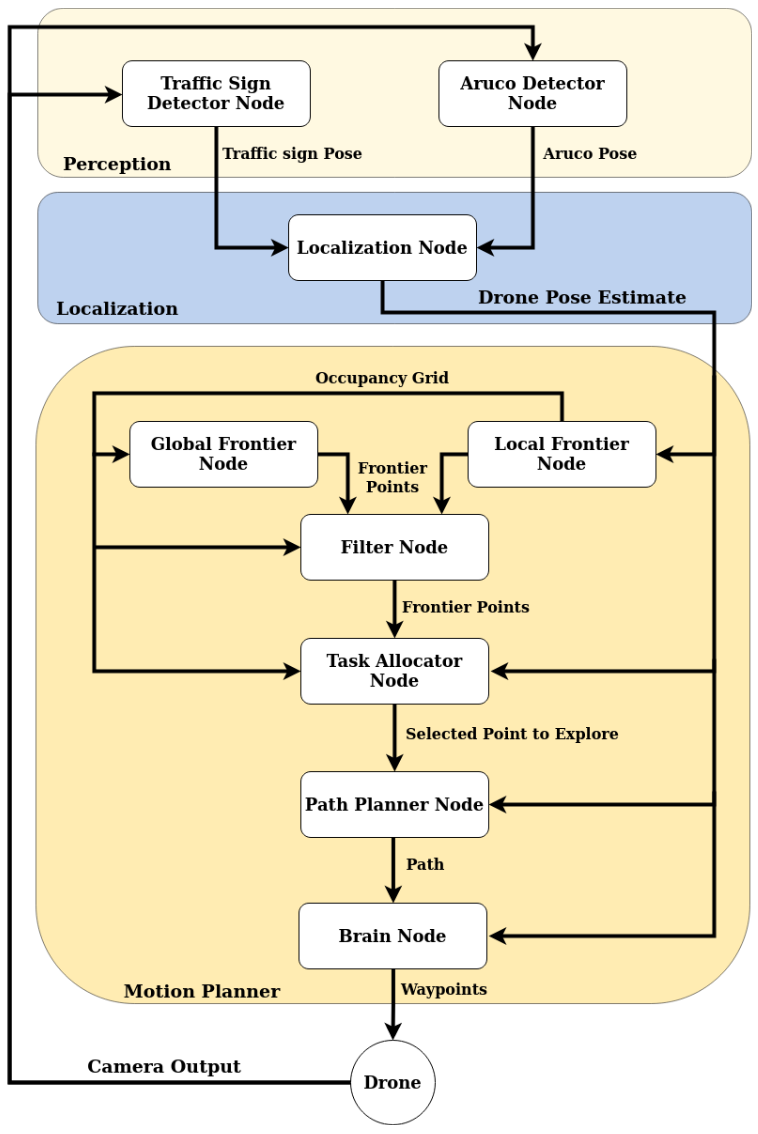

Figure 1: System Diagram

Perception

Perception is a crucial component for our system. As can be seen in Figure 1, it consists of the *Traffic sign detector node* and the *Aruco dectector node.* Moreover, the Aruco perception component was provided as a built-in functionality, hence this section will mainly discuss the traffic sign detection.

Our approach to finding the pose of traffic signs, involves Deep Neural Networks (DNNs) and other image analysis techniques such as SIFT and RANSAC. Considering the DNNs, we initially utilized MobileNet. However, we encountered that with this approach the detection and classification of traffic signs was not fast enough to assist in localization. As such, we shifted to the use of SqueezeNet, which actually sped up the process by approximately 4 times. For training the network, the provided dataset of images was combined with the images we took and the ones shared by the other groups. Moreover, in the end, the images given by other groups were not utilized. One of reasons being difference in labelling and having no improvement in performance with them.

Apart from the layers hidden in SqueezeNet, we had 3 additional layers. The input to the network is an image and the output is a $39 \times 29 \times 20$ tensor. This means, for each grid cell we have (x, y, w, h) and the probabilities for each traffic sign class. In the end, we get a bounding box for the detected sign and the class prediction. Going on a tangent, on starting this detector node, the SIFT (Scale-Invariant Feature Transform) algorithm is run for the traffic sign classes and the output is stored. The SIFT output for the region inside the bounding box (from DNN) is then compared to the one calculated initially, for the predicted class. This allows us to do feature matching and remove some outliers. Following this, we use the Brute-Force matcher to further help the matching process. Finally, using RANSAC, we are able to get the transform from camera_link to the sign. It is worth mentioning that we also tried using ORB for faster 3D pose estimation, however, SIFT provided faster and accurate results.

Localization

The need for localization is solidified by the fact that none of the planning techniques can be applied if the drone is unaware of its position in the map. Accordingly, we had the option of either using a Kalman Filter or a Patricle Filter to get good drone pose estimates. We opted to use the Kalman Filter, the reason being that the system we have is linear and also that it allows for a less computationally heavy approach. This turned out to be a good choice, because both the perception and the planning components were computationally intensive. In terms of the actual implementation, the localization process is best encapsulated by the diagram in Figure 2.

An aspect that is crucial to the discussion of our localization system, is noise models. Initially, we started with experimentally found static noise models for both process and measurement noises. With the inclusion of traffic signs in the pose estimation process, we opted to go for separate noise models for Aruco markers and traffic signs. As it turned out, the pose estimates for the traffic signs is comparatively worse than the Aruco markers, thus the noise models for the traffic signs were made to represent this higher uncertainty. On top of this, we are also doing *adaptive noise adjustment.* It is done such that, at each step of the Kalman filter, we use the innovation to adaptively adjust the process noise R and the residual to adjust the measurement noise Q. The adaptive noise adjustment is implemented based on [1].

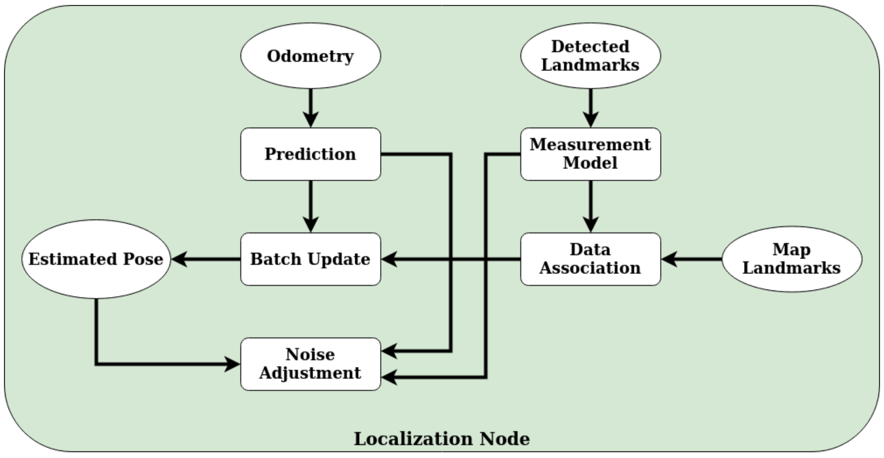

Figure 2: Localization Process

Finally, *data association* is a detail that needs to be mentioned to make this discussion complete. This becomes unavoidable, when all the markers in the map are of the same id. Therefore, we utilized maximum likelihood association with Mahalanobis distance to find the most likely measurement. Apart from that, we also discarded any measurements that were too far away.

Planning

As it can be seen in Figure 1, planning is the component responsible for the motion of the drone. This section aims on exploring the implementation of the planning process influenced by *et al.* [2]:

Gridmap

The exploration process is heavily dependent on the generated occupancy grid. It is initialized with unexplored and inflated regions, which is then updated as the drone explores the environment by visiting *Frontier points*. In context of this implementation, the frontier points are those that lie in the unexplored regions. The *Local frontier node* is responsible for updating the gridmap by marking the cells within a certain radius around the current drone location as explored.

Local and Global Frontier Nodes

Based on RRT, both of these nodes generate explorable frontier points. The *Local frontier node*, works such that it builds an RRT tree from the drone location and as soon as it finds a point belonging to an unexplored region, it sends it to the *Filter node*. The *Global frontier node*, also performs a similar procedure. Moreover, a key difference between these two is that the local frontier node resets its tree every time it finds a new frontier point, and restarts again from the current robot location. On the other hand, the global node keeps on growing from the start position of drone and never resets. The need for having these two separate point generators is justified by the reasoning that the local node increases the chance of finding points in an unexplored space, as it starts from the drone location. The global node, on the other hand ensures that small corners are not left unvisited. The local node also helps in providing points when the global node becomes slow due to continuous growth of the tree since the start.

Filter Node

The points that are received from the *Local* and *Global Frontier Nodes*, can have multiple closely spaced points. Therefore, to avoid computations on similar points, we use mean shift clustering to cluster the obtained frontier points and just keep the center of these clusters. After obtaining the cluster centers, of these that lie either in an already explored region or inflated space are discarded. Now, the *Task Allocator node* can choose from these select few points.

Task Allocator Node

This node can actually be considered the brain of the system, as it makes the decisions which help in exploration. It has continuous stream of explorable points from the *Filter node*, which are then processed to choose the next region to explore. For all the points it receives, a revenue/score is calculated, this is the combination of information gain and the cost to reach that point. To simplify the problem, the cost is the straight line distance between the drone and the frontier point. On the other hand, the information gain is the area of the unknown region which will be explored if this point is chosen. In our implementation, the information gain is given more importance (through higher weights). Subsequently, after computing the revenue for all points, the one with the highest revenue is chosen and given to the *Path planner*.

Planner Node

This node is responsible for planning an obstacle free path, given the final destination and the current location of the drone. When considering the problem of path planning, there were various techniques available, however, the major contenders were A* and RRT. We opted to implement RRT, given that it is a sample based method, as opposed to A*, a grid based method. Being sample based, allowed for an easier shift from 2D to 3D planning. On the contrary, A* would have required setting up an OctoMap to plan paths in 3D. Therefore, considering the computation costs and the convenience of upgrading, RRT was the chosen method.

Without going too much into details, the implemented RRT planner builds a random tree starting from the location of the drone. The expansion of the tree is stopped, once one of the branches' endpoint is close to the final point. The preliminary path from RRT, is then smoothened. At each of the points on the path, the yaw of the drone is decided such that the drone is facing towards the nearest marker. Finally, the path with yaw set, is published for the *Brain node* to implement.

Brain Node

This node is mainly responsible for publishing the planned path points, and making sure that the drone follows the given path. It receives the path from the *Planner node* and in a piecemeal fashion publishes the setpoints. It also keeps track of the robot state i.e. moving or stationary, which is utilized by other processes such as *Localization* and *Task allocator*.

References

[1] Shahrokh Akhlaghi, Ning Zhou, and Zhenyu Huang. "Adaptive adjustment of noise covariance in Kalman filter for dynamic state estimation." In: *2017 IEEE Power Energy Society General Meeting*. 2017, pp. 1–5. DOI: 10.1109/PESGM.2017.8273755.

[2] Hassan Umari and Shayok Mukhopadhyay. “Autonomous robotic exploration based on multiple rapidly-exploring randomized trees.” In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 1396–1402. DOI: `10.1109/IROS.2017.8202319`.