INTER IIT TECH MEET'21

IIT Guwahati

DRDO DGRE's Vision Based Obstacle Avoidance Drone

Project Documentation

Team Number 16

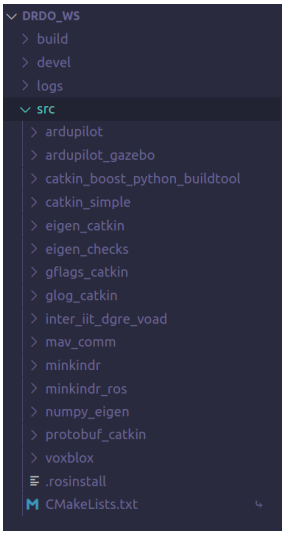
24 March 2021

Contents

1 Installation and Setup Instructions

1.1 Pre-Installation Assumptions

The following instructions assume that the system has a working installation of Ubuntu 18.04 with ROS Melodic and Gazebo 9. Further, the Ardupilot Stack and the ardupilot_gazebo have also been properly installed. We will be using the **catkin build** system instead of catkin_make. The following workspace structure was maintained during development:
Instructions followed for installing Ardupilot Stack and ardupilot_gazebo plugins are reproduced below for verification:




```
$ mkdir -p ~/drdo_ws/src
$ cd ~/drdo_ws/src
$ git clone https://github.com/Ardupilot/ardupilot
$ cd ardupilot
$ git submodule update --init --recursive
```

```
$ Tools/environment_install/install-prereqs-ubuntu.sh -y
$ sudo apt-get install genromfs
```

To avoid having to restart the system we move the changes made by the script in `~/.profile` to `~/.bashrc` and verify the paths mentioned by it. Further we remove the following line from the bashrc file since it is not available on the branch:

```
$ source $HOME/drdo_ws/src/ardupilot/Tools/completion/completion.bash
```

Next, we shift to the recommended firmware branch - 3.6 and build it. Within the ardupilot directory,

```
$ source ~/.bashrc
$ git checkout Copter-3.6
$ git submodule sync
$ git submodule update --init --recursive
$ ./waf configure --board px4-v3
$ ./waf copter
```

Next, we install the ardupilot_gazebo package as follows:

```
$ cd ~/drdo_ws/src
$ git clone https://github.com/khancyr/ardupilot_gazebo.git
$ cd ardupilot_gazebo
$ git checkout dev
$ mkdir build
$ cd build
$ cmake ..
$ make -j4
$ sudo make install
```

If mavros is not installed, it can be installed as follows:

```
$ sudo apt-get install ros-melodic-mavros
$ ros-melodic-mavros-extras
$ cd /opt/ros/melodic/lib/mavros
$ ./install_geographiclib_datasets.sh
```

1.2 Installing Dependencies

First, install the following additional system dependencies:

```
$ sudo apt-get install python-wstool python-catkin-tools \
$ ros-melodic-cmake-modules protobuf-compiler autoconf \
$ libboost-dev libeigen3-dev libgoogle-glog-dev

$ pip install protobuf
```

Next, we create a new catkin workspace, if not already done:

```
$ mkdir -p ~/drdo_ws/src
$ cd ~/drdo_ws
$ catkin init
$ catkin config --extend /opt/ros/melodic
$ catkin config --cmake-args -DCMAKE_BUILD_TYPE=Release
```

Note: We installed the Ardupilot stack and plugins within the same workspace, however this is not necessary and will not affect the working of the submission (provided the paths of the installation are properly set where required) since these are not built with catkin.

To ensure the neither folder interferes with the catkin build process, please add a file named CATKIN_IGNORE within the two folders:

```
$ cd ~/drdo_ws/src
$ cd ardupilot
$ touch CATKIN_IGNORE
$ cd ../ardupilot_gazebo
$ touch CATKIN_IGNORE
```

Place the provided package folder (i.e. the submission) into the src folder. We will install external dependencies using wstool. The libraries are downloaded from GitHub via SSH, hence please ensure that SSH keys have been setup on the system by following this link.

```
$ cd ~/drdo_ws/src
$ wstool init . inter_iit_dgre_voad/install_ssh.rosinstall
$ wstool update
```

If in case wstool says it has already been initialized, just replace `wstool init` with `wstool merge -t` in the above instructions

The submission depends on the following ROS packages. Use the following command to ensure all of them are installed. Most of these will already be installed, however all packages have been mentioned here just to be sure.

```
$ sudo apt-get install ros-melodic-std-msgs ros-melodic-std-srvs \
ros-melodic-geometry-msgs ros-melodic-mavros-msgs ros-melodic-nav-msgs \
ros-melodic-message-generation ros-melodic-message-runtime \
ros-melodic-tf2-ros ros-melodic-cv-bridge ros-melodic-tf \
ros-melodic-tf-conversions ros-melodic-eigen-conversions \
ros-melodic-sensor-msgs ros-melodic-image-transport \
ros-melodic-cv-bridge ros-melodic-visualization-msgs \
ros-melodic-pcl-ros ros-melodic-pcl-conversions \
ros-melodic-interactive-markers ros-melodic-trajectory-msgs
```

Once all dependencies have been satisifed, build the workspace:

```
cd ~/drdo_ws/src
catkin build
```

Note: In case there are any installations issues, please use the contact details mentioned in the README.md file provided with the package.

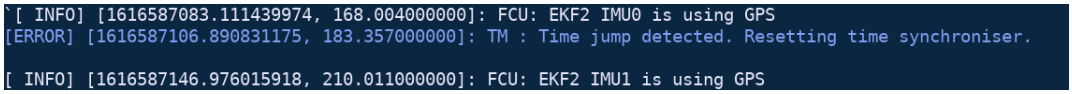
1.3 Running the Simulation

After ensuring that all 27 packages in the workspace have been built successfully (with non-fatal warnings), run the simulation using the following two launch files:

- `roslaunch explorer default.launch world_name:=testing_world1`

  (By changing the world_name argument(testing_world1/testing_world2/testing_world3) you can change the world to be tested.)

- Wait for GPS lock(it should take about 1-2 min for this)



- `roslaunch explorer explorer_node.launch`

  This launch file will launch the explorer node and rviz.

Now the drone will autonomously explore and land on the specified marker.

NOTE: Our algorithms and simulation code are according to the given orientation of camera and quad models (given on 23 March 11:30PM). So for running the simulation, we request the judges not to change any model orientations while evaluation, if at all possible. Kindly use the copy of the provided worlds and models to run the simulation as those are in sync with our testing configuration.

An RViz visualization file has also been provided. This is just for debugging and obtaining additional information and will increase computation considerably. This is disabled by default since it is not relevant for evaluation. To enable this:

- Uncomment the rviz node line in explorer_node.launch
- Set visualize_frontier to true in active_planner/cfg/frontier_params.yaml to visualize frontier generation related topics
- Set visualize_planner to true in active_planner/cfg/params.yaml to visualize pathfinder related topics
- Set visualize_path to true in active_planner/cfg/params.yaml to visualize pathfinder related topics

- Set publish_pointclouds to true in active_planner/cfg/voxblox_params.yaml to visualize ESDF/TSDF map related topics

Set verbose_planner to true in active_planner/cfg/params.yaml to disable terminal output. It is set to true by default to enable the user to understand what the planner is currently upto.

1.4 Version Information for Dependencies

ROS Version: melodic 1.14.10

Gazebo Version: 9.16.0

Pip Version : 20.3.4

Pip3 Version : 21.0.1

Kernel Version: 5.4.0-67-generic

Commit IDs for git installed dependencies: ArduPilot : Copter-3.6
All other repositories use the **main** or **master** branch.

Versions for additional system installed dependencies:

python-wstool	: 0.1.17-1
python-catkin-tools	: 0.6.1-1
ros-melodic-cmake-modules	: 0.4.2-0bionic.20201015.021712
protobuf-compiler	: 3.0.0-9.1ubuntu1
autoconf	: 2.69-11
libboost-dev:amd64	: 1.65.1.0ubuntu1
libeigen3-dev	: 3.3.4-4
libgoogle-glog-dev	: 0.3.5-1

The following dependency versions are the default versions.
Version for ROS binary installed dependencies:

ros-melodic-mavros-msgs	: 0.5.12-0bionic.20201017.034204
ros-melodic-std-srvs	: 1.11.2-0bionic.20201017.034224
ros-melodic-geometry-msgs	: 1.12.8-1bionic.20210112.173042
ros-melodic-nav-msgs	: 1.12.8-1bionic.20210112.173737
ros-melodic-message-generation	: 0.4.1-1bionic.20201017.033327
ros-melodic-message-runtime	: 0.4.12-0bionic.20201017.033232
ros-melodic-tf2-ros	: 0.6.5-0bionic.20210112.183245
ros-melodic-cv-bridge	: 1.13.0-0bionic.20210112.181516

ros-melodic-tf	: 1.12.1-1bionic.20210112.183814
ros-melodic-tf-conversions	: 1.12.1-1bionic.20210112.190235
ros-melodic-eigen-conversions	: 1.12.1-1bionic.20210112.180039
ros-melodic-sensor-msgs	: 1.12.8-1bionic.20210112.173755
ros-melodic-image-transport	: 1.11.13-0bionic.20210112.181401
ros-melodic-visualization-msgs	: 1.12.8-1bionic.20210112.173714
ros-melodic-pcl-ros	: 1.7.1-1bionic.20210112.185053
ros-melodic-pcl-conversions	: 1.7.1-1bionic.20210112.181845
ros-melodic-interactive-markers	: 1.11.5-1bionic.20210112.190840
ros-melodic-trajectory-msgs	: 1.12.8-1bionic.20210112.180155

1.5 Changes in provided Model Files

Following changes in the file drone_with_depth_camera/model.sdf:

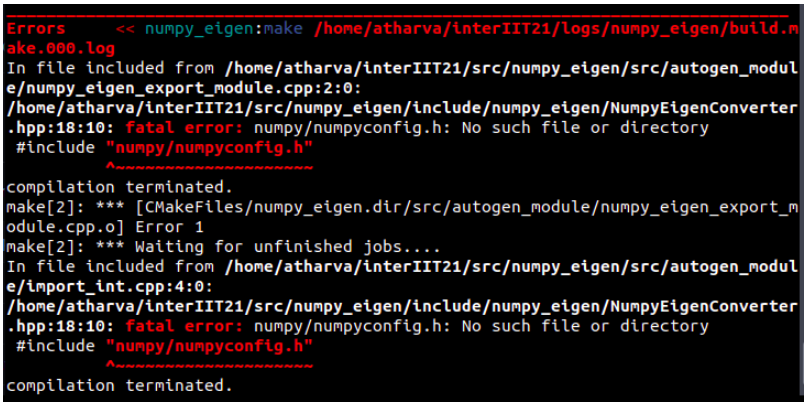
- line 43: **update_rate** tag of depth_camera sensor changed from **10.0 to 20.0**
- line 47: **horizontal_fov** tag changed from **1.047198 to 2** : We increase the horizontal fov of our depth camera to receive more points or simply to get better maps
- line 54: **near** tag under depth_camera sensor changed from **0.1 to 0.01** : done to augment the pointcloud to get nearer points as well
- line 59: **baseline** tag in depth_controller plugin changed from **0.2 to 0.11**
- line 71: **pointCloudCutoffMax** tag changed from **3.0 to 10.0** : increased range of the point-cloud which allows us to plan farther and faster

1.6 Possible Fixes for Potential Build Issues

- ArduPilot does not build completely. We can use the following command instead of "catkin build". Note that building ArduPilot with catkin is absolutely unnecessary since it has its own independent build system.

  ```
  $ catkin build --continue-on-failure
  ```

- Error in building numpy_eigen



To solve this copy the contents of the file: add_python_export_library.cmake to that same file ie. catkin_boost_python_buildtool/catkin_boost_python_buildtool/cmake/add_python_export_library.cmake, then try building it again.

- No downward camera topic
  Try deleting gimbal_small_2d model in $\sim$ /.*gazebo*/*models*

2 Overall Approach and Algorithm Description

2.1 Autonomous Takeoff and Landing

We have utilized **ROS Services** and **Service Clients** for takeoff and landing. For executing the same the following services are being called sequentially:

For takeoff :

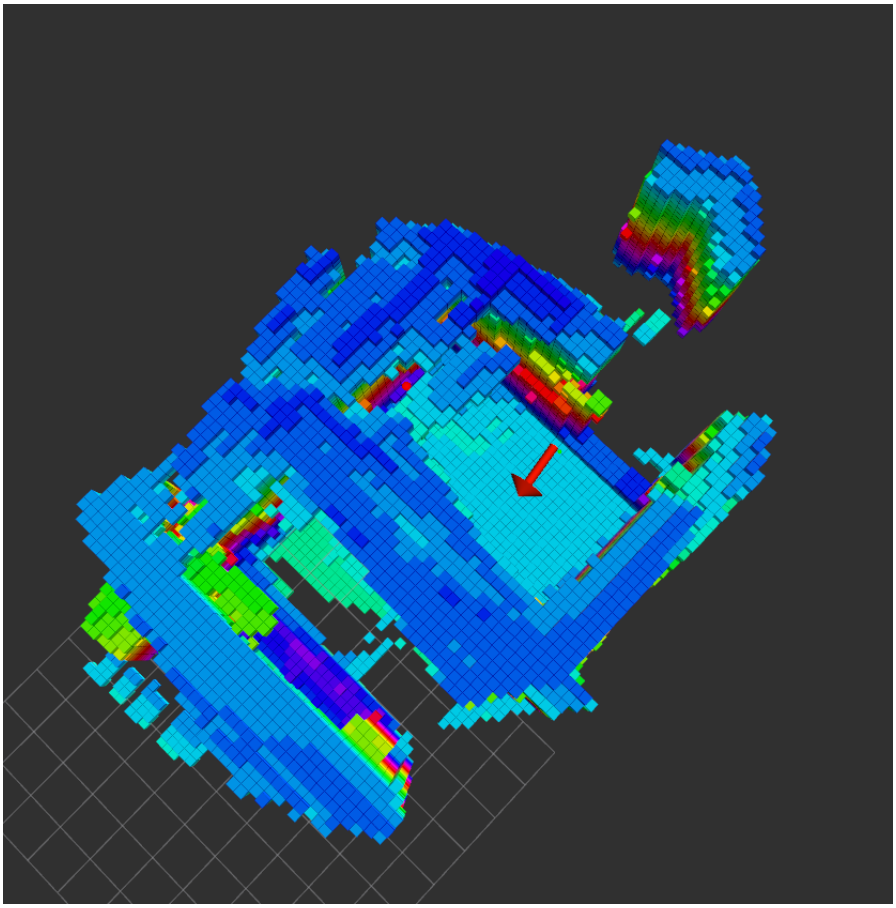
- "**/mavros/cmd/arming**" - This arms the drone enabling takeoff.
- "**/mavros/cmd/set_mode**" - Using this we are setting the drone in "GUIDED" mode, allowing it to take setpoints.
- "**/mavros/cmd/takeoff**" - This service initiates takeoff with the specified takeoff altitude and yaw.

For landing :

- **Logic** : If the absolute distance between the estimated marker centre(in global frame) and the drone in x and y direction is less than or equal to 0.25 units call the landing service. This ensures that the drone lands on the marker more accurately.
- "**/mavros/cmd/land**" - Similar to the takeoff service this land the drone on its current position or a specified position.

2.2 Map Generation

We use Voxblox for map generation. Voxblox is a volumetric mapping library based mainly on Truncated Signed Distance Fields (TSDFs).



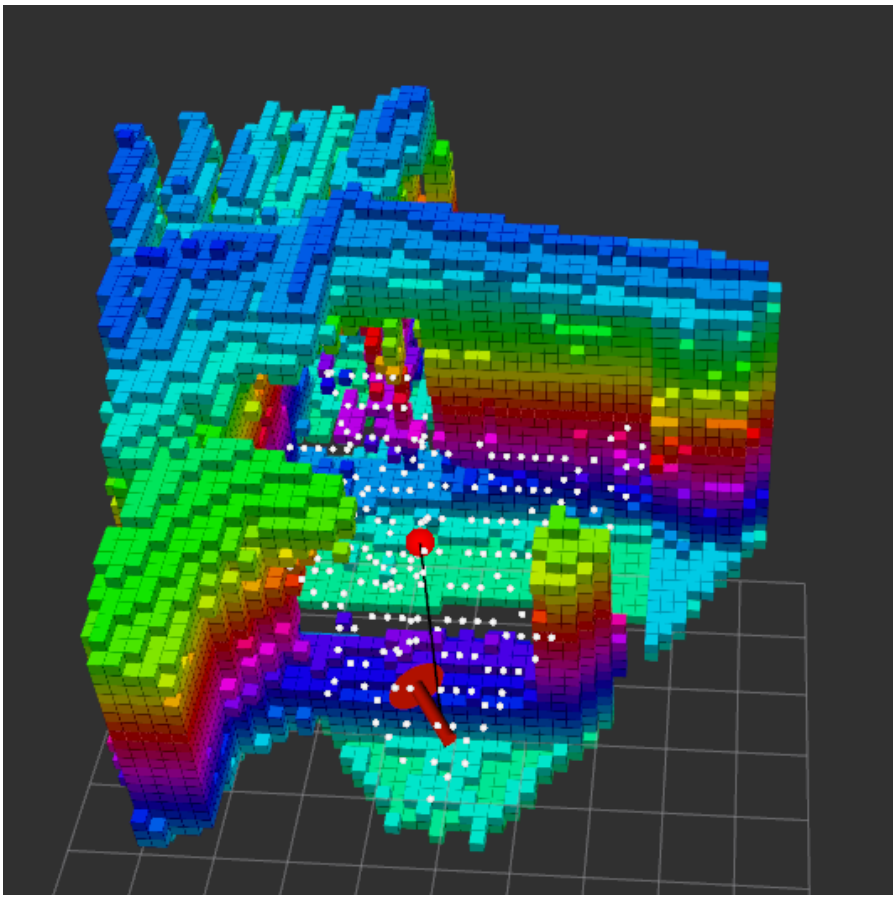
The system functions in two parts: first, incorporating incoming sensor data into a TSDF, and then propagating updated voxels from the TSDF to update the ESDF. Sensor data from stereo or RGB-D sensors comes in as colored pointclouds,which are then integrated into the TSDF. The ESDF is then updated by propagating changes from the TSDF.

For computational feasibility for exploration, the library uses a dynamically sized map that makes use of voxel hashing. This allows the use of hash table for fast mapping between block positions and their locations in memory.

2.3 Frontier Selection

A frontier is defined as a voxel that has more than one occupied neighbour. We iterate over all TSDF voxels to calculate all the frontier points, and then recursively cluster them together. The center of each frontier cluster is used as a potential waypoint for the path finder. In each planning iteration, the frontier centers are calculated from the

current TSDF map.



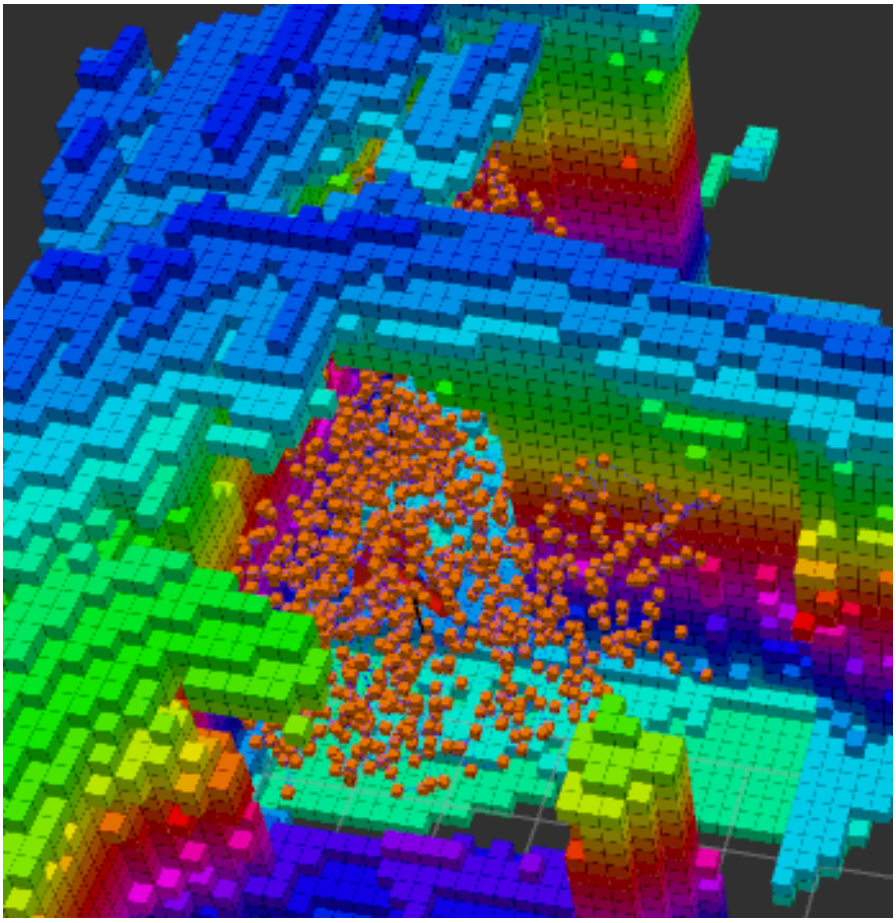
Out of these frontier centers, the one the lies the farthest along the direction that the quadrotor is currently facing is selected as the next waypoint for the planner. The current frontiers are then stored in a hash map to ensure that no frontier is visited again in subsequent planning iterations.

In case there are no visible frontiers, the quadrotor executes a routine where it spins around and changes its height from 4.0 to 1.0 before returning to its original odometry position, to develop the map in the surroundings.

2.4 Path Finding

Given a starting point (generally set to the current position of the quadrotor), and an endpoint (the next waypoint), the path is calculated by generating a random graph and

running A* on said graph. Using the start and end point as reference, a sampling box is defined along the line connecting both these points with specified width and height. Points are sampled in this sampling box, and added into a graph. Then the graph is connected by constructing an RTree and finding the 4 nearest neighbours for each point in the graph.



Once the path is generated, it is shortened by checking each pair of points for a free path in between. If a free path exists, the intermediate points are deleted. The pairs are checked in a divide and conquer fashion. Once the shortened path is generated, it is published to MAVROS and tracked in the planner node.

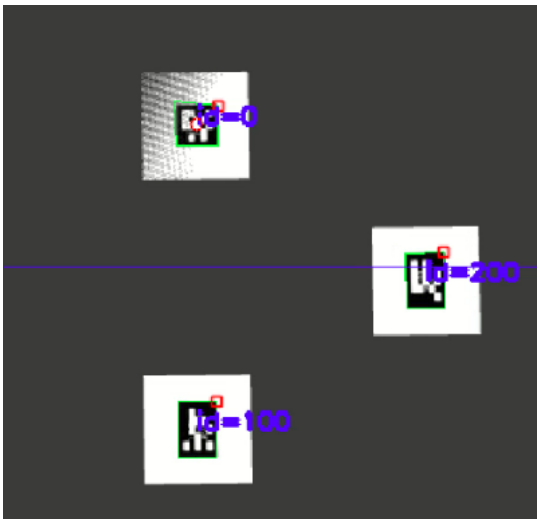
If the pathfinder fails to generate a path, a second attempt is made after increasing both the sampling density and the size of the sampling box. If a path is still unfeasible, a last attempt is made by trying to find a path to the point nearest to the end waypoint. If even this fails, the waypoint is marked infeasible and no path is returned.

As the path is being followed, the planner regularly looks at the next 4 waypoints and

checks them for the appearance of any obstacles on the path. If there is an obstacle, we query the pathfinder again to check if a replanned path is feasible. If it is, the waypoint is sent again, otherwise it is discarded and a new frontier is obtained.

2.5 Marker Detection

Aruco marker detection and pose estimation is being carried out in our node using the in-built openCV aruco module.



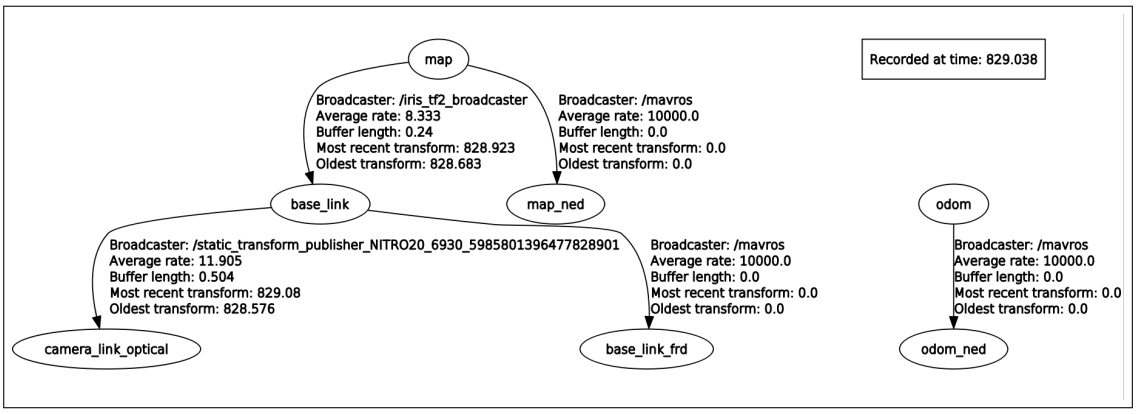
- **Detection** : The dictionary to be detected has a specification of 5x5_1000(ie. 1000 markers of 5x5 bits). For this we are using the *cv::aruco::detectMarkers()* function to get the *markerIds*, *rejectedCandidates* and corresponding *marker_corners*.
- **Centre estimation** : After obtaining the markerIds we can look up the corners corresponding to the required Id and take their average to get the marker centre.
- **Pose estimation** : Using the *cv::aruco::estimatePoseSingleMarkers()* function withe the estimated centre coordinates as input we get the camera-frame coordinates of the centre. By performing *cam-to-quad* and *quad-to-mavros frame transformations* on the camera-frame coordinates we obtain the map-frame(global) coordinates of the required aruco marker.

2.6 Fixes in Frame Tree Structure

The tf tree formed between the links was incorrect due to incorrect/non-existent transformations between the camera_link_optical, base_link and map. Rectification of this was done as follows:

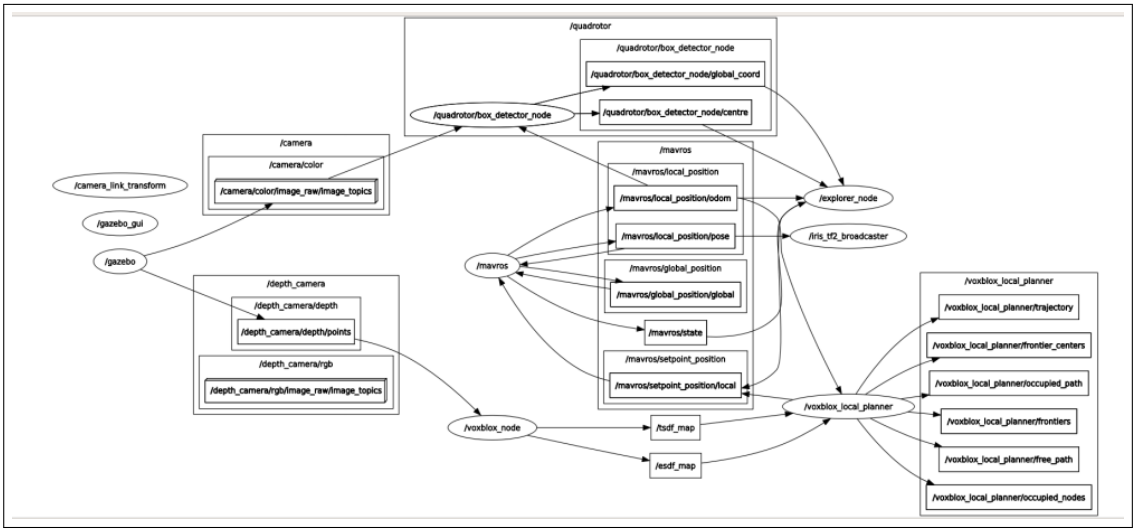
- Used a **static_transform_publisher** between base_link and camera_link_optical with a transformation of (0 0 0 -1.57 0 -1.57)
- Wrote a simple **tf static broadcaster** in which we use the odometry from mavros and publish the frame transformations between map and base_link via tf.

The final tf tree is as follows:



3 Software Architecture Description

3.1 rqt_graph



3.2 Parameter Files

The parameter files being used are:-

3.2.1 active_planner

- *frontier_params.yaml* - Parameters for frontier exploration
- *params.yaml* - General robot and debug parameters
- *voxblox_params.yaml* - Parameters for the voxblox server

3.2.2 box_detector

- *params.yaml* - Parameters specifying camera properties and position

3.2.3 explorer

- *params.yaml* - Single parameter for verbose output

3.3 Nodes

The nodes being run are as follows:-

3.3.1 voxblox_local_planner

Subscribed topics:-

- */esdf_map* - Gets the ESDF map generated by the Voxblox server
- */tsdf_map* - Gets the TSDF map generated by the Voxblox server
- */mavros/local_position/odom* - Subscribes to the current position of the quadrotor

Published topics:-

- */voxblox_local_planner/trajectory* - Visualize generated trajectory
- */voxblox_local_planner/frontier_centers* - Visualize frontier centers
- */voxblox_local_planner/occupied_path* - Visualize occupied waypoints on trajectory
- */voxblox_local_planner/frontiers* - Visualize frontier points
- */voxblox_local_planner/free_path* - Visualize free waypoints on trajectory
- */voxblox_local_planner/occupied_nodes* - Visualize occupied nodes in ESDF map
- */mavros/setpoint_position/local* - Publishes the setpoints that the quadrotor has to reach

Services served:-

- *activate* - Starts the planning loop
- *shutdown* - Shuts the planner down

3.3.2 voxblox_node

Subscribed topics:-

- */depth_camera/depth/points* - Subscribes to the point cloud obtained from the depth camera

Published topics:-

- */esdf_map* - Publishes ESDF map for the planner
- */tsdf_map* - Publishes TSDF map for the planner

3.3.3 explorer_node

Subscribed topics:-

- */mavros/local_position/odom* - Subscribes to the current position of the quadrotor
- */mavros/state* - Subscribes to the current state of the quadrotor
- */quadrotor/box_detector_node/global_coord* - Subscribes the coordinates of the detected ArUco marker (having id=0) in the world frame
- */quadrotor/box_detector_node/centre* - Subscribes the pixel coordinates of the detected ArUco marker (having id=0) in the image frame.

Published topics:-

- */mavros/setpoint_position/local* - Publishes the setpoints that the quadrotor has to reach
- */Aruco/message* - Publishes the status of marker detection

3.3.4 quadrotor/box_detector_node

This node detects the arUco marker and calculates the position of the arUco marker if it has id 0.
Subscribed topics:-

- */mavros/local_position/odom* - Subscribes to the current position of the quadrotor
- */camera/color/image_raw/image_topics* - Subscribes to the image from the downward facing camera.

Published topics:-

- */mavros/setpoint_position/local* - Publishes the setpoints that the quadrotor has to reach
- */quadrotor/box_detector_node/global_coord* - Publishes the coordinates of the detected arUco marker (having id=0) in the world frame
- */quadrotor/box_detector_node/centre* - Publishes the pixel coordinates of the detected arUco marker (having id=0) in the image frame.

4 References

- For Takeoff : px4-docs
- For tf broadcaster : tf2-Tutorials
- Helen Oleynikova, Zachary Taylor, Marius Fehr, Juan Nieto, and Roland Siegwart, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning", in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.
- Lukas Schmid, Michael Pantic, Raghav Khanna, Lionel Ott, Roland Siegwart, and Juan Nieto, "An Efficient Sampling-based Method for Online Informative Path Planning in Unknown Environments", in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 1500-1507, April 2020
- Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight, Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu and Shaojie Shen, IEEE Robotics and Automation Letters (RA-L), 2019.
- VoxBlox Repo : https://github.com/ethz-asl/voxblox