Introduction

The task of navigation at high altitudes imposes quite a challenge for vehicles and the snowfall at these levels worsens the conditions with road visibility tending to a dangerous low. The fact that snow clearing bulldozers have a hard-time navigating through these snow-clad roads creates a need for an efficient and cost-effective solution to such a problem.

With the advancement in drone technology, UAVs can serve a great purpose in solving this problem due to the advantage of being able to guide the ground vehicle while being aware of the terrain ahead due to its higher visibility from a high vantage point and ease of navigation.

Problem Statement

In the problem statement, we were asked to use a UAV to explore and map a given textured world. After mapping, the texture would be removed and the UAV must be used to guide a UGV through the mapped world along the center of the road.

Key points from the PS

- The UAV has an IMU, a GPS and a RGBD camera as sensors.
- The UGV has no sensor.
- The mapping needs to be completed in the textured world while the UGV navigates in the untextured world.

Keeping these things in mind we went forward with our approach and our solution has been explained below.

Approach

The idea was to create a rigid solution that our soldiers could use in varying real-life scenarios. The project is divided into three parts, namely - Mapping, UAV Planning and Control, and UGV Planning and Control. For Mapping, we segmented roads based on the UAV Camera data using U-Net model, and we used frontier exploration to map the entire road. We used Ardupilot, coupled with our own control and planning algorithms for UAV control. The localization of the UAV was done using filtering and combining its GPS and IMU data. Now for localization of UGV, we used Yolov5 and object detection algorithms on the UAV camera feed and localized it relative to the UAV. We used the Se2_Navigation package and Navfn for path planning and controlling the UGV.

DRONE LOCALISATION

For achieving this task, a ROS package (robot_localization) was used which aims at providing non-linear state estimation through sensor fusion of an arbitrary number of sensors. Since, in the given problem statement, GPS and IMU sensors were provided on the UAV. So, our goal was to fuse the incoming data from these sensors(rostopic /odometry/gps and /mavros/imu/data) to produce our desired odometry through the rostopic /odometry/filtered.

The state estimation nodes ekf_localization_node and ukf_localization_node tracks the 15-dimensional state of our UAV (X, Y, Z, $roll$, $pitch$, yaw, $\dot{X}$, $\dot{Y}$, $\dot{Z}$, $\dot{roll}$, $\dot{pitch}$, $\dot{yaw}$, $\ddot{X}$, $\ddot{Y}$, $\ddot{Z}$). In addition, other standard parameters like frequency, sensor_timeout, two_d_mode, transform_time_offset, odom0_differential, imu0_differential, odom0_relative, imu0_relative, mu0_remove_gravitational_acceleration was also configured accordingly.

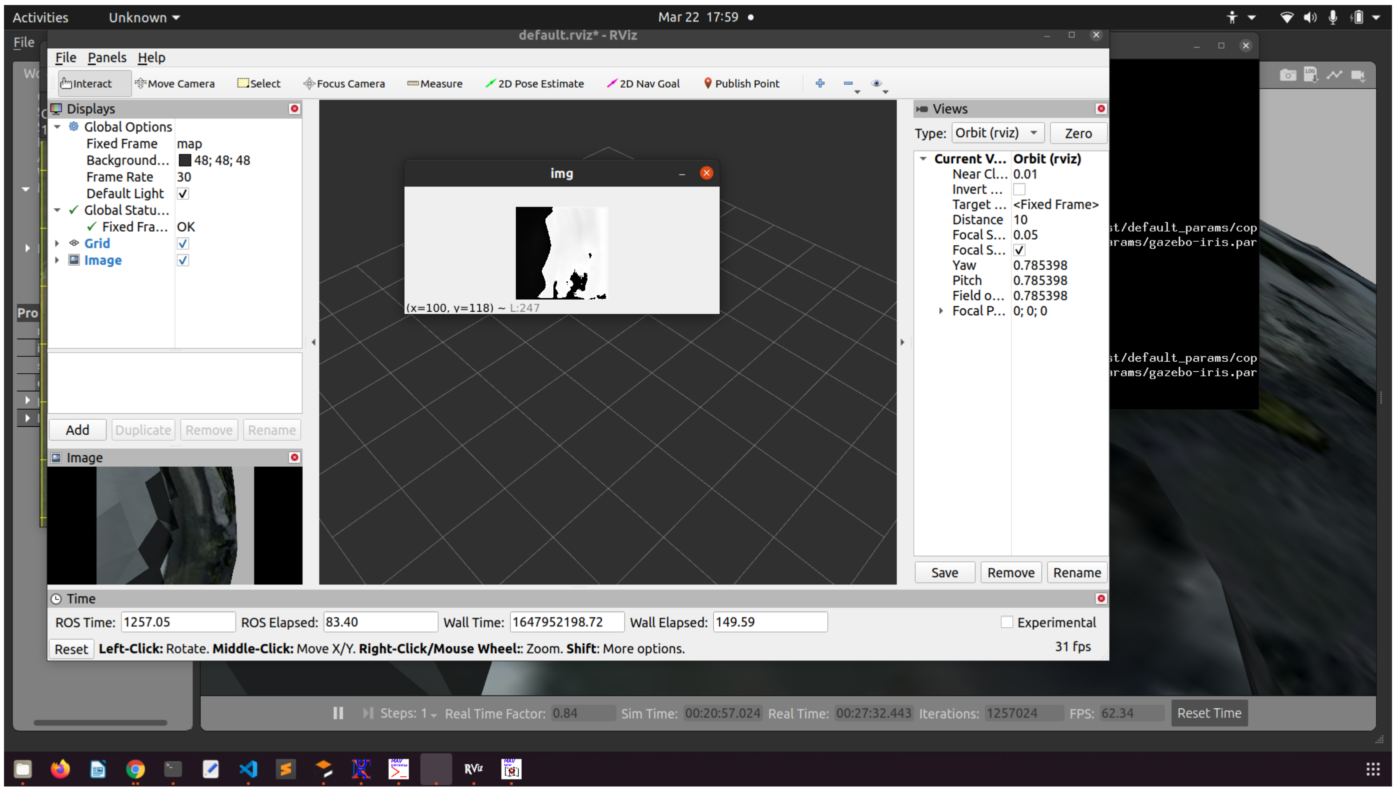
The chief task was to find out which variables of those messages (the 15-dimensional state of UAV in sensor_config) in param= " odom0_config " and param="imu0_config" should be fused into the final state estimate to produce the best results.

Coming to the navsat_transform_node, it transforms GPS data into a frame that is consistent with our robot’s starting pose (position and orientation) in its world frame and integrates it. For this, it was provided with message type sensor_msgs/NavSatFix (rostopic /mavros/global_position/raw/fix) and sensor_msgs/Imu (rostopic /mavros/imu/data) and other changes in parameters were done accordingly.

SEGMENTATION OF ROAD

- The task was to segment roads in real time in video footage obtained by the drone.
- For segmentation in real time, the computational cost of the segmentation model needed to be computationally less expensive. So, we built a custom UNET architecture which was under our computational constraints.
- The encoder of the UNET consisted of Convolutional, Batch Normalization and MaxPooling Layers with ReLU activation function.

- The number of channels in the encoding part were 64, 128, 256, 512 and 1024.
- The decoder used Transposed convolutions with skip connections.
- The input image to the UNET consisted of 128x128 Normalized RGB images with mean 0 and standard deviation 1 i.e. from the standard normal distribution.
- The data for training had to be made as there was no dataset available online.
- We made the dataset in the following way:
    - First, we took video footage of the drone on all three worlds.
    - Then, we annotated the data using the CVAT web tool.
    - We kept the annotations like the cityscapes dataset.
- Because of lack of data, the following augmentations were done on data using the albumentations library:
    - Rotation with limit 60 degree and probability 0.6
    - Horizontal Flip with probability 0.5
    - Vertical Flip with with probability 0.5
- The following steps taken for training:
    - Optimizer – Adam
    - Loss Function - Binary Cross Entropy with Logits
    - Learning Rates were used in range of 2e-3 to 5e-7
    - Epochs - 10,000
- After training the BCE Loss went down from 2.1 to 0.07
- After training on RGB images, we also trained the model on depth data obtained from the drone.
- The predictions from depth data and and RGB images were combined using bitwise and operator.
- The accuracy went up to more that 96.4% on the training set.



Global Options	
Fixed Frame	map
Background...	48; 48; 48
Frame Rate	30
Default Light	✓
Global Statu...	
Fixed Fra...	OK
Grid	✓
Image	✓

(x=100, y=118) ~ L:247

Type: Orbit (rviz) Zero

Current V...	Orbit (rviz)
Near Cl...	0.01
Invert ...	
Target ...	<Fixed Frame>
Distance	10
Focal S...	0.05
Focal S...	✓
Yaw	0.785398
Pitch	0.785398
Field o...	0.785398
Focal P...	0; 0; 0

ROS Time: 1257.05 ROS Elapsed: 83.40 Wall Time: 1647952198.72 Wall Elapsed: 149.59 Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

31 fps

```
t/default_params/cop
rams/gazebo-iris.par

t/default_params/cop
rams/gazebo-iris.par
```

Steps: 1 Real Time Factor: 0.84 Sim Time: 00:20:57.024 Real Time: 00:27:32.443 Iterations: 1257024 FPS: 62.34

Road detection results on live camera stream(white part is the road)

Mapping & Exploration

The mapping performed by the UAV drone has been implemented using the RTABMAP Ros package. The package uses RGB-D Slam approach and has been used create a 2d Occupancy grid map using the 3D pointcloud values obtained by the RGBD camera atop the drone.

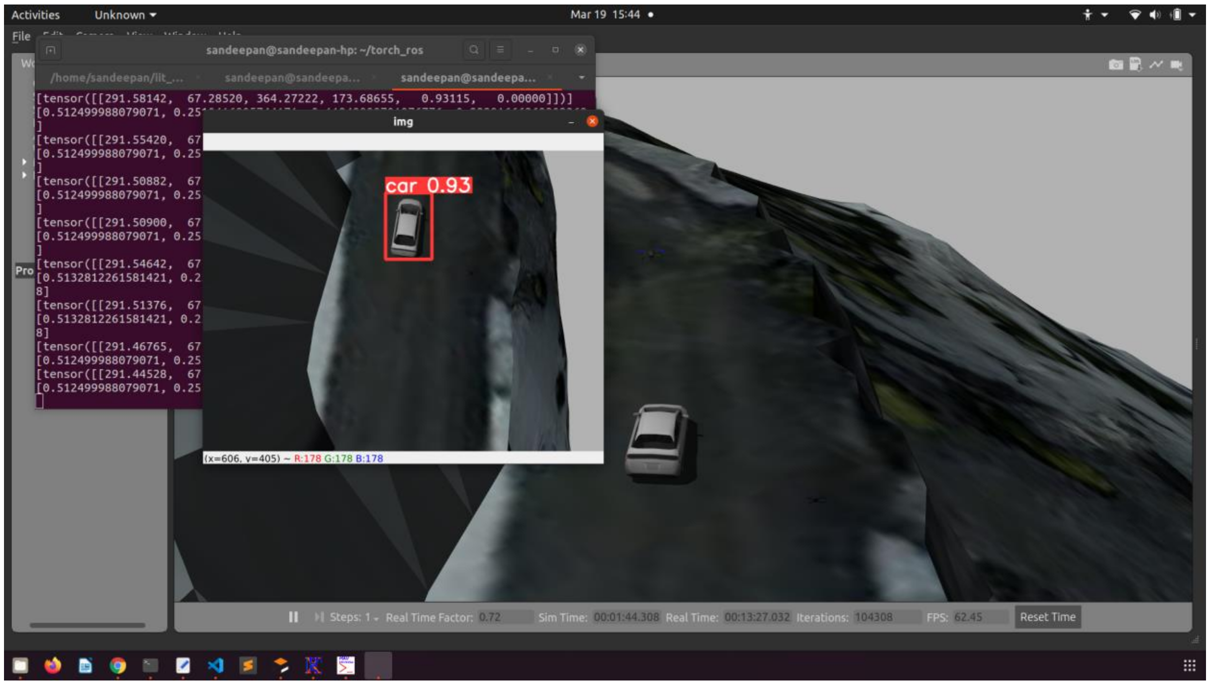
The package takes in the values from the camera as well as the odometry, and publishes the projected 2D map of the 3D environment.

Next, we’ve implemented the Frontier Exploration approach using the Frontier_Exploration ROS package in order to explore the world terrain. Frontiers are regions on the boundary between open space and unexplored space. By moving to a new frontier, we can keep building the map of the environment, until there are no new frontiers left to detect.

We create these frontiers atop the map received from the RTABMAP package and publish the movement commands to the “/mavros/setpoint_velocity/cmd_vel” topic to navigate the UAV through the world and perform mapping.

Tracking UGV

To track the UGV we use the drone’s RGB image as well as the depth information. First, we are getting the bounding box of the UGV using yolov5 object detection algorithm. We are training a yolov5s model using RGB images taken from drone and giving the bounding boxes. The model is then further used to detect bounding boxes of the car. We used the following Github repository to train our model and detect the bounding boxes: - https://github.com/ultralytics/yolov5

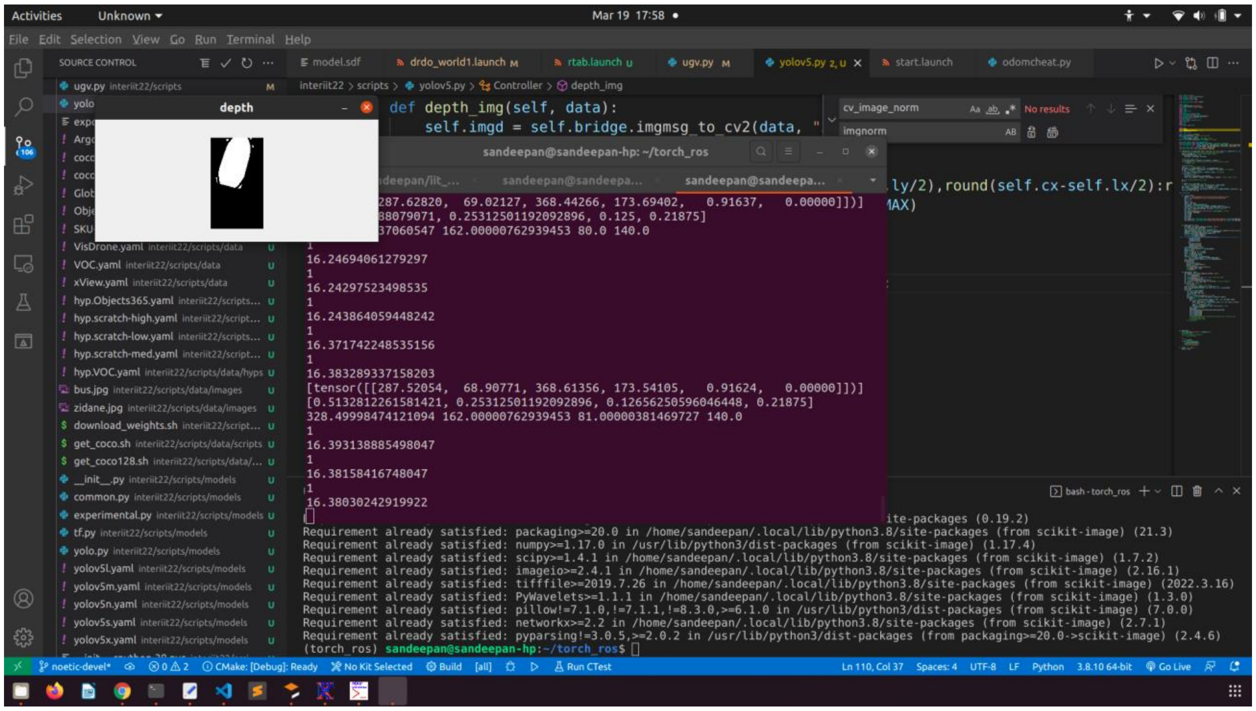


```
[tensor([[291.58142,  67.28520, 364.27222, 173.68655,   0.93115,   0.00000]])]
[0.512499988079071, 0.25
]
[tensor([[291.55420,  67
[0.512499988079071, 0.25
]
[tensor([[291.50882,  67
[0.512499988079071, 0.25
]
[tensor([[291.50900,  67
[0.512499988079071, 0.25
]
[tensor([[291.54642,  67
[0.5132812261581421, 0.2
8]
[tensor([[291.51376,  67
[0.5132812261581421, 0.2
8]
[tensor([[291.46765,  67
[0.512499988079071, 0.25
[tensor([[291.44528,  67
[0.512499988079071, 0.25
```

```
(x=606, y=405) ~ R:178 G:178 B:178
```

Car detection on live camera stream

Now that we have the bounding box of the car in the RGB image, we are extracting only that region from the depth image and then normalizing the depth values from a range of 0 to 1 only in that region. Now we get the car shape from the depth data and using OpenCV techniques we have obtained the orientation of the car accurately.



```
def depth_img(self, data):
    self.imgd = self.bridge.imgmsg_to_cv2(data, "
```

```
287.62820,  69.02127, 368.44266, 173.69402,   0.91637,   0.00000]])]
88079071, 0.25312501192092896, 0.125, 0.21875]
37060547 162.00000762939453 80.0 140.0
1
16.24694061279297
1
16.24297523498535
1
16.243864059448242
1
16.371742248535156
1
16.383289337158203
[tensor([[287.52054,  68.90771, 368.61356, 173.54105,   0.91624,   0.00000]])]
[0.5132812261581421, 0.25312501192092896, 0.12656250596046448, 0.21875]
328.49998474121094 162.00000762939453 81.00000381469727 140.0
1
16.393138885498047
1
16.38158416748047
1
16.38030242919922
```

```
ite-packages (0.19.2)
Requirement already satisfied: packaging>=20.0 in /home/sandeepan/.local/lib/python3.8/site-packages (from scikit-image) (21.3)
Requirement already satisfied: numpy>=1.17.0 in /usr/lib/python3/dist-packages (from scikit-image) (1.17.4)
Requirement already satisfied: scipy>=1.4.1 in /home/sandeepan/.local/lib/python3.8/site-packages (from scikit-image) (1.7.2)
Requirement already satisfied: imageio>=2.4.1 in /home/sandeepan/.local/lib/python3.8/site-packages (from scikit-image) (2.16.1)
Requirement already satisfied: tifffile>=2019.7.26 in /home/sandeepan/.local/lib/python3.8/site-packages (from scikit-image) (2022.3.16)
Requirement already satisfied: PyWavelets>=1.1.1 in /home/sandeepan/.local/lib/python3.8/site-packages (from scikit-image) (1.3.0)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in /usr/lib/python3/dist-packages (from scikit-image) (7.0.0)
Requirement already satisfied: networkx>=2.2 in /home/sandeepan/.local/lib/python3.8/site-packages (from scikit-image) (2.7.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/lib/python3/dist-packages (from packaging>=20.0->scikit-image) (2.4.6)
(torch_ros) sandeepan@sandeepan-hp:~/torch_ros$
```

(As can be seen in the terminal the angle calculated is approximately 16 degrees, using the depth data)

Then we calculate the distance of the car from the drone using the center coordinates of the bounding box and the camera specifications (K matrix) and then publish the transform between the drone and the car, thus connecting the transform from the Odom frame to the car base frame.

We have also calculated the velocity of the car by calculating the change in the relative distance between the drone and the car using the bounding box and orientation calculated above.

UGV Control

To control the UGV, we used a ROS package called se2_navigation which subscribes to the odometry topic of the car which is the position and orientation of the car along with the velocity of the car with respect to a suitable odom frame which in this case is the initial starting position and orientation of the drone.

Now when we publish a suitable goal point in the same odom frame the se2 OMPL planner inside this package plans a proper trajectory for the car to follow which is a sequential array of poses in the odometry frame.

To initiate this planning we are calling the '/se2_planner_node/ompl_rs_planner_ros/planning_service'.

Consequently which, we call the path tracking service '/prius/controller_command_service' which starts tracking the path as planned above.

So, briefly, we just feed the current position of the car and a desired goal location to the package it calculates a trajectory and starts tracking it. The inputs are only the position and velocity of the car in the starting drone frame.