A project to develop/adapt a navigation system for outdoor robotics particularly aiming for use-cases in agriculture.

- We call this *vox_nav* since it relies on voxel-based 3D occupancy grid (Octomap) in order to navigate a mobile robot.
- This project’s main use case is navigation of a mobile robot in rough outdoor environments.
- At the moment, environment representation is done with Octomap. Based on the 3D topology of map we overlay cost layers onto octomap, this bases on several critics such as elevation, roughness etc.
- Planning of vox_net relies on OMPL, we provide SE2, SE2.5 planners. SE2 planners can also be constrained to output plans for Ackerman type robots with DUBINS AND REEDS-SHEEP.
- SE2.5 planner computes valid plans that can go through ramps and hills, giving the robot chance to traverse through them, we this as one of the main difference of vox_net over existing navigation frameworks.
- Octomaps are usually acquired from a pointcloud map of the environment that you would like to navigate in.
- Map server manages pointclouds maps, a pointcloud map is required to have a datum (GPS coordinates and orientation), see the configuration section.

This branch (foxy) is aiming for ROS2 Foxy distro.
• Install ROS2 foxy.

Deb installation is strongly recomended. You can always find updated step to install [here](https://index.ros.org/doc/osal/Installation/Foxy/Linux-Install-Debians/) To install ROS2 foxy desktop:

```bash
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
sudo apt update && sudo apt install curl gnupg2 lsb-release
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key --add -
sudo sh -c 'echo "deb [arch=amd64,arm64] http://packages.ros.org/ros2/ubuntu lsb-release -cs main" > /etc/apt/sources.list.d/ros2-latest.list'
sudo apt update
sudo apt install ros-foxy-desktop
source /opt/ros/foxy/setup.bash
```

• A few helper packages we use for gui and installation:

```bash
source /opt/ros/foxy/setup.bash
sudo apt install python3-colcon-common-extensions
sudo apt install --y python3-rosdep2
sudo apt-get install python3-vcstool
sudo apt-get install xdotool
sudo apt-get install coinor-libipopt-dev
rosdep update
```

• Get the project repository, source build deps and build deps first:

```bash
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws
source /opt/ros/foxy/setup.bash
wget https://raw.githubusercontent.com/jediofgever/vox_nav/foxy/underlay.repos
vcs import src < underlay.repos
rosdep install --y --from-paths src --ignore-src --rosdistro foxy
colcon build --symlink-install --cmake-args -DCMAKE_BUILD_TYPE=Release -DWITH_IPOPT=true --packages-select casadi ompl
sudo cp install/ompl/lib/libompl.so* /usr/local/lib/
sudo cp install/casadi/lib/libcasadi.so* /usr/local/lib/
sudo rm -rf src/ompl/
sudo rm -rf src/casadi/
```
There are essentially 2-3 dependency libraries that needs source build. perception_pcl , OMPL and casadi. The above script will build and install them and remove the source code as its not needed.

- With above we only built dependencies, now lest build vox_nav itself

```
source /opt/ros/foxy/setup.bash
cd ~/ros2_ws
colcon build --symlink-install --cmake-args -DCMAKE_BUILD_TYPE=Release --packages-skip-regex archived --packages-skip ompl casadi vox_nav_openvslam
source ~/ros2_ws/install/setup.bash
```

**Note:** Pay attention that we have disabled the build of *archived_* packages. Some packages(e.g vox_net_openvslam) has quite some amount of deps. If you like to use please refer to SLAM vox_net_openvslam section , there you will find instructions to install vox_net_openvslam deps. After that you can remove --packages-skip vox_net_openvslam part from colcon build command.

You can update code by vcs tool, you can also us classic git pull.

```
cd ~/ros2_ws
vcs import src < underlay.repos
vcs pull src
```

**Note:** If for some reason you could not build vox_nav, a good place to seek for a solution is the github actions file that we have. After each pull-push github actions is setup to build the vox_nav on remote to ensure stability of builds. Find a recent successful build and see the commands in .github/workflows/main.yml. The commands should more or less look as in this page.
Botanbot sim is a good reference to see how it is possible to configure vox_nav for your custom robot. Be aware that at the moment we only cover Ackerman type robot. Although it is possible to configure MPC Controller’s params, such that it works for Diff-Drive robots, the primary model is aimed for Ackerman type robots.

Please see botanbot_sim in this link: https://github.com/jediofgever/botanbot_sim

botanbot is a basic Ackermann type robot with all essential sensors (GPS, LIDAR, CAMERA, IMU). botanbot bringup package includes sample parameters and launch files to run vox_nav.

Important nodes of vox_nav are: controller_node, map_server_node, planning_node.

parameters for these nodes are more or less as following:

```yaml
vox_nav_planner_server_rclcpp_node:
  ros__parameters:
    planner_plugin: "ElevationPlanner" # other options: "SE2Planner",
    "ElevationPlanner"
    expected_planner_frequency: 1.0
    planner_name: "PRMstar" # PRMstar, LazyPRMstar, RRTstar,
    "PRMstar"
    interpolation_parameter: 20 # ABITstar, AITstar, CForest, LBTRRT,
    "PRMstar"
    planner_timeout: 10.0
    octomap_voxel_size: 0.2
    robot_body_dimens:
      x: 1.2
      y: 0.9
      z: 0.8
    robot_mesh_path: "package://botanbot_description/meshes/base_simplified.stl" # leave empty if you do not have one, robot_mesh_path: ""
    SE2Planner:
      plugin: "vox_nav_planning::SE2Planner" # CForest: Recommended
      se2_space: "REEDS" # "DUBINS", "REEDS", "SE2" ## PS.
    "SE2Planner"
    z_elevation: 1.5 # Elevation of robot from ground
    planer: robot should not collide with plane
    rho: 2.5 # Curve radius form reeds and dubins
    only
    state_space_boundaries:
      minx: -50.0
      maxx: 50.0
      miny: -50.0
```

(continues on next page)
maxy: 50.0
minyaw: -3.14
maxyaw: 3.14

ElevationPlanner:
  plugin: "vox_nav_planning::ElevationPlanner"  # PRMstar: Recommended
  se2_space: "DUBINS"  # "DUBINS", "REEDS", "SE2" ### PS.

Use DUBINS OR REEDS for Ackermann
  rho: 1.0  # Curve radius for reeds and

--dubins only
state_space_boundaries:
  minx: -50.0
  maxx: 50.0
  miny: -50.0
  maxy: 50.0
  minz: -2.0
  maxz: 12.0

vox_nav_controller_server_rclcpp_node:
ros__parameters:
  controller_plugin: "MPCControllerCasadiROS"  #

--other options: non
  controller_frequency: 15.0

MPCControllerCasadiROS:
  plugin: "mpc_controller_casadi::MPCControllerCasadiROS"
  N: 8  # timesteps

--in MPC Horizon
  DT: 0.2  #

--discretization time between timesteps(s)
  L_F: 0.66  # distance

--from CoG to front axle(m)
  L_R: 0.66  # distance

--from CoG to rear axle(m)
  V_MIN: -1.0  # min / max

--velocity constraint(m / s)
  V_MAX: 2.0
  A_MIN: -2.0  # min / max

--acceleration constraint(m / s ^ 2)
  A_MAX: 1.0
  DF_MIN: -1.5  # min / max

--front steer angle constraint(rad)
  DF_MAX: 1.5
  A_DOT_MIN: -1.0  # min / max

--jerk constraint(m / s ^ 3)
  A_DOT_MAX: 1.0
  DF_DOT_MIN: -0.8  # min / max

--front steer angle rate constraint(rad / s)
  DF_DOT_MAX: 0.8
    Q: [10.0, 10.0, 0.1, 0.1]  # weights
--on x, y, psi, and v.
  R: [10.0, 100.0]  # weights

--on jerk and slew rate (steering angle derivative)
  debug_mode: False  # enable/

--disable debug messages
params_configured: True

vox_nav_map_server_rclcpp_node:
ros__parameters:
  pcd_map_filename: /home/atas/colcon_ws/src/botanbot_sim/botanbot_bringup/maps/uneven_world.pcd
  # PCD PREPROCESS PARAMS
  pcd_map_downsample_voxel_size: 0.2 # Set to smaller if you do not want downsample
  pcd_map_transform:
    # optional rigid-body transform to pcd file
    translation:
      x: 0.0
      y: 0.0
      z: 0.0 #1.0
    rotation:
      # intrinsic
      rotation X-Y-Z (r-p-y)sequence
      r: 0.0 #3.14
      p: 0.0 #1.57
      y: 0.0 #1.57
  apply_filters: False
  remove_outlier_mean_K: 50
  remove_outlier_stddev_threshold: 0.1
  remove_outlier_radius_search: 0.1
  remove_outlier_min_neighbors_in_radius: 1
  # COST REGRESSION CRITICS AND PARAMS
  cell_radius: 0.8 # Works as resolution of cost regression onto map
  max_allowed_tilt: 0.6 # 1st Cost critic Any angle(radians) higher than this is marked as NON-traversable
  max_allowed_point_deviation: 0.2 # 2nd Cost critic Point deviation from plane, this could be viewed as roughness of each cell
  max_allowed_energy_gap: 0.2 # 3rd Cost critic Max Energy in each cell, this is determined by max height difference between edge points of cell
  node_elevation_distance: 1.8 # According to cell_radius, cell centers are sampled from original point cloud map, they are elevated from the original cloud
  plane_fit_threshold: 0.1 # when fitting a plan to each cell, a plane_fit_threshold is considered from plane fitting from PCL
  robot_mass: 0.1 # approximate robot mass considering cell_radius
  average_speed: 1.0 # average robot speed(m/s) when calculating kinetic energy m = 0.5 * (m * pow(v,2))
  cost_critic_weights: [0.6, 0.2, 0.2] # Give weight to each cost critic when calculating final cost
  # PCD MAP IS TRANSLATED TO OCTOMAP TO BE USED BY PLANNER
  octomap_voxel_size: 0.2
  octomap_publish_frequency: 1
  publish_octomap_visuals: true
  octomap_point_cloud_publish_topic: "octomap_pointcloud" # sensormsgs::msg::PointCloud2 that represents octomap
(continues on next page)
octomap_markers_publish_topic: "octomap_markers" #

- visualization_msgs::msg::MarkerArray that represents octomap
  map_frame_id: "map"
  utm_frame_id: "utm"
  yaw_offset: 1.57  # see
  navsat_transform_node from robot_localization, this offset is needed to recorrect
  orientation of static map
  map_datum:
    latitude: 49.89999996757017
    longitude: 8.89999997371747
    altitude: 1.6
    quaternion:
      x: -0.0001960611448920198
      y: -0.003682083159658604
      z: 4.672499893387009e-05
      w: 0.9999932007970892

vox_nav’s skeleton is made by following ROS2 nodes;

Some highlights of the features for this nodes are as follows.

1. vox_nav_planner_server_rclcpp_node

   You can select an available planner plugin(SE2Planner or SE3Planner), be sure to see through the parameters.
   SE2Planner can be configured such that kinematic constrains of ackemann robots are respected. e.g select REEDS
   OR DUBINS spaces. The planner plugins are interfaced with OMPL. Many of OMPL planners could be selected.
   The planners are Sampling-Based, they utilize a octomap of environment in order to perform collision checks. You
   also need to provide a 3D volume box that represents body of your robot. see the robot_body_dimens params for that.

2. vox_nav_controller_server_rclcpp_node

   TODO

3. vox_nav_map_server_rclcpp_node

   You will need to provide a pre-built pcd map of environment for this node to consume. This map needs to have a datum
   of its origin(GPS coordinates and IMU acquired absolute heading). This is basically the pose where you initialize your
   SLAM algorithm to build your map. This is needed in order to geo-reference your map. vox_nav_openslam can help
   you with building such maps, these is also a helper node to dump map meta information including datum. Refer to
   SLAM section to see more details. With this information the node is able to grab your pcd map and georeference it
   utilizing robot_localization package. The pcd map is converted to an octomap and published with configured voxel sizes
   and topic names. You should visualize topics in RVIZ, in order to make sure the map looks as expected. visualizing as
   markers usually lags RVIZ, instead we recommand you to visualize pointcloud topic of octomap.
Botanbot is a simple 4 wheeled, ackermann driven mobile robot. It is simulated under Gazebo with all required essential sensors in order to do outdoor navigation. The following table shows currently supported sensors.

1. Sensor support for Botanbot

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Topic</th>
<th>MSG TYPE</th>
<th>Update Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIDAR</td>
<td>/velodyne_points</td>
<td>sensor_msgs::msg::PointCloud2</td>
<td>30</td>
</tr>
<tr>
<td>RealSense D435 COLOR</td>
<td>/camera/color/image_raw</td>
<td>sensor_msgs::msg::Image</td>
<td>30</td>
</tr>
<tr>
<td>CAMERA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RealSense D435 DEPTH</td>
<td>/camera/aligned_depth_to_color/image_raw</td>
<td>sensor_msgs::msg::Image</td>
<td>30</td>
</tr>
<tr>
<td>CAMERA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RealSense D435 IR1 CAM-</td>
<td>/camera/infra1/image_raw</td>
<td>sensor_msgs::msg::Image</td>
<td>1</td>
</tr>
<tr>
<td>ERA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RealSense D435 IR2 CAM-</td>
<td>/camera/infra2/image_raw</td>
<td>sensor_msgs::msg::Image</td>
<td>1</td>
</tr>
<tr>
<td>ERA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPS</td>
<td>/gps/fix</td>
<td>sensor_msgs::msg::NavSatFix</td>
<td>30</td>
</tr>
<tr>
<td>IMU</td>
<td>/imu/data</td>
<td>sensor_msgs::msg::IMU</td>
<td>1</td>
</tr>
</tbody>
</table>

• Botanbot navigation in farming world
• Botanbot in Hilly Gazebo world
(botanbot_sim)[https://github.com/jediofgever/botanbot_sim] is configured to use vox_nav system for 3D navigation. If you are trying to setup your robot for use with vox_nav, botanbot_sim is the right place to look for. A RQT based gui is also provided in botanbot_gui, through this gui it is possible to send goal commands to action server(NavigateToPose) of vox_nav. If using simulation, botanbot_gazebo includes sensor plugins.
The project has a RQT GUI plugin that lets you to interact with robot. To start with this plugin make sure in previous sections you built project successfully.

- source your `colcon_ws` and start the project with:

```bash
source install/setup.bash
rqt --force-discover
```

The `rqt` window should open as above. You should now find our plugin under;

`Plugins -> Visualization -> Control Plugin.`

Click on Control Plugin and you would be able to see;
1. Interact with GUI

Select a world that you would like to run the robot in then Click on Start Gazebo Stand Alone, to start botanbot simulation. Note that the Gazebo worlds we use are large, so your computer needs to have a dedicated GPU, it takes approx. 10 seconds for simulation to start in my case. After a while you should see Gazebo starting.

You may not see the Botanbot at first, At left side of Gazebo simulation, find the models -> botanbot right click and then follow botanbot model. This should put the focus onto botanbot.

You can also click on start RVIZ and you should be able to see sensor data and robot model in rviz ;
You can jog botanbot with RQT plugin, use $L/R$ for giving angular speed and $D/R$ for linear speed.
Chapter 4. Running Project
OpenVSLAM[1] is an open-source Visual SLAM framework. It supports various types of camera models in order to achieve SLAM only based on a camera image. In agric-fields with repetitive/featureless, textureless environments, results on LIDAR-based SLAM are poor. For instance, neither with Cartographer or LIDAR SLAM package, we could build a reasonable map. However, openvslam performed quite well in a simulated Gazebo world that covers tomato fields.

- Installation and Usage of OpenVSLAM

It is best to refer to their read the docs website of OpenVSLAM here: https://openvslam.readthedocs.io/en/develop/installation.html for an updated information related to installation. We will still provide steps here to achieve installation of openvslam.

Most of openvslam dependencies will be already present in your Ubuntu system, but make sure you have all following dependencies:

- **Eigen**: version 3.3.0 or later.
- **g2o**: Please use the latest release. Tested on commit ID 9b41a4e.
- **SuiteSparse**: Required by g2o.
- **DBoW2**: Please use the custom version of DBoW2 released in https://github.com/shinsumicco/DBoW2.
yaml-cpp: version 0.6.0 or later.  
OpenCV: version 3.3.1 or later.  
Pangolin: Required for visualization and GUI.

yaml-cpp:

```bash
sudo apt-get install libyaml-cpp-dev
```

Some of above deps will need a source build. Which can be done as:

DBoW2:

```bash
cd ~/
git clone https://github.com/shinsumicco/DBoW2.git
cd DBoW2
mkdir build && cd build
cmake \n   -DCMAKE_BUILD_TYPE=Release \n   -DCMAKE_INSTALL_PREFIX=/usr/local \n   ..
make -j8
sudo make install
```

g2o:

```bash
cd ~/
git clone https://github.com/RainerKuemmerle/g2o.git
cd g2o
git checkout 9b41a4ea5ade8e1250b9c1b279f3a9c098811b5a
mkdir build && cd build
cmake \n   -DCMAKE_BUILD_TYPE=Release \n   -DCMAKE_INSTALL_PREFIX=/usr/local \n   -DCMAKE_CXX_FLAGS=-std=c++11 \n   -DBUILD_SHARED_LIBS=ON \n   -DBUILD_UNITTESTS=OFF \n   -DBUILD_WITH_MARCH_NATIVE=OFF \n   -DG2O_USE_CHOLMOD=OFF \n   -DG2O_USE_CSPARSE=ON \n   -DG2O_USE_OPENGL=OFF \n   -DG2O_USE_OPENMP=ON \n   ..
make -j4
sudo make install
```

Pangolin:

Build openvslam

```bash
cd ~/
git clone https://github.com/xdspacelab/openvslam.git
cd openvslam
git checkout develop
mkdir build && cd build
```

(continues on next page)
After all of this `vox_nav_openvslam` should compile fine.

In order to build a map with provided server package do following:

```bash
ros2 launch vox_nav_openvslam openvslam_mapping.launch.py output_map_filename:=${HOME}/test_map.msg
```

Currently mono and RGBD cameras are supported. RGBD is recommended and in the default settings we use ATM. Mono images cannot be correctly scaled to real world unlike RGBD. See the mapping.launch.py under `vox_nav_openvslam` and make sure the camera topics are correctly remapped.

**Note:** text Visual SLAM has difficulties dealing with pure rotations. So the robot needs at least some translation as well when taking sharp turns.

Jog the robot with rqt gui plugin and visualize the map with pangolin viewer. A map with extension of `.msg` will be dumped to the path you passed to `output_map_filename`. The scripts provided in `vox_nav_openvslam` are able to visualize and convert this `.msg` to `.pcd` extension.

In order to visualize the created map do the following

```bash
make -j4
sudo make install
```

```bash
python3 visualize_openvslam_map.py map.msg
```
You can convert .msg to .pcd with provided script:

```
python3 convert_msg_to_pcd.py map.msg out.pcd
```

Lastly Localization can be performed in a pre build map;
where the argument is pull path to prebuild map in .msg format.

[1](https://github.com/xdspacelab/openvslam)
ARCHIVED PACKAGES

Note that the material covered in this page is not actively in use by vox_nav. However this is kept for archival purposes. Some of the packages may still be useful for future development.

1. Related to Google Cartographer 3D SLAM

Unfortunately Cartographer has not been ported to ROS2 completely ATM. We can still create maps and visualize results, the vox_nav_cartographer is correctly configured and tested. But the node named cartographer_asset_writer has not been ported, this node is used to recieve actual 3D map in .pcd or other formats, therefore we cannot exploit created map now. Hopefully soon enough this will be available. For exploiting the actual maps created see the next SLAM options below this section. We have added configuration package(vox_nav_cartographer) in order to build 3D maps using google cartographer.

In order to use the provided configuration package, one will need to create ros2 bag files with required topics. For now we use 1 LIDAR and 1 IMU(Can be changed to 2 LIDAR in future). In order to record a bag file this the required topics, do the following:

```bash
ros2 bag record /velodyne_points /imu/data
```

A bag file will be created. Cartographer expects that you define the rigid body transforms between sensor links and robot body frame(base_link). This transforms are defined in archived_vox_nav_cartographer/urdf. You might need to modify translation and rotation between velodyne sensor and imu sensor for different setup. A strict calibration might not be necessary between IMU and LIDAR.

Also see the cartographer.launch.py file and make sure the data topics are remapped correctly. After we have the bag file and configuration ready, we do the following to build the 3D map.

```bash
ros2 launch archived_vox_nav_cartographer cartographer.launch.py use_sim_time:=true bag_file:=${HOME}/rosbag2_2020_12_18-10_25_37/rosbag2_2020_12_18-10_25_37_0.db3
```

Wait for cartographer to finish and do optimizations on the map.

2. Related to LIDAR SLAM ROS2 3D PACKAGE

As second (but primary for now) SLAM package we have [lidarslam_ros2](https://github.com/jediofgever/lidarslam_ros2) package. The package is able to generate nice maps(in city like envirnments) based on GICP/NDT.

This package will be automatically cloned to workspace with the vcs tool. Following same fashion with google cartographer, we need to record bag files and generate maps with data inside this bags. Assuming that we have recorded bag file that contains point cloud and imu(optional) with topic names velodyne_points, imu/data, we can generate maps with ;
ros2 bag play -r 0.5 rosbag2_2020_12_18-10_25_37/
ros2 launch lidarslam lidarslam.launch.py
rviz2 -d src/lidarslam_ros2/lidarslam/rviz/mapping.rviz

Note; if the sensor topic names are different then you need to recorrect them in tha launch file `lidarslam.launch.py` execute all commands in separate terminals. Here are a few example maps created when vox_nav was taking a tour to gas station and retruning back.
3. Related to grid_map package

`archived_vox_nav_grid_map` is a package that reads a prebuilt map in .pcd format and publishes grid_map with several layers(elevation, traversability). The maps can be built with any SLAM algorithm, at the default we have openvslam though. There are several important parameters in configuration files of `archived_vox_nav_grid_map`.

```
archived_vox_nav_grid_map_node:
ros__parameters:
  resolution: 0.15
  pcd_file_full_path: /home/ros2-foxy/f.pcd
  map_frame: grid_map
  topic_name: grid_map
  map_publish_fps: 10
  min_points_in_cell: 1
  cloud_transform:
    translation:
      x: 0.0
      y: 0.0
      z: 0.5
    rotation:  # intrinsic rotation X-Y-Z (r-p-y) sequence
      r: 3.14
      p: 1.57
      y: 1.57
  downsample_voxel_size: 0.01
  remove_outlier_mean_K: 50
  remove_outlier_stddev_threshold: 0.5
```

We should specify a full path to the pcd file here. There are other important parameters related to transform of pcd. In openvslam the coordinate frames are not complying with standard ROS frames(x forward, y left, z upwards). Therefore we need to transform pcd to comply with standard ROS frames. There are also some PCL utilities to denoise pcd and get
better map. Play with them to find best working in for each case. In simulation the above parameters leads to following grid maps depicted in pictures.

**Note:** we use tf_static_transform publisher to create a dedicated coordinate frame for grid_map. You can change the translation and rotation in mapping.launch.py. This is important, it gives us flexibility to align map with other global localization such as GPS. But we do not have done any work towards that yet.

```
ros2 launch archived_vox_nav_grid_map archived_vox_nav_grid_map.launch.py
```