

UVA FI tenth Assignment 4

Instructor: Prof. Madhur Behl

ROS Cartographer SLAM, AMCL Particle Filter, and Pure Pursuit Race Trajectory Following

Due Date: Tue, December 5th 2023 - 9:30am Rice 242.**Assignment Overview:**

In this lab assignment your team will need do the following:

[Section A] SLAM: You will first create a 2D occupancy grid map for the Rice 242 track using ROS Cartographer SLAM. Section A in this assignment, walks you through how to do this and save a map. We have provided everything needed for this.

[Section B] AMCL Localization: Having build the map, you will use AMCL particle filter to localize the FI tenth racecar on that map. We have provided everything needed for this.

[Section C] Waypoint Logging: Once you are able to obtain the pose of the car from AMCL, you can teleop the car around the track to roughly drive a good racing line. While this is happening, you will log the poses of the car on the map into a trajectory of waypoints. We have provided everything needed for this.

[Section D] Pure Pursuit: Now you have a map, a reference trajectory (the saved waypoints), and the ability to localize the car in that map. **You will implement the pure-pursuit trajectory following algorithm** to optimize your lap time.

Deliverables and Grading

We need a submission from each team + peer evals as usual but with more details.

>> What do you need to submit ?

Submit the following as `team_name_a04.zip` containing the 4 things below. One submission needed per team.

1. **PDF assignment report** - A single PDF document addressing the questions Q1-Q3 below.
2. **The `base_map.pgm` file** for the map your team created.
3. **The `fltenth_purepursuit` package** you created with the **`pure_pursuit.py`** node and any launch files you created.
4. **Link to a youtube video** showing: Add the video link to a text file named `team_name_a04vid.txt`
 - (a) *Map creation*: The video should show both a screen recording from Rviz (with all the relevant topics added for vizualization) as well as a video of what is occurring with the car on the track.
 - (b) *Particle Filter Localization*: The video should show both a screen recording from Rviz (with all the relevant topics added for vizualization) as well as a video of what is occurring with the car on the track.
 - (c) *Pure Pursuit*: The video should show both a screen recording from Rviz with the reference path, pose, target and the steering angle visualized synchronized with the video of what is occurring with the car on the track.

The relevant Rviz vizualization topics are described in their respective sections.

In addition,

Each team member must submit a completely filled out 'Peer Evaluation' form for their team.

>> What do you need to demo ?

During the demo, you will launch your `pure_pursuit` node and Rviz using a single launch file called `pure_pursuit.launch` as part of the `fltenth_purepursuit` package.

This means that all the arguments needed for the `pure_pursuit` node to run properly are passed from the launch file itself and the same launch file launches Rviz showing, the map, the pose of the robot from AMCL, the reference path, the base projection point, the target point, and the steering direction computed by `pure_pursuit`. The launch files for `move_base`, and localization can be separate.

Let the car run multiple laps as we ask some questions about your implementation.

>> Prepare a PDF report with the following:

Q1: **[Mapping]:** An image of the map that you created using ROS cartographer. An Rviz screenshot is fine similar to the example we have provided at the end of Section A. Be sure to add the 'Trajectory >> Path' topic as described in Section [A] later.

Q2: **[Localization and Raceline]** An image (screenshot) of the car being localized by the particle filter on the same map as above (Section B) but also showing the reference path you created using trajectory_builder (Section C).

Q3: **[Pure Pursuit]** How did you arrive at the value of the lookahead distance for pure pursuit ?

>> Detailed Grading:

The total possible points for this assignment are 350

- ☐ [50 pts] PDF Report (10 (mapping) + 20 (localization + raceline) + 20 (lookahead))
- ☐ [20 pts] base_map.pgm file submitted
- ☐ [60 pts] Video Submission (20 (mapping) + 20 (localization) + 20 (pure pursuit))
- ☐ [20 pts] Launch File for Pure Pursuit
- ☐ [130 pts] Pure Pursuit Implementation and Demo - split into
 - ☐ [10 pts] Base Projection Calculation
 - ☐ [10 pts] Target/Goal Calculation
 - ☐ [10 pts] Look Ahead Distance Tuning
 - ☐ [40 pts] Pure Pursuit Steering Angle Calculation
 - ☐ [40 pts] Pure Pursuit Demo
 - ☐ [20 pts] Pure Pursuit Rviz vizualization during the demo
- ☐ [30 pts] Dynamic Velocity Scaling
- ☐ [40 pts] Peer Evaluations

Preparing the Fltenth stack for this assignment










First, we need to install and configure a few launch files on the Fltenth vehicle (**denoted by SSH notation**) as well as the local Ubuntu machine (**denoted by keyword Local Machine**)

On your Ubuntu Local Machine:

Follow the instructions below before you connect to the car as these require an Internet connection.

We will prepare the Local Machine to run cartographer.

1. Open a terminal instance and run the following commands: This is installing a few tools to help build the cartographer package and help us save maps.
 - a. `sudo apt-get update`
 - b. `sudo apt-get install -y python-wstool python-rosdep ninja-build stow`
 - c. `sudo apt-get install ros-melodic-map-server`
2. Now git pull the latest changes from the course labs repository:
<https://github.com/linklab-uva/fltenth-course-labs>
3. Three additional packages have been added to this repo:
 - a. cartographer
 - b. cartographer_ros.
 - c. fltenth_purepursuit

 madhurbehl cartographer packages	
 autoturtle	Added the autoturtle package
 beginner_tutorials	changes
 cartographer	cartographer packages
 cartographer_ros	cartographer packages
 race	rviz test and bag files
 .DS_Store	rviz test and bag files
 .gitignore	Initial commit
 README.md	Update README.md

4. Move these 2 Cartographer packages to the catkin_ws/src folder on the local machine. *This assumes that a ROS catkin_ws already exists on the local machine. If not create that first using catkin_init_workspace.*

5. Your ROS workspace on the Local Machine should look like:

```
catkin_ws >>
    devel
    build
    src >>
        cartogrpaher
        cartographer_ros
    ...
```

6. Now you need to install cartographer_ros dependencies. First, we use rosdep to install the required packages. The command 'sudo rosdep init' will print an error if you have already executed it since installing ROS. This error can be ignored.

From the catkin_ws root folder run the following commands one at a time:

- a. sudo rosdep init
 - b. rosdep update
 - c. rosdep install --from-paths src/cartographer* --ignore-src --rosdistro=melodic -y -r
7. Cartographer uses the abseil-cpp library that needs to be manually installed using this script (from the catkin_ws folder) run :
 - a. src/cartographer/scripts/install_abseil.sh
 - b. sudo apt-get remove ros-\${ROS_DISTRO}-abseil-cpp
 8. Finally Build and install: Regular catkin_make does not work for the ROS cartographer package so we need to use the following command from the catkin_ws root folder:
 - a. catkin_make_isolated --install --use-ninja

Note: From this point onwards, if you want to run catkin_make and build any package in the catkin_ws folder, we will need to use the catkin_make_isolated command since we have copied the cartogrpaher packages within the catin_ws folder.

This is all we need to do on the Local machine, we can now connect to the Fltenth car and start preparing the car for this assignment.

On your Fltenth Racecar (SSH) :

Connect to your car.

Ssh into your team's car and make the following edits:

1. First install the map saver package on the racecar.
 - a. **[SSH Session]** sudo apt-get install ros-melodic-map-server
2. Verify/Edit **base_mapping.launch**:

Be sure to source the workspace first:

```
$ source ~/depends_ws/devel/setup.bash
```

a. In a SSH session: `$ rosed base_mapping base_mapping.launch`

b. Change default value of argument 'car_name' to 'car_X', where 'X' is your car's number. This might already be changed correctly on your car, but verify that it is correct.

3. Verify/Edit **localization.launch**:

a. In a SSH session: `$ rosed particle_filter localization.launch`

b. Verify/Change 'car_name' default value to 'car_X'

c. Verify/Change 'scan_topic' default value to 'car_X/scan'

d. Verify/Change 'odom_topic' default value to 'car_X/odom'

4. Change configurations of VESC board. Open the following file on the car

`~/depend_ws/src/Fltenth_car_workspace/move_base/config/vesc.config`, and update vehicle RPM values. This will lower the limits of vehicle speed, meaning that when you control the throttle from the joycon for teleoperation, the car will be less sensitive to the throttle and will be easier to driver around at low speeds.

- brake_min: -4000.0
- brake_max: 4000.0
- speed_min: -4000
- speed_max: 4000

Triple CAUTION!!!: Be careful while editing the vesc config file. Any unintended edits can break the ability of the car to drive properly.

5. When you performed the git pull from <https://github.com/linklab-uva/fltenth-course-labs> the third package **fltenth_purepursuit** will be copied onto the car.

Copy the fltenth_purepursuit package from your local machine to the catkin_ws/src on the car. You can use sftp or scp for this.

Your ROS workspace on the car (SSH) should look like:

```
catkin_ws >>
    devel
    build
    src >>
        fltenth_purepursuit
        packages you created for other assignments such as 'race'
        ...
```

\$ run catkin_make on the car and build fltenth_purepursuit on the car.

Now you are all set to start the assignment

[A] 2D Occupancy Grid Map - ROS Cartographer SLAM

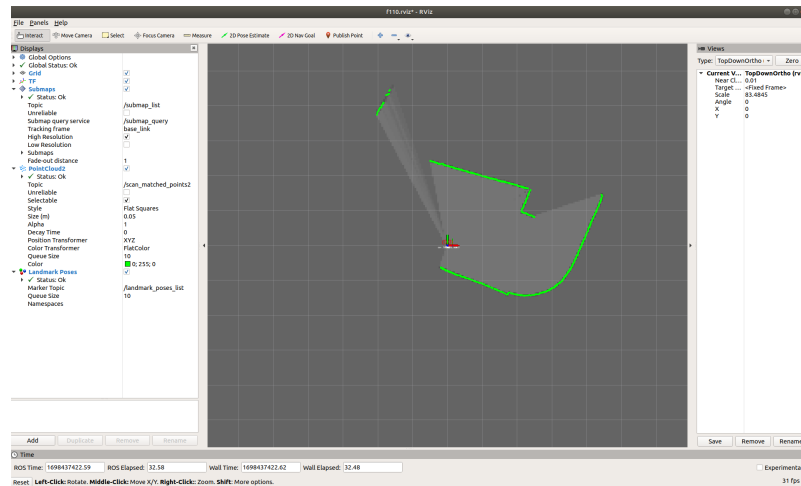
1. Connect to the race car:

```
$ ssh nvidia@192.168.1.1
```
2. Launch the move_base launch file on the car:

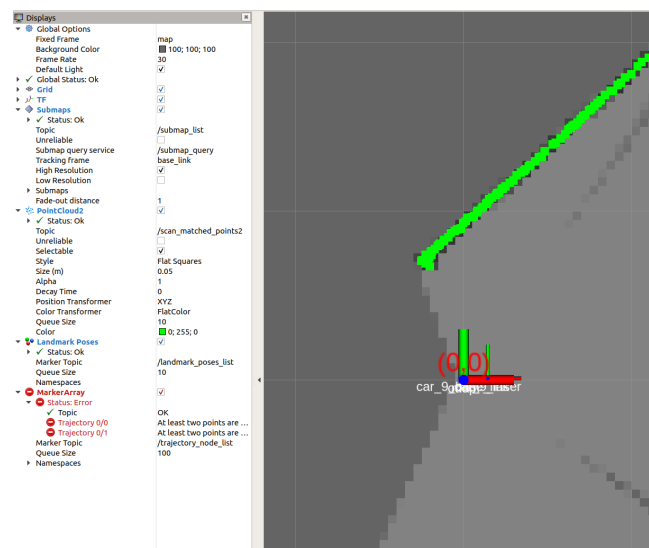
```
$ roslaunch move_base move_base.launch [SSH session]
```

 (Remember to source your env)
3. On the Local Machine, run the following command: (so cartographer will run on the Local Machine)

```
$ roslaunch cartographer_ros fl10_cartographer.launch
```
4. A Rviz window should automatically open on your local machine and you should see something like this:



If you zoom in to the Rviz grid, you can infact see that a coordinate frame (0,0) is placed at the starting point.



5.I Add the following additional topic to Rviz:

Add by topic: 'Trajectory >> Path'

6. On the local machine, in another terminal session launch remote_teleop.launch

`$ roslaunch move_base remote_teleop.launch` **[Local Machine]**

Drive around and build the map: Use the joystick controller to drive the car around track slowly (the VESC cnfig you changed should help with this). You can keep track of the map building progress via Rviz. Avoid making sharp turns, and driving backwards. You can make multiple laps giving cartographer chances for loop closure.

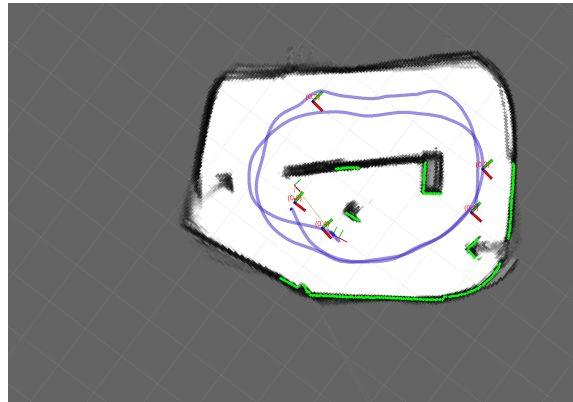


Fig. Your map will look something like this

6. **[Saving the map]** We will now use map_saver on the car to save the map. The ROS cartographer session can continue to run while you save the map. In a new SSH session, we will save the map at a specific location on the car.

First source the depends_ws ROS workspace-

[SSH session]

`$ source ~/depends_ws/devel/setup.bash`

`$ roscd base_mapping/map/`

(if the map folder is not present, you can create it inside the base_mapping folder first)

`$ rosrun map_server map_saver -f base_map`

```
nvidia@tegra-ubuntu: ~ 94x13
631 packages can be updated.
406 updates are security updates.

Last login: Wed Feb 26 23:26:00 2020 from 192.168.1.100
nvidia@tegra-ubuntu:~$ rosrun map_server map_saver -f base_map
[ INFO] [1582760247.072318768]: Waiting for the map
[ INFO] [1582760247.281014064]: Received a 2048 X 2048 map @ 0.050 m/pix
[ INFO] [1582760247.281124592]: Writing map occupancy data to base_map.pgm
[ INFO] [1582760247.498037296]: Writing map occupancy data to base_map.yaml
[ INFO] [1582760247.498375280]: Done
nvidia@tegra-ubuntu:~$
```

Fig.4 Save the map, and name it as “base_map”

The 2D occupancy grid map built by cartographer has been saved in the files base_map.pgm and base_map.yaml in the base_mapping/map/ folder on the car

You can stop the cartogrpaher nodes now and close Rviz.

[B] AMCL Particle Filter Localization

Now we have a map, we will localize the car within that map using **AMCL** particle filter.

1. `$ roslaunch move_base move_base.launch` **[SSH session]** (ignore this command if move_base is already running)
2. `$ roslaunch particle_filter slam.launch` **[SSH session]**

This starts the AMCL package nodes.

- The map (previously saved) is loaded.
- The Odometry message is recognized
- The LIDAR scans are subsscribed.

You should wait until you see the Received Odometry and LIDAR message confirmations as shown below:

```

/home/nvidia/depend_ws/src/particle_filter/launch/slam.launch http://192.168.1.1:11311 94x28
* /rostdistro: kinetic
* /rosversion: 1.12.14

NODES
  /car_9/
    particle_filter (particle_filter/particle_filter.py)
  /
    map_server (map_server/map_server)

ROS_MASTER_URI=http://192.168.1.1:11311

process[map_server-1]: started with pid [4337]
process[car_9/particle_filter-2]: started with pid [4338]
[ INFO] [1582760371.234678960]: Loading map from image "/home/nvidia/depend_ws/src/base_mapping/map/base_map.pgm"
[ INFO] [1582760371.339194256]: Read a 2048 X 2048 map @ 0.050 m/cell
('getting map from service: ', '/static_map')
map service name: /static_map
[ INFO] [1582760373.035833264]: Sending map
Initializing range method: rmgpu
Done loading map
Precomputing sensor model
GLOBAL INITIALIZATION
Finished initializing, waiting on messages...
...Received first Odometry message
...Received first LiDAR message
41

```

Fig.5 received odom and lidar telemetry slam particle filter

3. Open RVIZ on your **[Local Machine]**, and add following topics:

By topic:

- 'map'
- 'laser_scan', change settings: 'size': 0.05; 'Color Trans...': FlatColor; 'Color': preferably Red
- Particle_filter >> viz:
 - inferred pose >> Pose (set its color to green)
 - particles >> posearray

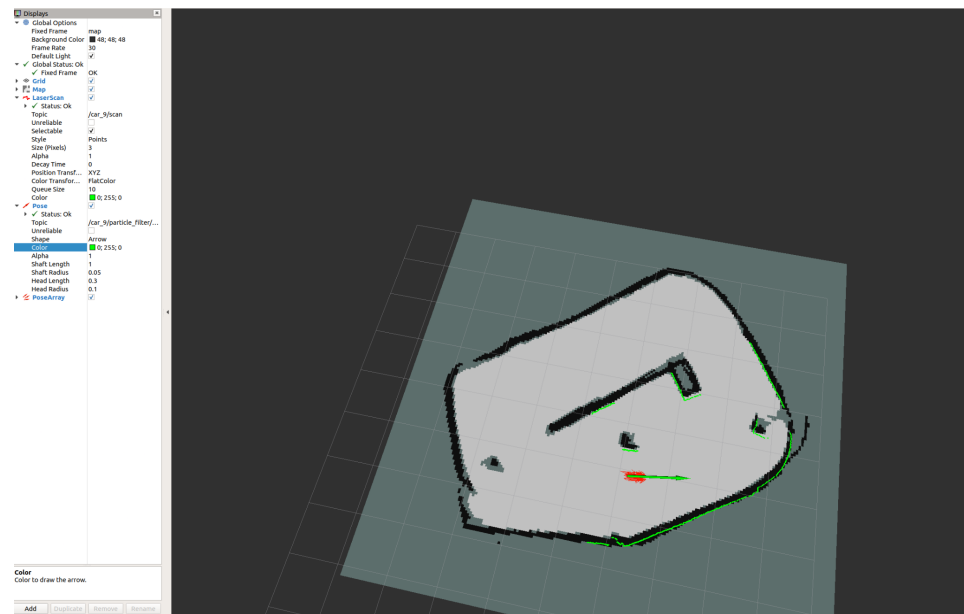


Fig. 6 Settings in RVIZ for AMCL particle filter

3. `$ roslaunch move_base remote_teleop.launch` [Local computer]

Drive the car around slowly, and you should be able to see the pose estimates generated from AMCL. You should also be able to see the particle arrays as well (red arrows).

The AMCL particle filter reports to the pose of your car (**in the map frame**) on the following topic `/car_x/particle_filter/viz/inferred_pose`

[C] Waypoint Logging - Racing Trajectory

Now that the map is built and you can localize the car within the map and obtain its pose, we can start logging a racing line trajectory.

The AMCL particle filter reports to the pose of your car (**in the map frame**) on the following topic `/car_x/particle_filter/viz/inferred_pose`

In order to generate a reference trajectory, the idea is to drive the car around the track roughly at the trajectory you want to take, and simply log the poses reported by the localization node (AMCL).

To do this, we have provided a node called the `trajecotry_builder.py` as part of the `fltenth_purepursuit` package. If you followed the stack preparation instruction, this node should be located inside the `fltenth_purepursuit/utls` folder in the `fltenth_purepursuit` package that you have git pulled previously and build in your `catkin_ws/src/` folder on the car.

You should go through the code for this node, since you need to specify certain paths based on your local machine. Briefly, this node does the following:

- It subscribes to the `/car_x/particle_filter/viz/inferred_pose` topic from AMCL
 - It reads the reported pose of the car
 - Logs new waypoint if the distance from the previous waypoint is greater than the a threshold called path resolution that you can set.
 - The path that you drive can be visualized in Rviz while AMCL is running
 - The logged path is saved in a csv file that is created when you kill this node.
1. Before running this node. Ensure that the path where the csv file will be stored is correctly specified in the node on Line 41. The name of the csv file is provided as an argument when you run the node (more on this below). The path should be such that you save the csv in the `fltenth_purepursuit/path/` folder.
 2. **[SSH Session]** Start the `move_base` node on the car (ignore if already running)


```
$ ssh nvidia@192.168.1.1
$ source depend_ws/devel/setup.bash
$ roslaunch move_base move_base.launch
```
 3. **[SSH Session]** Start particle filter (AMCL) on the car (ignore if already running)


```
$ ssh nvidia@192.168.1.1
$ source depend_ws/devel/setup.bash
$ roslaunch particle_filter slam.launch.py
```
 4. **[Local Machine]** Start the remote teleop node and verify you can drive the car around before starting the `trajecotry_builder` node.


```
$ source depend_ws/devel/setup.bash
$ roslaunch move_base remote_teleop.launch
```
 5. **[SSH Session]** Start trajectory builder. When we run this node we need to provide 3 arguments to the node:
 - a. Argument 1 is the `<car_X>` where X is your team number e.g. `car_9`

- b. Argument 2 is the csv filename where logged points will be stored - this file will be created by the trajectory_builder itself and stored at the path specified in line 41.
- c. Argument 3 is a Boolean flag to indicate whether you would like to visualize the logged path as it is being generated - you can set this to true or false. If set to true, you will see a topic /trajectory_builder/visualize_path created that can be added to Rviz as you drive around.

putting this all together:

```
$ ssh nvidia@192.168.1.1  
$ source depend_ws/devel/setup.bash  
$ rosrun f1tenth_purepursuit trajectory_builder.py car_9 raceline true
```

Now drive a loop around the track using the remote teleop, then kill the trajectory_builder node - the csv file is created upon node exit.

[D] Implement the Pure-Pursuit Trajectory Following Method

Now you will implement the pure pursuit trajectory following method as discussed in the lectures.

We have provided template code for a node called `pure_pursuit.py` that can be used as a starting point for this part.

This code is heavily commented and you should carefully review the template code for detailed instructions. In this document, we only list the high level things that you need to do:

1. First, modify Line 36 in the `construct_path()` function to point to exactly where the csv file created by `trajecotry_builder` is located.
2. TODO 1 - Using the reference path, find the `base_projection` of the car on the reference line.
3. TODO 2 - Fine tune the lookahead distance (default is set to 1m)
4. TODO 3 - Using the base projection point and the lookahead distance, compute the target/goal point for pure pursuit.
5. TODO 4 - Implement the pure pursuit calculations for computing the steering angle using the pose of the car (x,y, heading), and the coordinates of the target point.
6. TODO 5 - convert the computed steering into a value between `[-100,100]` to be sent to the `steering_ablge` command.
7. TODO 6 - Implement dynamic velocity scaling

Once you fill in the missing pieces, this node can be run using the following:

```
$ ssh nvidia@192.168.1.1
$ source depend_ws/devel/setup.bash
$ rosrn fltenth_purepursuit pure_pursuit.py car_9 raceline
```

Note that `pure_pursuit.py` also takes 2 arguments as inputs:

- `car_x`: your team number
- `<name of the csv file containing the reference path>`

Finally, this node will also publish a topic for you to visualize the pose of the car, the base projection, the target point etc. in Rviz (see the template for details). You should run Rviz to and add the following topic to visualize `pure_pursuit` points: `/car_x/purepursuit_control/visualize`.

Note: While its good to run slowly while debugging, remember we purposely limited the VESC to aid with mapping. You should set the vehicle RPM back to default values in the `vesc.config` file to go faster:

- `brake_min`: -20000.0
- `brake_max`: 20000.0
- `speed_min`: -20000
- `speed_max`: 20000