NDT-AMCL Design

Navigation 2 Working Group

July 2020

1 Purpose

This file proposes a software design for the implementation of the new localization module of ROS 2's Navigation Stack (nav2). It has been done with the implementation of a vanilla Normal Distributions Transform MCL (NDT-MCL) [9] in mind, as well as the future integration of possible improvements. This design is meant to be modular and easy to update and test.

The document will introduce the required main modules to implement a localization framework in Section 2, expose and explain the proposed design given those requirements in Section 3 and discuss potential improvements and how they might be integrated using the mentioned modules in Section 4.

2 Requirements

ROS 2 is in need of a new standard for Light Detection and Ranging (LiDAR)-based 2D and 3D localization. After a previous survey [8], NDT-MCL was chosen as the alternative to be implemented and, if needed, modified to include improvements in the matching and/or in performance in dynamic scenarios.

Our implementation of NDT-MCL will be designed as a highly modular localization framework, whose main components will be:

- **Solver Module**: To track the estimated state of the robot. For NDT-MCL, it will be the Particle Filter (PF) based Monte-Carlo Localization (MCL). Having this independent module will make it possible to use approaches based in Kalman Filters (KFs) or graphs in the future.
- **Matching Module**: It will match the sensor measurements (LiDAR scans, images, etc.) with the map. The first implementation will be based in Normal Distributions Transform (NDT) maps and its first upgrade will be to be able to switch between 2D and 3D matching. The existence of this module will allow the usage of another sensors or different scan-based matchers, such as some of the potential improvements proposed in Section 4.
- **Map Module**: The model of the environment. It will provide with options for processing the surroundings in one or another way, such as a 2D or 3D grid-map, a point cloud, etc. It also makes this design applicable for Simultaneous Localization and Mapping (SLAM) applications, where more interaction with the map is needed.
- **Motion Module**: It will dictate how to modify the state (PF's particle states, KF's belief, etc.) according to the odometry and the motion model that is being used (ackermann, bicycle, differential, omnidirectional etc.).

3 Software Design

This section will define which classes will be needed (and how will they interact) given the modules exposed in Section 2 and focusing on an NDT-MCL implementation.

The main class will be the **Solver** interface's plugins (*MCL* here), which will contain a particle filter, a map, a matcher and a motion model (each of them will have an independent interface to use different plugins). It will

also provide with the three steps of an MCL algorithm (*sample()* the map, *update()* the particles' positions and *resample()* using the new measurement and what each particle should have perceived). The **PF** will be composed of a collection of particles (going from 1 to infinite) and methods that will allow performing all the necessary actions for its usage for localization purposes, such as sample a given map (*sample_map()*), apply a certain motion to all particles (*apply_motion()*), compute the importance weights of the particles (*compute_weights()*), decide which particles will be killed (*purge()*) and perform the re-sampling step to reach the desired number of particles again (*resample()*). For some of these functions, the ***Particle*** class will provide a method to apply the action in an specific particle.

Figure 1 shows the classes and its attributes (listed with a -) and methods (listed with a +).

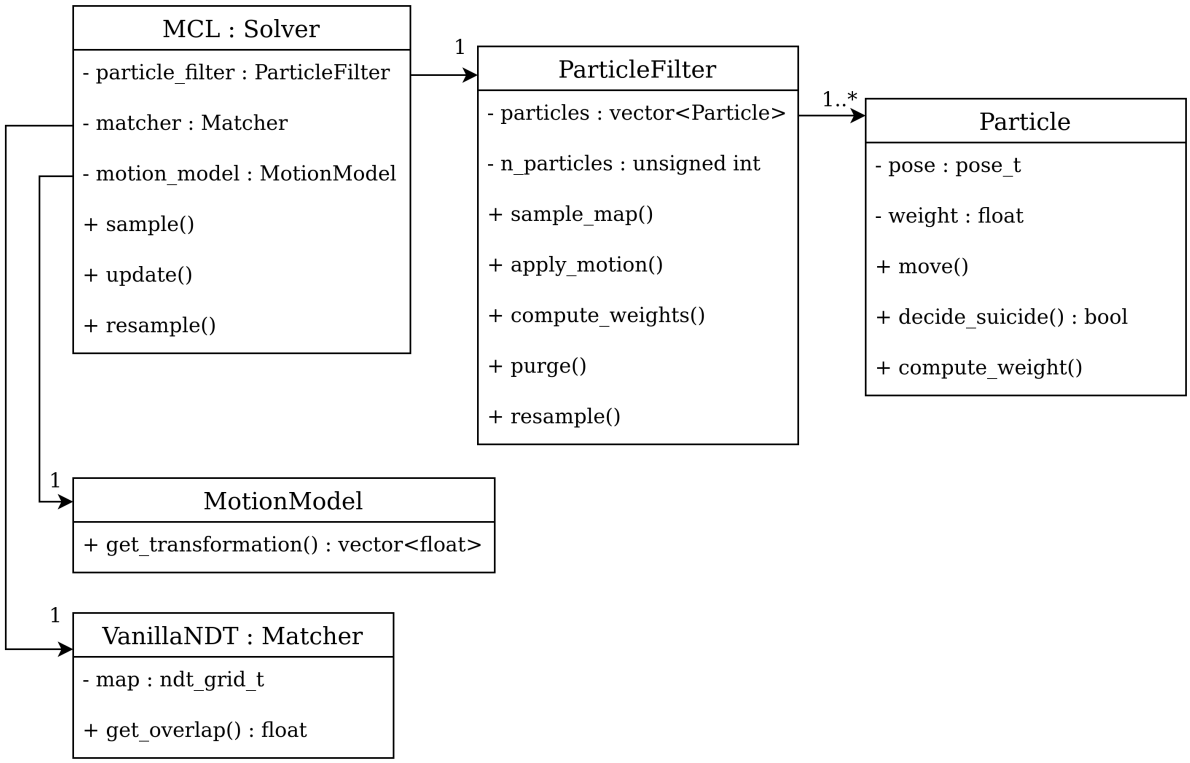

Figure 1: Design diagram for an upgradable/modifiable NDT-MCL implementation.

Regarding the implementation of each of the modules listed in Section 2, ***pluginlib*** [2] and interface classes will be used to allow to load the modules dynamically as plugins. Each module will be implemented as an interface and each plugin (e.g. different matchers, such as the vanilla one, vanilla with beam skipping, vanilla with DT or IRON, etc.) will be called in run time. Further details about the implementation of each module is listed next:

- **Solver Module**: It will start having only the MCL plugin, containing mainly the state estimator (a PF), but also the rest of relevant elements for the localization (map, matcher and motion model) and methods for the localization..
- **Matching Module**: Implemented in the *Matcher* class. It will provide a measure of how overlapped the current measurement is with what a particle should perceive from the map. The Point Cloud Library (PCL) [3] can be useful for this purpose given the methods it provides for point cloud registration. For further ampliations to support other sensors such as cameras, OpenCV [1] would be very helpful for the keypoint extraction, description and matching.
- **Map Module**: In the first version, a subscriber to the map topic in the main application can do the job, but a specific object can be useful for future (and more complex) usages of this design, such as in SLAM implementations, where publishing and subscribing the map may be needed.

- **Motion Module**: The *MotionModel* will provide a method to transform the current pose into the new one given the odometry.

4 Possible Future Improvements

- Apply **beam skipping** to increase the robustness against small dynamic objects (such as persons). If many particles (a certain percentage stablished in the code) are not available to match a certain LiDAR beam, it will not be considered. This can be added in the Matching Module.
- Usage of the dual timescale proposed in **DT-NDT-MCL** [11], which may help improving the performance on environments with medium sized dynamic environments. **IMPORTANT**: The usage of a local submap, even it may help for the purpose it was conceived for, may introduce worse behaviors whenever the system gets delocalized or drifts. This will belong to the Matching Module.
- The **IRON** detector and descriptor [10] are NDT-specific and are supposed to increase the matching rate. This would be a modification of the Matching Module.
- The usage of **ROS Components** [4] instead of the previously mentioned plugins, which has the potential to provide more modularity and avoid missing sensor scans or readings whenever ROS 2 improves its performance. Different resources regarding components are:
  - Comparison between ROS Components and Plugins for ROS 2 [7].
  - Example Base class using static inheritance (CRTP) [5].
  - Example Derived classes using the above mentioned base class [6].
- Add support for **multiple sensor scans** (e.g. Autonomous Vehicle (AV) with several LiDAR sensors). The Sensor Module (*SensorScan* class) could allow the possibility of having a collection of point clouds and a method to fuse them to provide the sensor scan that each particle (or KF) will use for the matching.

5 Contact

This contribution was done by LI Xin and TORRES CÁMARA José M., while mentored by MACENSKI Steve.

Acronyms

AV Autonomous Vehicle. 3

KF Kalman Filter. 1, 3

LiDAR Light Detection and Ranging. 1, 3

MCL Monte-Carlo Localization. 1, 2

nav2 ROS 2's Navigation Stack. 1

NDT Normal Distributions Transform. 1, 3

NDT-MCL Normal Distributions Transform MCL. 1

PCL Point Cloud Library. 2

PF Particle Filter. 1, 2

SLAM Simultaneous Localization and Mapping. 1, 2

References

[1] Opencv library.

[2] pluginlib's page in ros wiki.

[3] Point cloud library.

[4] Ros 2 code composition with ros components.

[5] Jeremie Deray. Base class using static inheritance (crtp).

[6] Jeremie Deray. Base class using static inheritance (crtp).

[7] Jeremie Deray. Comparison between components and plugins in ros 2, 08 2019.

[8] Navigation 2 Working Group. Potential amcl improvements.

[9] Jari Saarinen, Henrik Andreasson, Todor Stoyanov, and Achim Lilienthal. Normal distributions transform monte-carlo localization (ndt-mcl). pages 382–389, 11 2013.

[10] Thomas Schmiedel, Erik Einhorn, and Horst-Michael Gross. Iron: A fast interest point descriptor for robust ndt-map matching and its application to robot localization. pages 3144–3151, 09 2015.

[11] Rafael Valencia, Jari Saarinen, Henrik Andreasson, Joan Vallvé, Juan Andrade Cetto, and Achim Lilienthal. Localization in highly dynamic environments using dual-timescale ndt-mcl. 05 2014.