A Comparative Study of Algorithms towards an Autonomous F1/10-Car

Jens de Hoog*, Thomas Huybrechts*§, Peter Hellinckx*§
jens.dehoog@student.uantwerpen.be, thomas.huybrechts@uantwerpen.be, peter.hellinckx@uantwerpen.be

§Department of Applied Engineering, Electronics-ICT, University of Antwerp, Belgium
§IDLab, Department of Applied Engineering, University of Antwerp - iMinds, Belgium

***Abstract*—Insert abstract of the paper.**

—

I. Introduction

II. **Testopstelling**

A. F1/10-car

Ik vertel over de wagen in het algemeen en welke sensoren eropzitten. Dit gebeurt heel kort. Ook vertel ik over de globale structuur van ROS en welke belangrijke nodes er gebruikt worden.

B. Prerequisites

In dit deel vertel ik over het traag rijden en het rijden tegen constante snelheid.

- Slow accurate driving: er wordt een beetje aandacht besteedt aan de twee approaches en waarom ze wel/niet werkten
- Ik vertel kort waarom het rijden op softwareniveau (met IMU) niet werkten en hoe het met de RPM-sensor is opgelost.

III. Algorithms

A. Global

The algorithms can

B. Fixed path

a) Literature: Quite an amount of research has been done in planning a particular route and navigating on that route. Henson et al. [1] propose an algorithm which processes waypoint instructions by calculating a route between each point. This algorithm is optimised for cars with an Ackermann-steering mechanism. Thus, if the waypoint is reached but not with the exact orientation, the car starts to manoeuvre until the positioning is correct.

Theodosis [11] proposes multiple methods, all of which use waypoints. The checkpoints are connected with racing lines such that a car-like robot can easily navigate without impossible pivot points.

In the ROS-framework, a complete navigator package is available, which is also known as the *Navigation Stack*. This concept calculates a path and velocity commands based on given information, such as odometry and data from rangefinders. A drawback of this navigator is the fact that it is created for robots with holonomic and differential steering mechanisms. Though, it is usable for car-like robots as planner implementations exist for Ackermann-steering mechanisms.

For this research, the navigation stack has been implemented. It is easy to set up on the F1/10-car as the software on the car is already running a ROS-environment.

b) Navigation Stack: As mentioned in the previous paragraph, the navigation stack is found An implementation is found in the ROS-framework and is realised using multiple nodes and packages. One of these nodes is the *move_base*-node, which has a major role in the system. This node provides the linkage between the global and local planning algorithms that are used to calculate a path from the start location to the destination. Other nodes outside the package provide the inputs and outputs of the *move_base*-node. The needed inputs are the odometry data, transformations, localisation information, the map on which the navigation would take place and the data from a certain rangefinder. For this research, a LiDAR-sensor (Light Detection and Ranging) served as rangefinder. The commands by which the car would drive, are the generated outputs of the central navigation node. Figure 1 displays an overall overview of the different nodes in the navigation stack [7].

Inside the *move_base*-node, multiple calculations take place in order to drive the car. First, a plugin called *global_costmap* produces a costmap which is used to plan the route from start to destination. This is done by the plugin *global_planner*. The default implementation of this planner is *navfn*, but can be replaced by the more advanced *global_planner*. For this research, the more advanced planner is used because of the ability to customise multiple parameters. Additionally, the central node produces a local costmap which is used by the local planner. By using this particular costmap and planner, the navigation stack can plan a route around an unexpected obstacle that is not present on the map. In addition, the *move_base*-node includes a recovery plugin which plans a recovery route when the car is stuck and thus unable to follow the planned route. Figure 1 shows an abstract overview of the navigation stack with its nodes and plugins. [8], [13]

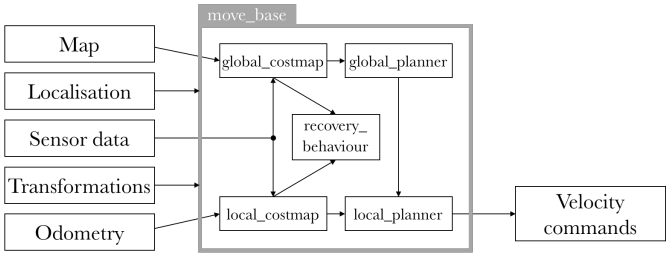

Fig. 1. Overview of the navigation stack with its input and output nodes, along with the plugins inside the *move_base*-node.

Before the car can drive using the navigation stack, one has to make sure a map of the environment is present. The map used in this research is built by using *hector_mapping*. This package is an implementation of the SLAM-concept (Simultaneous Localisation And Mapping), which only uses data from the LiDAR-sensor. Therefore, this is the most suitable implementation for our F1/10-car because of the lack of odometry data. More information about *hector_mapping* can be found in [2] and [5].

In order to use the *move_base*-node properly, certain precautions have to be made. First, the transformation tree has to be correct such that the laser data can be transformed to the center of the robot. To build such a tree, two *tf*-nodes are started. The first node transforms the LiDAR-data to a link that connects all types of sensors. The second node transforms the linking point to the center of the frame. Since only one sensor is present, we located the linking point to the center of the frame; thus, the second transformation is one-to-one. Though, this second transform is needed as the navigation stack needs the *base_frame*-transformation. More information about transformations can be found in [6] and [12].

Second, the navigation stack needs a source of odometry data. Since the car does not feature a suitable source, we have to generate such data from the LiDAR-sensor. *hector_mapping* can provide odometry data which is derived from the laser scan. In order to use this data in the navigation stack, a particular node is needed which translates the odometry information to transformation data. The implementation of this node can be found in the *f1tenth*-repository of Nischal K.N. [4].

The third precaution is the provision of localisation data such that the robot can guess where it is located on the given map. This is implemented by the *AMCL*-node (*Adaptive Monte Carlo Localisation*). In order to generate a suitable pose guess, the node also needs odometry data, along with the map, transformations and LiDAR-information. More information abount *AMCL* can be found in [3] and [10].

The last precaution is the translation from the velocity commands of the navigator to instructions for the motor controller. As the Figure 2 shows the transformations created

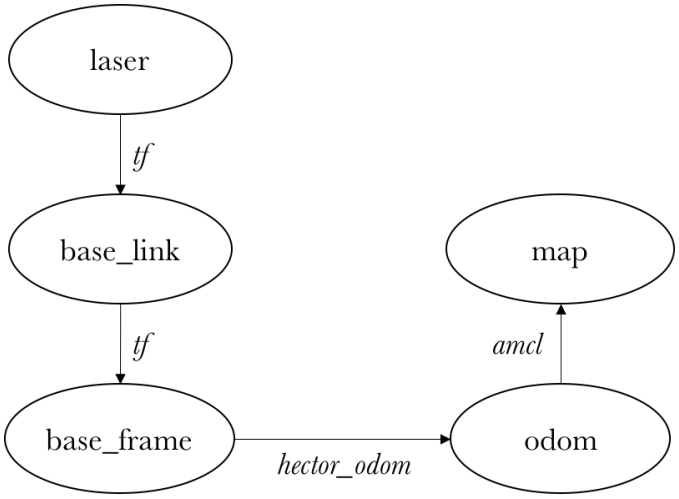

Fig. 2. Created transformation frames for the navigation stack. The *tf*-links between *laser*, *base_link* and *base_frame* are static transforms. Next, the *hector_odom*-link is created by a node which translates the odometry of *hector_mapping*. The *amcl*-transform is created by the localisation provider.

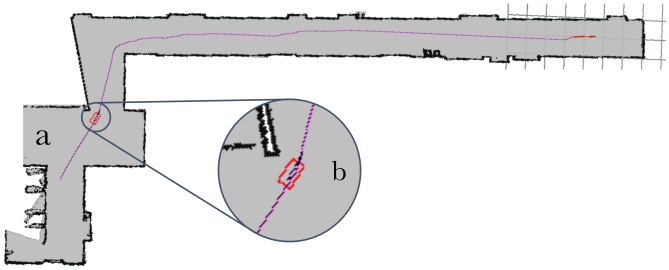

Fig. 3. a) shows the route which is planned by the navigation stack. The map is built with *Hector_mapping*. b) displays a close-up of the car moving on the route.

by the aforementioned nodes.

c) Results: The planned route is shown in figure 3, along with a close-up of the car navigating on that path.

A major problem of the navigation stack is that it is not optimised for car-like robots. Instead, it is especially created for robots with holonomic or differential steering mechanisms. Therefore, the car is sometimes unable to follow the calculated path. Though, if the is drifting from its global route because of the steering mechanism, the local planner corrects the route such that the car is back on track. Also, another local planner is available for car-like robots. This planner is also known as *teb_local_planner*. More information can be found in [9].

C. Waypoints

Literature
: In dit deel komen verschillende algoritmes naar boven die in de literatuur te vinden zijn. Ook wordt er gezegd waarom deze algoritmes niet goed genoeg zijn voor mijn doel en dat ik daarom zelf een algoritme heb ontwikkeld.

Concept
: Deze paragraaf legt het algoritme uit, samen met alle edge cases (wat als een muur wegvalt, wat als ik een kruispunt tegenkom, ...).

Resultaten
: Dit deel beschrijft de resultaten. Hierin komt ook aan bod dat het algoritme in het algmeen goed werkt, maar dat de LiDAR-sensor gevoelig is aan de soort reflectie (zwarte voorwerpen hebben minder reflectie) en dat er daardoor veel afstelwerk vereist was. Dit komt ook aan bod in further research.

D. Optimisation Algorithm

Literature
: Ik leg verschillende algoritmes naast elkaar en maak een kleine vergelijkende studie. Ik leg ook uit waarom ik een bepaald algoritme heb gekozen.

Concept
: Het gekozen algoritme wordt in meer in detail besproken.

Resultaten
: De resultaten van dit algoritme wordt besproken.

IV. Comparative Study

De hierboven voorgelegde approaches worden met elkaar vergeleken en er worden verschillende toepassingen opgesomd waarin elk algoritme faalt of uitblinkt.

V. Conclusion

De conclusie wordt gemaakt.

VI. Further Research

In dit deel wordt kort besproken wat er allemaal nog gedaan kan worden van research. Er kan bijvoorbeeld gekeken naar worden optimalisaties voor de Navigation Stack en meer specifiek voor Ackermannsteering. Ook kan er gekeken worden naar betere herkenning van hoeken en inhammen voor het tweede algoritme. Als laatste zouden er betere optimalisatie-algoritmes onderzocht kunnen worden, en met of zonder neural network.

REFERENCES

[1] G. Henson, M. Maynard, G. Dimitoglou et al. Algorithms and performance analysis for path navigation of ackerman-steered autonomous robots. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, pages 230–235, New York, NY, USA, 2008. ACM.

[2] S. Kohlbrecher, O. Von Stryk, J. Meyer et al. A flexible and scalable SLAM system with full 3D motion estimation. *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011*, pages 155–160, 2011.

[3] B. P. Gerkey. amcl - ROS Wiki. http://wiki.ros.org/amcl, 2016. [Online; accessed May 2017].

[4] Nischal K.N. Repository: 1/10th Scale Autonomous Race Car. https://github.com/nischalkn/F1tenth, 2016. [Online; accessed May 2017].

[5] S. Kohlbrecher. hector_mapping - ROS Wiki. http://wiki.ros.org/hector_mapping, 2012.

[6] S. Kohlbrecher. How to set up hector_slam for your robot - ROS Wiki. http://wiki.ros.org/hector{_}slam/Tutorials/SettingUpForYourRobot, 2012. [Online; accessed April 2017].

[7] E. Marder-Eppstein. move_base - ROS Wiki. http://wiki.ros.org/move_base, 2016. [Online; accessed May 2017].

[8] E. Marder-Eppstein. nav_core - ROS Wiki. http://wiki.ros.org/nav_core, 2016. [Online; accessed May 2017].

[9] Christoph Rösmann. teb_local_planner - ROS Wiki. http://wiki.ros.org/teb{_}local{_}planner, 2016. [Online; accessed May 2017].

[10] S. Thrun, W. Burgard, D. Fox. *Probabilistic Robotics*. 1999.

[11] Paul Alan Theodosis. *Path Planning for an Automated Vehicle using Professional Racing Techniques*. Degree of doctor of philosophy, Stanford University, 2014.

[12] W. Woodall. Setting up your robot using tf - ROS Wiki. http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF, 2015. [Online; accessed April 2017].

[13] K. Zheng. ROS Navigation Tuning Guide. Technical report, 2016.