Local planning methods for autonomous navigation on sidewalks: a comparative survey

Daniela Gómez-Ayalde * and Victor A. Romero-Cano †
Robotics and Autonomous Systems Laboratory
Faculty of Engineering, Universidad Autónoma de Occidente, Cali, Colombia
*daniela.gomez_ayalde@uao.edu.co, †varomero@uao.edu.co

***Abstract*—This paper presents a comparative survey of autonomous navigation systems for mobile robots on sidewalks. The kinds of systems mentioned above endow robots with the capability of estimating an optimal trajectory that once followed, allows the robot to move autonomously from a current pose to a target pose on a sidewalk. In addition, they allow avoiding both static and dynamic obstacles reliably and efficiently. An autonomous navigation system functions on data from multiple sensors and a global representation of the environment in which it is located, and allows the robot to perform its motion task satisfactorily without leaving the sidewalk on which it is moving.**

It is important to mention that an intense search of the state of the art was made around the existing autonomous navigation algorithms. Finally, different tools are mentioned such as ROS2 middleware, the Gazebo simulation program and the Rviz2 visualization program.

***Index Terms*—Autonomous navigation, sensors, obstacle avoidance, sidewalks, ROS, Gazebo, Rviz.**

I. Introduction

Mobile robots are being used for a large number of applications due to their ability to move on different surfaces and adapt to different situations, applications such as space exploration, transportation, mining, agriculture, housework, home delivery, among others [1]. These robots have sensors and algorithms that allow them to detect and perceive their environment, make decisions and execute them to fulfill assigned tasks. However, it is important to clarify that these sensors and algorithms must be adapted to the environment in which the robot must work. Many of the mobile robots face a wide variety of challenges, mainly because they move in dynamic environments where a large number of variables and aspects to take into account influence. Especially if they move on the sidewalks of a city.

This is because these robots can find themselves with different situations, which they do not find in other types of places. In addition to everything that can be found on a sidewalk, such as people walking in opposite directions, lamp posts, traffic signals, traffic lights, animals and some objects such as tables or chairs, among others, there is a very important aspect that a sidewalk has that does not have, for example, a shopping center or a university campus, and it is the reduced space through which the robot must move, since it is of vital importance that it does not leave the sidewalk when it continues its trajectory.

978-1-6654-7470-2/22/$31.00 ©2022 IEEE

Therefore, the need arises to develop an autonomous navigation system that allows a mobile robot to move from point A to point B (target position) autonomously, estimating a trajectory and detecting and avoiding any type of obstacle, taking into account the environment in which it is going to move, which is, in this case, an outdoor environment made up of a sidewalk.

This process includes both the planning of a trajectory, as well as the detection and avoidance of obstacles in a reliable and efficient way, for which the trajectory must be constantly updated, due to the fact that there is no prior knowledge of the obstacles that may arise during the robot's route, letting it make decisions with the help of the information it receives from its sensors and the processing of said information.

The problem of autonomous navigation in populated areas has been studied many times in the past, especially in indoor, non-urban outdoor environments or on roads. However, relatively few studies have been carried out in urban environments where there are large numbers of people or dynamic objects, such as city centers or sidewalks, in which they face a large number of similar challenges. In [2], an autonomous sidewalk navigation system for mobile robots is proposed, taking into account pedestrian flows in real time to imitate their behavior and navigate all towards the same goal. In [3], an autonomous navigation system in urban centers is described, in which a SLAM approach is used for learning maps of urban areas and subsequent location on them, as well as dynamic obstacle avoidance. A similar approach is developed in [4], in which a navigation system for a robot in unknown and dynamic environments is created, using the information extracted from its interaction with people and its perception system.

II. Challenges

For autonomous navigation on sidewalks, it must be taken into account that the mobile robot faces two major challenges: the first, moving only on the sidewalk, that is, not leaving to the road, residential or commercial areas; and the second, to navigate autonomously, without human intervention and successfully reaching the target pose, without colliding with obstacles. The way to tackle both approaches will be developed throughout this work. However, it is important to mention first that, for the robot to satisfactorily fulfill its task, it is very important to have sensors that allow it to obtain

the necessary data for subsequent processing and decision making. A mobile robot can have a wide variety of sensors depending on the function for which it is built. However, there are three types of sensors that are very important for the task of autonomous navigation on sidewalks:

A. LiDAR

A Light Detection And Ranging (LiDAR) sensor allows to measure the distance between the sensor and a person or object. For this, the sensor emits a beam of light into the environment. That light beam is then reflected by the object and returns to the sensor. To calculate the distance, the sensor measures the time it takes for the pulse to go back and forth, taking into account the speed of light. It is important to note that light travels at a high speed, approximately 300.000 km/s, which means that the device responds quickly. The laser, for its part, delivers points in 2D (a single plane) or in 3D (three-dimensional reconstructions), which allow knowing how far away those physical elements are that caused the light beam to return to the sensor. The data measured by the sensor can be used to find the closest points and thus, for example, make the robot stop in front of an obstacle.

B. IMU

The Inertial Measurement Unit (IMU) sensor is an electronic device capable of measuring the speed, orientation and gravitational force of an object using the combination of three sensors: accelerometer, gyroscope and magnetometer. The accelerometer is used to measure inertial linear acceleration in m/s^2, the gyroscope to measure angular velocity in rad/s, and the magnetometer to measure magnetic field strength in mGs or μT, which can improve the gyroscope reading. This sensor is very important because it allows you to obtain information about the robot's odometry, which means that it allows you to obtain precise information about the robot's pose and speed based on its movement.

C. Camera

The camera allows to obtain information and images of the environment in which the robot is. A commonly used type of camera is the stereo camera, which is made up of two fixed monocular cameras, side by side, taking images simultaneously. These two images can be further processed to obtain the distance (depth) at which the objects captured in the photograph are located. For this reason, these types of cameras are used to build three-dimensional representations of the captured space.

III. Background about Autonomous Navigation

To tackle the challenge of autonomous navigation in this paper, the ROS2 Navigation Stack [7] will be taken into account, which contains all the necessary tools to successfully achieve the goal. An autonomous navigation system must necessarily be composed of the three navigation servers: planners, controllers and recoveries. Next, a description of each of them will be given.

A. Planners

The main task of a planner is to calculate an optimal path to go from point A (current pose) to point B (destination pose), avoiding obstacles along the way. In this way, the robot can successfully move towards its target position, with the help of the information from its sensors and a global representation of the environment in which it is located. There are various planners used in autonomous navigation tasks, which differ in the implemented algorithm and the type of robot in which they can be used. The planners mentioned below operate on a costmap, using occupancy grids.

To correctly choose the planner to use, the type of mobile robot in which the navigation system is going to be implemented must be taken into account, as mentioned above. The recommended planners for circular differential and circular omnidirectional robots (holonomic robots), which can drive in any direction or safely rotate on their axis, are the following:

- **NavFn Planner:** The NavFn planner uses the A star (A*) or Dijkstra algorithms to find the path between the current pose and the target pose. Dijkstra's search algorithm guarantees to find the shortest path between the two poses mentioned above, using the actual cost (in distance or time) it takes to go from the starting point of the trajectory to each of the points connected to it, at equal to the cost between each of the points that make up the trajectory. With the above, it is guaranteed to find the route with the lowest cost. On the other hand, the search algorithm of A* uses a heuristic function that takes into account not only the actual cost it takes to reach each of the points of the trajectory from the initial point, but also the estimated cost from each one of them towards the target pose. In this way, priority can be given to the points of the trajectory that have a lower estimated cost towards the target pose, making the search faster and more efficient depending on the environment in which the robot is located [5].
- **Smac Planner 2D:** Smac Planner 2D starts with the implementation of the cost-conscious, highly optimized, and reconfigurable A* search algorithm that supports Moore and Von Neumann models. Although this 2D A* planner is a bit slower, it is important to highlight the higher quality of the routes. Also, this planner provides a path between neighboring vertices of 4 or 8 connections. Said route may have small zig-zags to reach another course that is not 90° or 45°, which will be the points taken into account by the controller to follow the trajectory. However, the robot's final movement does not reflect that mentioned zig-zag behavior.
  The 2D A* algorithm is responsible for the search of a graph of nodes, which contain the necessary methods to calculate the heuristics (explained above), the search neighborhoods and the travel costs between the different points of the trajectory. In the 2D Smac planner, a 2D node template is provided by default that searches a 2D

grid to make connections between neighboring points [6].

On the other hand, there are non-holonomic robots, such as Ackermann-type robots or legged robots, which have rolling restrictions. The recommended planner for this type of robots is the following:

- **Smac Hybrid-A* Planner:** The Smac Hybrid-A* planner implements the Hybrid-A* algorithm, which is an optimized and reconfigurable extension of the A* algorithm for non-holonomic robots, supporting the Dubin and Reeds-Shepp models. This algorithm expands the possible robot trajectories while considering the constraint of the robot's minimum turning radius and the total footprint of the robot to avoid collisions [7]. This algorithm is useful for planning with curvature constraints, such as when planning a robot at high speeds to ensure that it does not tip over or run out of control [6].
  Therefore, this algorithm is designed for robots that can only make orientation changes while moving between vertices. Due to this, the trajectory generated by the algorithm can include curved trajectories to reach the target pose, that is, smooth trajectories (without sudden changes in direction) that satisfy the dynamics of the robot and that are kinematically feasible. This is due to the fact that non-holonomic robots may have difficulty following acute angles between trajectories when they make the necessary orientation changes to accurately and precisely follow the final route.

B. Controllers

The main task of a controller is to calculate a valid control effort to follow the trajectory globally calculated by the planner or to complete a local task. For this, the controller performs route planning from the current pose to a few meters ahead (up to the sensor range). In this way, it builds a trajectory to avoid dynamic obstacles, which are not on the map, but can be detected with the help of sensor data on the local costmap. This is possible because the controller has access to a representation of the local environment to try to calculate the feasible control efforts to follow the global plan [7].

The controllers used in autonomous navigation are the following:

- **DWB:** The DWB controller implements a Dynamic Window Approach (DWA) algorithm, modified with plugins that are configured to calculate the commands that control the robot. This controller implements Critic plugins, which allow users to specify new critical functions to use in the system, and Trajectory Generation plugins, in charge of generating the set of possible trajectories of any shape that the robot can follow, which are later evaluated by one or more critical plugins, determining an overall score for each of them and choosing the one with the best score. This path will determine the speed of the output command [8].
  The DWB controller is mainly used in circular or non-circular differential and circular or non-circular omnidirectional robots, and its main task is dynamic obstacle avoidance [7].
- **Timed Elastic Band (TEB):** The TEB controller is an optimal model predictive control (MPC) timing controller, which implements the Timed Elastic Band approach. This approach optimizes the robot's trajectory based on its execution time, distance to obstacles, and feasibility with respect to the robot's kinematic constraints [7]. This approach can be used in holonomic and non-holonomic robots, which means that it can be used in both differential and omnidirectional robots as well as Ackermann and legged robots. By considering the temporal information, TEB explicitly considers and controls the velocities and accelerations of the robot.
- **Regulated Pure Pursuit (RPP):** The RPP algorithm can be considered as the best trajectory algorithm among the available variations of Pure Pursuit algorithms (Normal Pure Pursuit and Adaptive Pure Pursuit), and is implemented to meet the needs of consumer, industrial or service robots, being suitable for use with differential, Ackermann and legged robots [7].
  This algorithm implements active collision detection, especially for confined or small spaces, by using a parameter that sets the maximum time allowed before the robot collides with a possible obstacle, which is used in the current speed command. In addition, it implements linear velocity regulations based on cost functions, which include curvature scaling. Curvature scaling is responsible for creating an intuitive behavior that allows the robot to slow down when it makes sharp turns, when it encounters sudden changes in direction, when it is close to a possible collision with an obstacle, and when it is in partially observable environments where it must turn into an unknown and dynamic environment, which helps make operations much safer. In this way, in environments where the curvature of the trajectory is very high, the linear speed of the robot drops and the angular speed takes over to turn towards the heading [9].
  Kinematic speed limits are also implemented in both linear and angular speeds, when moving to follow the trajectory and when performing pure rotations. This ensures that the output commands are safe and kinematically feasible [9].

C. Recoveries

The main task of recoveries is to allow the robot to deal with different problems and get out of bad situations autonomously, without the need for human intervention. A bad situation can be, for example, a failure in the perception system, which would cause a wrong representation of the environment with obstacles that do not exist; encounter unknown situations or get stuck due to the movement of dynamic obstacles [7]. In order for the robot to be fault tolerant, a task server is implemented that allows the robot to perform a certain behavior depending

on the situation in which it finds itself. Currently, there are three types of recovery behaviors:

- **Spin:** The robot performs a rotation about its own axis, according to the given angle. This angle can be set. This can allow the robot, for example, to get out of a confined space into a free space where it can successfully navigate.
- **Back Up:** This behavior is useful when the robot is stuck or in case of a total failure. In this behavior, a linear translation is performed for a given distance, which can be configured [7]. In this case, the robot backs up a certain distance and thus draws the attention of the operators for help.
- **Wait:** The robot waits in a stationary state for a certain time, which can be configured. This is useful in case of time-based obstacles. For example, it is useful when the robot is in crowded spaces where there are many people, which does not allow it to move forward successfully. In this case, the robot waits for a while until the space is a bit clearer.

IV. Background about Environment Representation

Environmental representation is "how the robot perceives its environment" [7]. In this way, the representation of the environment is vital for the correct operation of the navigation servers and the satisfactory execution of the tasks that the robot must perform, since it becomes one of the main sources of data. This environmental representation is possible thanks to one of the best known tools within autonomous navigation, called the Costmap. A costmap can be defined as "the representation of sensor data on a map" [7], in this case, the map of the environment. The costmap is a regular 2D grid of the environment, in which information from a series of sensor processing plugins, also called costmap layers, is stored. The cost of each grid cell can be unknown, free, occupied or inflated, depending on the current conditions of the environment where the robot is. Costmap layers are very useful when detecting obstacles in the environment, as it helps to prevent the robot from colliding with them [7].

The costmap layers currently available are: the static layer, which is responsible for storing the occupancy information in the costmap from the obtainment of a static map of the environment, which represents a largely unalterable portion of the map; the inflation layer, which adds new values around the obstacles so that the robot does not collide with them and can follow the path safely; the obstacle layer, which represents the obstacles read by the sensors in a two-dimensional costmap; and the Voxel layer, which does the same as the previous one but in three dimensions [7]. For the construction of a costmap, the use of a LiDAR sensor is important. Currently, there are two types of Costmaps:

A. Global Costmap

The global costmap is generated upon the static map obtained from the environment map, and helps to avoid known obstacles of the map. In addition, it is important to note that the global costmap remains static on the environment map, that is, it does not move as the robot moves. This type of costmap is used by the planner to generate a global long-range plan [7].

B. Local Costmap

The local costmap "is created from sensor data over a small region specified in the parameters." Unlike the global costmap, the local costmap helps avoid dynamic obstacles that are not known within the map. In addition, it moves with the robot on the global costmap. This type of costmap is used by the controller to calculate local control efforts and generate a short-term plan, such as avoiding a moving obstacle or completing an immediate task [7].

By correctly configuring each of the parameters that make up the navigation servers and using the global and local costmaps built from a static map of the environment, the robot can navigate autonomously, avoiding both static and dynamic obstacles that are in the trajectory that the robot must follow to reach its objective.

V. Results

To tackle the way in which the robot moves without leaving the sidewalk, it is important to make (using the laser sensor) and use the static global map of the environment, where the sidewalk is the main element to be mapped. In addition to the sidewalk, the static obstacles must also be part of the global map. For this, the SLAM method is implemented, which means Simultaneous Localization and Mapping. Using this tool, a space can be fully or partially mapped in 2D, enhancing and updating the map as the robot interacts with its environment over time, as well as locating it in space. Since the sidewalk is not detected by the laser sensor as an obstacle, it is important to modify the map obtained in an editing program, in such a way that the sidewalk can be drawn on the map, thus guaranteeing that the robot cannot leave the sidewalk thanks to the implementation of the global costmap.

To validate the behavior of a mobile robot in different configurations, where what changes is the planner and the controller to be used, an environment in the Gazebo simulation program and the simulated model of a differential robot were created, with two fixed standard wheels and a castor wheel. The robot has IMU, LiDAR, camera and GPS sensors. The simulated version of the differential mobile robot can be seen in the figure 1.

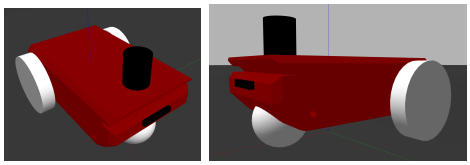

Fig. 1. Simulated version of a differential robot.

Likewise, the simulated environment in which the robot is going to move can be seen in the figure 2.

Fig. 2. Simulated environment with the sidewalk where the robot moves.

The global map of the environment, made using the SLAM technique and modified in the editing program, can be seen in the figure 3.

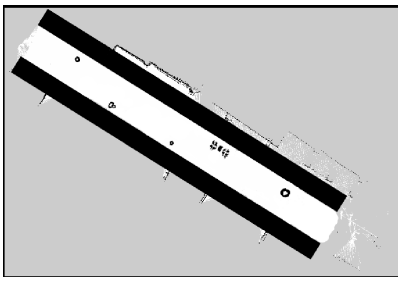

Fig. 3. Global environment map.

The parts of the map in black are the areas where the robot cannot pass, that is, the areas that are occupied by other objects. On the other hand, the parts of the map in white are the clear areas where the robot can move, which in this case would be the sidewalk.

It is important to mention the methodology used for the development of this work, that is composed of three steps: analyze the algorithms used for autonomous navigation in urban environments composed of sidewalks, implement those algorithms in the simulated version of the mobile robot and evaluate the performance of the system developed in said version. Taking into account the above, the tests are carried out with the planners and controllers mentioned above that can be used for this type of robot. In this case, as it is a differential robot, the NavFn and Smac 2D planners are implemented, as well as the DWB, TEB and RPP controllers. The initial pose of the robot is set both in Gazebo and in the Rviz visualization program, as can be seen in the figures 4 and 5, respectively.

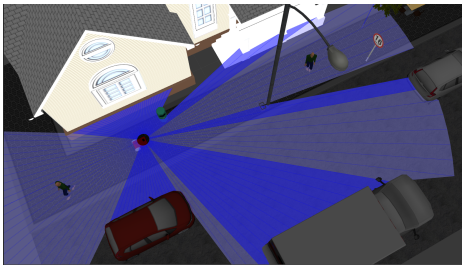

Fig. 4. Initial pose in Gazebo.

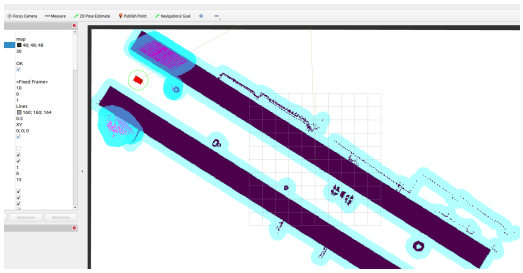

Fig. 5. Initial pose in Rviz.

Once the initial pose of the robot is determined, the time it takes to follow the trajectory from the initial pose to the final pose is taken for each of the mentioned planners and controllers, as part of the evaluation of the developed system. For this, a maximum speed of 0.25 m/s was configured. The results are shown in the table I.

TABLE I
ROBOT NAVIGATION TIME TO REACH THE TARGET, USING DIFFERENT PLANNERS AND CONTROLLERS

Controller/Planner	NavFn (Dijkstra)	NavFn (A*)	Smac 2D
DWB	77s	69s	73s
TEB	59s	62s	66s
RPP	71s	63s	79s

The global trajectory calculated by the NavFn planner can be seen in the figure 6.

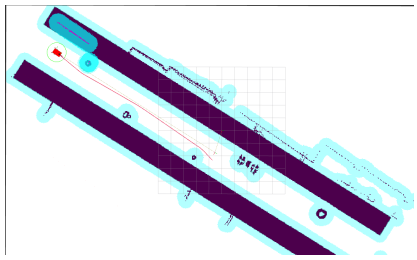

Fig. 6. Trajectory calculated by the NavFn planner.

On the other hand, the global trajectory calculated by the Smac 2D planner can be seen in the figure 7.

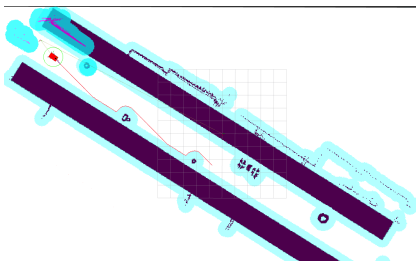

Fig. 7. Trajectory calculated by the Smac 2D planner.

After navigation, the robot reaches the target pose given by the user, as can be seen in the figures 8 and 9.

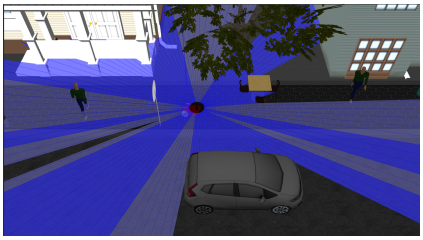

Fig. 8. Final pose in Gazebo.

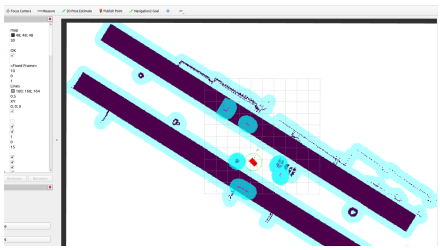

Fig. 9. Final pose in Rviz.

The previous experiment was carried out taking into account only static obstacles, that is, those that could be seen on the global map. Now, the same experiment was performed, but this time with a greater saturation of obstacles, increasing the number of both static and dynamic obstacles. The results are shown in table II.

TABLE II
ROBOT NAVIGATION TIME TO REACH THE TARGET, USING DIFFERENT PLANNERS AND CONTROLLERS, WITH MORE SATURATION OF OBSTACLES

Controller/Planner	NavFn (Dijkstra)	NavFn (A*)	Smac 2D
DWB	89s	80s	84s
TEB	65s	68s	71s
RPP	75s	70s	91s

According to the results shown in tables I and II, the configuration that obtained the shortest navigation time, managing to avoid both static and dynamic obstacles, was the TEB controller together with the NavFn planner, using Dijkstra's algorithm. The results obtained may be due to different factors. One of them is the shape of the trajectory calculated by each planner. For example, the trajectory generated by the Smac 2D planner passed very close to the detected obstacles, which sometimes caused the trajectory to not update as quickly when it encountered obstacles that were not on the global map, which caused soft collisions with them, but without affecting tracking towards the target. On the contrary, the trajectory planned by NavFn did not pass as close to the obstacles, which allowed the robot to have a little more margin when avoiding them. Other elements that can influence the result can be computational factors (performance), the geometry of the robot, the type of obstacle detected, the speed of movement of the robot or the speed with which the trajectory was updated. However, regarding speed, the one that best contributed to the achievement of the objective was chosen, allowing the robot to correctly avoid obstacles.

The planners and controllers used present very good efficiency at the time of robot navigation, fulfilling the main objective which is, in this case, to move from a start point to an end point without colliding with obstacles or leaving the sidewalk.

VI. CONCLUSIONS

This article presents a systematic review of the literature and comparative evaluation of different trajectory planning and tracking methods, validated in a simulated version of a differential robot. When starting navigation, it is very important to take into account the maximum speed at which the robot can move to accurately follow the trajectory estimated by the planner. Sometimes, when the speed was very high, the orientation changes that can be observed, for example, in the trajectory calculated by the Smac 2D planner, could not be performed accurately by the robot, which caused the collision with some obstacles. Another important aspect that could be observed during the experiment is that the trajectories calculated by the planners are very different, which can influence the time it takes for the robot to move from the initial point to the final point. As can be seen in the images, the trajectory calculated by the NavFn planner is more direct and linear than the one calculated by the Smac 2D planner. Finally, it is important to highlight the importance of correctly configuring the parameters for the different navigation servers, especially the controller, since it is responsible for the robot being able to faithfully follow the globally calculated trajectory.

REFERENCES

[1] Ollero, A. (2001). Robótica. Manipuladores y robots móviles. In Marcombo S.A, p. XVII.

[2] Du, Y., Hetherington, N. J., Oon, C. L., Chan, W. P., Quintero, C. P., Croft, E., Van der Loos, H. M. (2018). Sidewalk Delivery Robot Navigation: A Pedestrian-Based Approach. In Human-Aiding Robotics: Open Issues and Future Direction 2018. IEEE, Institute of Electrical and Electronics Engineers.

[3] Kümmerle, R., Ruhnke, M., Steder, B., Stachniss, C., Burgard, W. (2015). Autonomous robot navigation in highly populated pedestrian zones. Journal of Field Robotics, 32(4), 565-589.

[4] Lidoris, G., Rohrmuller, F., Wollherr, D., Buss, M. (2009, May). The autonomous city explorer (ACE) project—mobile robot navigation in highly populated urban environments. In 2009 IEEE International Conference on Robotics and Automation (pp. 1416-1422). IEEE.

[5] Correl, P. N. (2016, April). Introduction to autonomous robots. Kinematics, Perception, Localization and Planning. CreateSpace Independent Publishing Platform.

[6] Macenski, S. (s.f.). smac_planner. (2022). ROS Index. [Internet]. Available: https://index.ros.org/p/smac_planner/#foxy

[7] Macenski, S., Martín, F., White, R., Clavero, J. (2020). The Marathon 2: A Navigation System. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

[8] Macenski, S. navigation2/nav2_dwb_controller at main · ros-planning/navigation2. GitHub. [Internet]. Available: https://github.com/ros-planning/navigation2/tree/main/nav2_dwb_controller

[9] Macenski, S. navigation2/nav2_regulated_pure_pursuit_controller at main · ros-planning/navigation2. GitHub. [Internet]. Available: https://github.com/ros-planning/navigation2/tree/main/nav2_regulated_pure_pursuit_controller