XLI CILAMCE
XLI Ibero-Latin American Congress on Computational Methods in Engineering
16-19 November, 2020 FOZ DO IGUAÇU PARANÁ - BRAZIL

Evaluation on IMU and odometry sensor fusion simulation results for a 4-Wheeled-Drive robot using AMCL on ROS framework.

Samir E. Silva[1], Ronaldo A. Oliveira[1], Vinícius R. Motta[1], Carlos T. Valadao[1], Daniel F.T. Gamarra[2], Marco A. Cuadros[1]

[1]*Industrial Automation Group, Federal Institute of Espirito Santo*
ES-010 - Km 6,5, Zip Code 29173-087, Serra/ES, Brazil
ehlertsamir@gmail.com, {ronaldo.oliveira, vinicius.motta, carlostvaladao, marcoantonio}@ifes.edu.br
[2]*Electrical Energy Processing Dept., Federal University of Santa Maria*
Roraima Ave., 1000, Zip Code 97105-900, Santa Maria/RS, Brazil
daniel.gamarra@ufsm.br

Abstract. Mobile robot localization algorithms establish correspondence between the robot's coordinate system and the environment map. However, usually, it is not possible to get an accurate direct measurement of the robot pose. Therefore, it must be inferred from local sensors, which are, in general, susceptible to noise. Some robot models, such as the 4-Wheeled-Drive, can rapidly increase odometry error, especially due to drift during angular movements. Hence, correction techniques are recommended to compensate these errors. This paper evaluates the impact of the Extended Kalman Filter (EKF) to fuse data from an Inertial Measurement Unit (IMU) sensor and odometry and how it impacts in the Adaptive Monte Carlo Localization (AMCL) algorithm. The use of data fusion brings better results than using data from only encoders. The experiments were simulated using ROS framework and Gazebo as simulation environment. A series of goals were given to test the localization algorithms performance using the move base control algorithm in a map with obstacles. This experimental setup allowed to verify how each localization method behaves comparing (a) only odometry; (b) odometry and the AMCL; (c) the fusion between odometry and IMU, and (d) the fusion of odometry with AMCL and IMU.

Keywords: mobile robot localization, data fusion, AMCL, ROS, EKF.

1 Introduction

Navigation is one of the main issues for mobile robotics, and includes important aspects, such as obstacle avoidance, odometry errors, accumulative errors in sensors, absolute and relative position, error correction, among others. To allow the robot to move in the correct trajectory, it needs to know its posture as accurately as possible to serve as input for the controller. This last one sets each motor speed to make the robot achieve the desired position. However, sensing the correct robot position may prove to be a difficult task, especially when using only embedded sensors. Odometry, which is one of the most used ways to measure a robot posture tends to add error throughout time. Therefore, after some cycles, the measured robot position is distant from the real one.

Absolute measurement techniques, on the other hand, can be used to specify a better localization of the robot, using fixed references. However, the devices used for such measurement usually need to be placed outside the robot, and sometimes they are unable to communicate with the robot. Examples of absolute localization are GPS and visual landmarks. Therefore, using multiple ways of sensing the robot position and combining that information to reach a more accurate posture can be a better approach for calculating the real one. Furthermore, fusing data among different sensors can also reduce cumulative errors from odometry.

In this paper, it is presented through simulation, using the Robot Operating System (ROS), four different techniques of measurement applied as input to the same control system of a 4-Wheel-Drive (4-WD) robot and their respective performances in making the robot achieving its goal. The localization methods applied were (a) using only the robot's odometry, (b) combining odometry with AMCL, which stands for Adaptive Monte Carlo

Localization and uses both a map and a laser sensor, (c) making a fusion between the odometry and an Inertial Measurement Unit (IMU) data using the Extended Kalman Filter (EKF) and (d) making a fusion between the odometry with AMCL and an IMU.

There are some applications of Monte Carlo Localization (MCL) to improve Robot localization, such as in the works of Thrun and Talwar in [1], [2] that uses laser sensors, or the work of Alves in [3] that used the AMCL algorithm with a Kinect sensor for robot localization. Also, the EKF has been used to fuse measurements of gyros, accelerometers and magnetometers for attitude estimation in drones in [4], Alatise used an EKF to fuse inertial sensors and information derived from a camera in order to calculate the position and orientation of a mobile robot in [5].

The robot model and the techniques for localization are presented in the Theoretical Background section followed by the Methodology, experimental Setup, Results and Conclusion topics. All the simulations added noises to the sensors, which are an important feature in real-world applications, making the simulation closer to reality.

2 Theoretical Background

2.1 Adaptive Monte Carlo localization

The AMCL, which stands for Adaptive Monte Carlo localization, is a probabilistic localization method that uses the map model and laser sensor to improve the odometry information about the robot. This algorithm uses particle filters to track the robot posture in relation to a known map [6].

The Monte Carlo localization generates random particles in different postures, which are the probabilities of the robot posture [7], [8]. Using the laser sensor and the map stored in memory, the algorithm can compute the particles which represent more accurately the robot actual posture. The particles that represent better the robot position are given higher weights. Throughout the time, while the robot moves, other particles are generated, and their weights are adjusted to best fit the map and the laser data. To optimize the algorithm and avoid the need for more computer processing, the Adaptive Monte Carlo localization excludes part of those particles, which are more irrelevant to the localization algorithm. Thus, the robot can still calculate its pose, but with less computational effort.

2.2 Extended Kalman Filter

Kalman Filter has an extensive application in different fields of Engineering and Science. The filter can improve the accuracy of measurement by doing a weighted average and calculating the weight of distinct sources of sensing and the mathematical analytic model of the system, according to their accuracy. In other words, it uses measured values and model information to infer the output [9], thus allowing the system to measure data throughout the time, which may have noise interference, and get better results.

The Kalman filter is recursive, propagating the effect of previous measures to know the current output state [10]. Although designed for linear models only, there is an extended version that applies to nonlinear models. More details about Kalman Filter and its extended version are in [10]. The robot has a nonlinear model, and the Extended Kalman Filter makes the fusion between the odometry and the IMU data. Therefore, it generates a more accurate new odometry signal, compensating the errors from the previous methods. ROS already has the Extended Kalman Filter package, which simplifies the whole process of implementing the filter.

2.3 Robot Kinematic Model

The robot encoders' information is integrated throughout time, thus giving its position by odometry. Equation (1) shows the mathematical model for a unicycle-like robot with nonholonomic constraints [11]:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -a \cdot \sin(\theta) \\ \sin(\theta) & a \cdot \cos(\theta) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}, \tag{1}$$

where v and ω are the linear and angular speed of the robot, θ is the robot orientation and $\dot{x}$, $\dot{y}$, $\dot{\theta}$ are the robot states, corresponding, respectively, to the variation of position in x, y and the orientation θ. The variable a refers to the distance between the robot's traction axis center and a point of reference on the robot.

Although practical, this method leads to errors throughout the time since the integration of the velocities can carry cumulative errors. The integration of the robot's states given in Equation (1) allows us to find the current pose of the robot. However, these equations do not consider some physical phenomena, such as robot slippage that encoders cannot measure and drifts, among others. These errors tend to happen and accumulate, especially when the robot makes an angular movement, and its actual position differs from the measured one since the robot was unable to measure those events.

3 Experimental Setup

3.1 Robotic Operating System

To run the simulations, the Robotic Operating System (ROS) was used, which is an open-source collection of frameworks that has thousands of available packages. This variety of packages, which comprise several controllers, algorithms, sensors, robots and more, allows effortless programming and simulation for robotics. The philosophy of the ROS creators, Eric Berger, and Keenan Wyrobek was that a software developer should not necessarily need the low-level programming hardware knowledge to implement code in a robot. Instead, these low-level programming should be abstracted, and a high-level framework used. Scott Hassan, from Willow Garage, shared the same idea from Berger and Wyrobeck, and the three joined to create a “Linux for Robotics.” The code from the ROS committee was posted in SourceForge on November 7th, 2008 [12].

Other programs, such as Gazebo, were also used. Gazebo is a robot simulator, compatible with ROS, made by the Open Robotics Foundation and can simulate different kinds of robots, including complex ones with artificial intelligence [13]. ROS provides several tools, such as RVIz, which shows the robot topics’ behavior graphically, making it easier to simulate and check the robot behavior visually.

3.2 4-Wheeled-Drive robot

The 4-Wheeled-Drive robot used in this work relies on four motors that move at the same speed on each side. In other words, this robot works as if there would be only a single motor for each side of the robot. The robot movement is based on the skid steering to give the movement direction, and, since both motors of each side share the same speed, its kinematic model can be approximated to a unicycle model with nonholonomic constraints [14], which means the robot cannot move sideways. Subsection 2.3 shows more details about the robot kinematic model.

4 Methodology

Four different localization techniques were compared using ROS and a simulated 4-WD mobile robot. As aforementioned, the first simulation relies only on odometry, while the second uses the AMCL algorithm, a map, and a laser sensor. The third simulation makes a data fusion between the odometry without AMCL and an IMU through the Extended Kalman Filter and, finally, the last method relies on a fusion between odometry with AMCL and IMU. All these localization methods generate input to the move base controller that, eventually, controls the robot, guiding it to its goal.

4.1 Move base controller

The move base package allows the robot movement to use a navigation stack. This package has the move base node, which contains both global and local planners and cost maps, besides a recovery behavior node. As inputs it can receive data from the AMCL, sensors, odometry and map, and, as outputs, it generates the velocity commands to the robot. More details can be viewed in [15].

4.2 General Architecture

The simulations employ ROS, Gazebo and Python using the four methods. A xacro model, with the odometry, laser and IMU sensors, was created to allow the robot to use the sensors in the algorithms. The fusion approach allowed the robot to have a more precise position of its actual posture, compensating the errors generated by the odometry-only data and using more sensors to infer the robot posture.

All simulations used the same map, and the robot received a set of six points it should go through. The idea of these simulations is to graph the error curve and the output of the localization algorithms using the four techniques and, thus, compare them. The overall control architecture can be viewed in Section 2.3, followed by experiments using the different approaches of localization.

The simulations used Gazebo to create and run the robot inside a 3D map. Additionally, RVIz is used to visualize the robot data and its behavior. The move_base is used to control the robot using data from the four proposed localization methods. Python code was written to generate the setpoints and gather data to generate the graphs.

5 Results

There were made four experiments, using distinct combinations of localization methods. The first and simpler one uses only the odometry; the second uses odometry and AMCL; the third uses data fusion between IMU and odometry; and the fourth uses data fusion of IMU, odometry and AMCL. Figure 1 shows how each localization method behaved in the map. The robot started from position (0,0,0) in all simulation and had to pass by all marked points, from the beginning point, passing through the point #1 to #6, before completing the cycle returning to position (0,0,0).

As shown in Figure 1(a) and Figure 1(c), the use of a localization method that is not reliable can lead to collisions, which can be viewed in these pictures, where the robot could not complete the whole trajectory. The following subsections will detail the localization method used and shows the error between the robot measurement and the real position, known as ground truth, that was acquired from Gazebo.

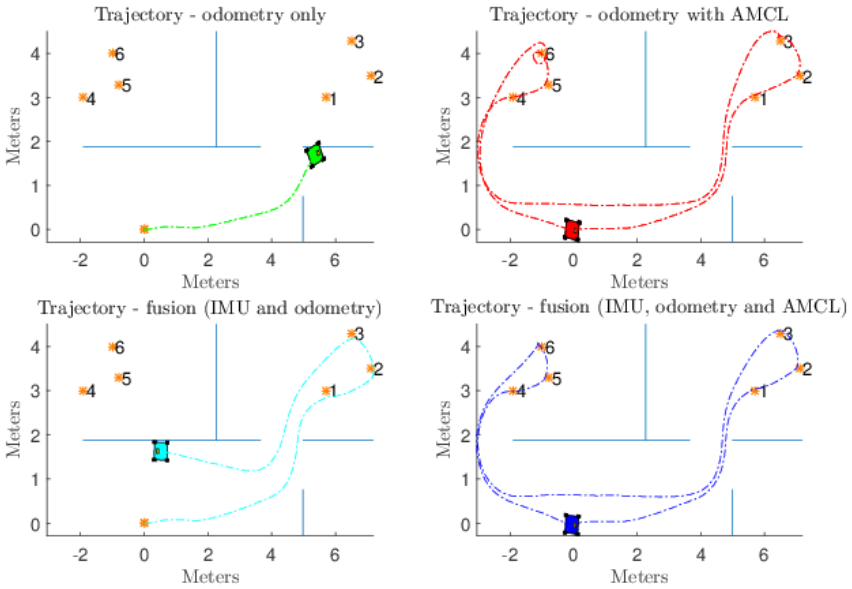

Figure 1 - The four localization approaches compared. Graph (a) shows the odometry only, (b) odometry with AMCL, (c) fusion between IMU and odometry and (d) fusion between IMU and odometry with AMCL.

5.1 Localization using only odometry

The odometry localization approach uses only the encoders, and the robot pose variation throughout the time to compute the current robot pose. Although very practical, odometry is sensitive to errors, since drifts and slippage sometimes are not detected, and the robot moves without encoders realizing the movement. Therefore, throughout

time, these errors are accumulated, making that the actual position of the robot will differ considerably from the position known by the controller. Figure 2 shows the graphs using only odometry as localization method.

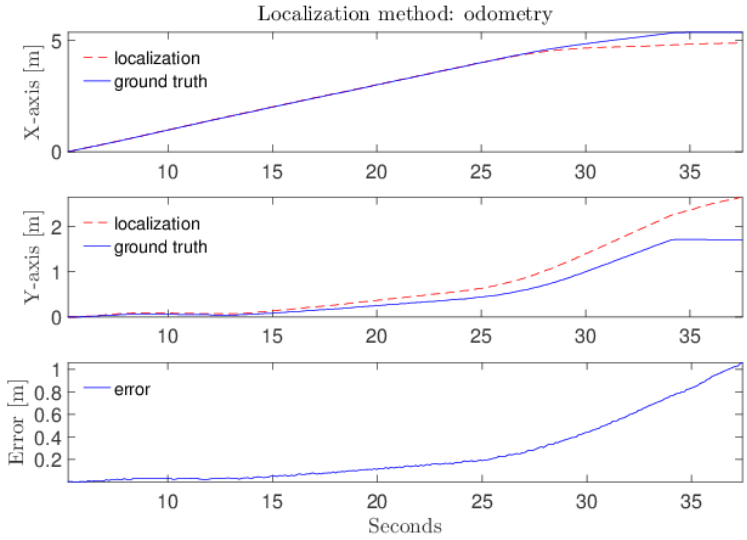

Figure 2 - Cartesian points (x; y) and the error of the localization and the real position of the robot for odometry.

5.2 Localization using odometry with AMCL

Adding the AMCL algorithm to the odometry signal allows the robot to recalculate its position and compensate for the errors due to the mistaken measurements of the encoder. The navigation with the AMCL is more precise, and the robot could reach the destination without much error compared with the first approach. Figure 3 shows the path and graph for odometry with AMCL localization.

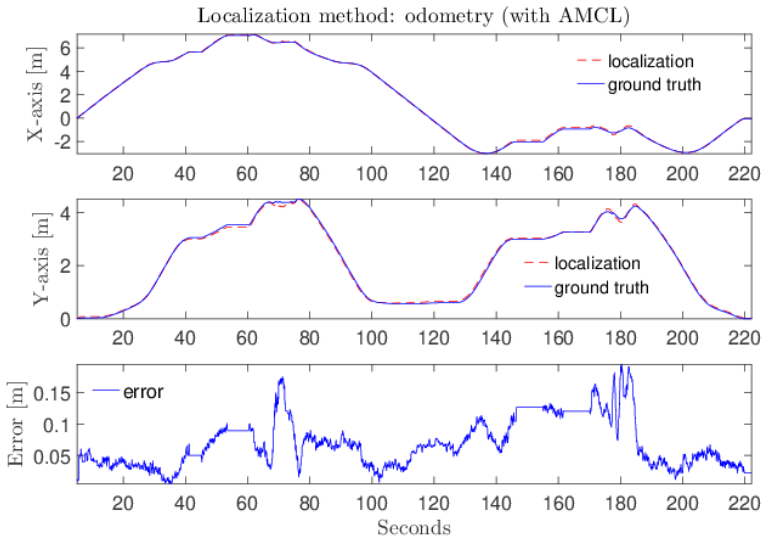

Figure 3 - Cartesian points (x; y) for odometry with AMCL localization method and error between the measured and real robot position.

5.3 Localization using data fusion of the odometry and IMU

Another simulation was done by making the data fusion between the odometry and the IMU. This signal generated a new odometry signal (odom_combined). This topic has the data fusion gives a new odometry signal with information from both sources. Figure 4 shows the graph of the data fusion localization vs ground truth.

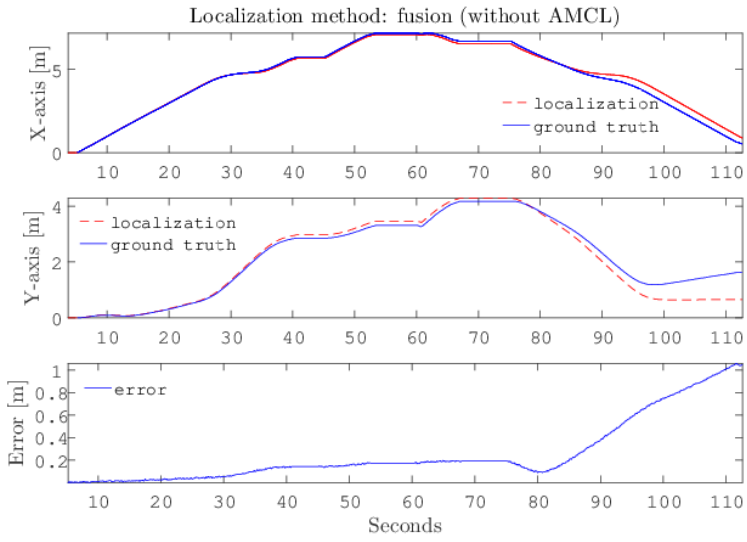

Figure 4 - Cartesian points (x; y) and the error of the localization and the real position of the robot for fusion between odometry and IMU localization method.

5.4 Localization with data fusion of the odometry with AMCL

The last simulation used data fusion between IMU and odometry with AMCL. As in the previous case, it was generated a new odometry signal called odom_combined. The fusion of the data gave a more precise odometry signal. Figure 4 shows graph for the complete data fusion localization vs ground truth.

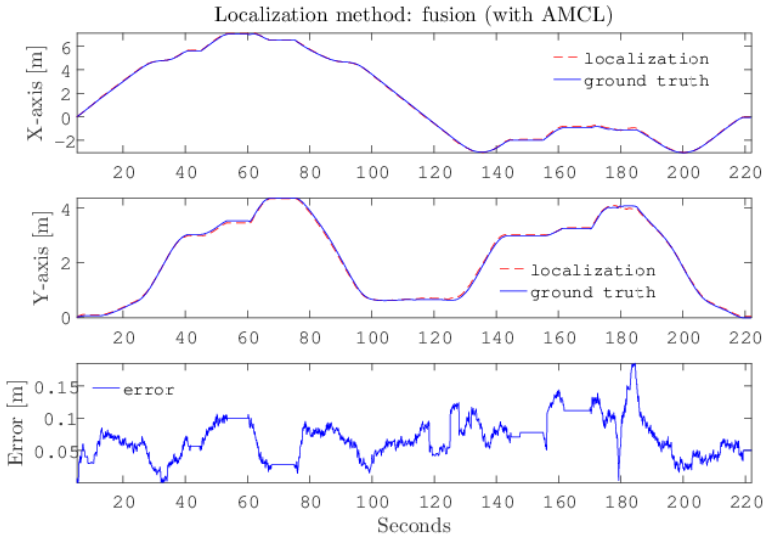

Figure 5 - Cartesian points (x; y) and the error of the localization and the real position of the robot for fusion

5.5 Error calculation

The error between the measured robot position and ground truth was computed using the Euclidean Distance. Table 1 shows the mean square error. Additionally, the average error and standard deviation for the error vector was computed. As Table 1 shows, the methods with higher errors were those that collided (odometry only and fusion of IMU and odometry without AMCL). Those that applied the AMCL to correct the odometry presented lower errors and completed the trajectory successfully.

Table 1. Comparison table with the errors of each localization method using the Euclidean Distance.

Method	Error (MSE)	Error (mean)	Error (std)	Final
Odometry only	0.1527	0.2612	0.2907	Collided
Odometry with AMCL	0.0062	0.0690	0.0385	Completed
Fusion (IMU + odometry)	0.1471	0.2582	0.2836	Collided
Fusion (IMU + odometry with AMCL)	0.0057	0.0687	0.0312	Completed

6 Conclusions

The simulations showed that using only odometry as localization method made the robot accumulate more error, since there were no corrections. The odometry with AMCL, on the other hand, compensated these errors, giving a more accurate localization. The fusion method without AMCL led to an improvement over the odometry only approach, but eventually made the robot collide with an obstacle. Finally, the fusion method with AMCL showed the best performance (less error and no collisions). As a conclusion, the fusion method (IMU and odometry with AMCL) gave the best result with the lowest error, although more computational costly. Another important conclusion is that the AMCL is necessary to make the robot find its actual position in the map. Even in the simulation using fusion of the IMU and odometry, the results did not perform well, which shows that AMCL in this configuration setup plays an important role to correct the robot position and minimize the error between the calculated and the real position.

Acknowledgements. Authors would like to thank the Federal Institute of Espírito Santo, Campus Serra (IFES-Serra) and the Industrial Automation Group (GAIn) for the technical support and Fundação de Amparo a Pesquisa e Inovação do Espírito Santo (FAPES) for the financial support.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artif. Intell.*, vol. 128, no. 1, pp. 99–141, 2001, doi: https://doi.org/10.1016/S0004-3702(01)00069-8.

[2] D. Talwar and S. Jung, "Particle Filter-based Localization of a Mobile Robot by Using a Single Lidar Sensor under SLAM in ROS Environment," in *2019 19th International Conference on Control, Automation and Systems (ICCAS)*, 2019, doi: 10.23919/ICCAS47443.2019.8971555.

[3] R. C. Alves, J. de Morais, and K. Yamanaka, "Cost-effective Indoor Localization for Autonomous Robots using Kinect and WiFi Sensors," *Intel. Artif.*, vol. 23, no. 65, pp. 33–55, 2020, doi: 10.4114/intartif.vol23iss65pp33-55.

[4] X. Ren and S. Liu, "Combining Extended Kalman Filter with Complementary Filter for UAV Attitude Estimation based on MEMS MARG Sensors," in *Proceedings of the 2016 International Forum on Management, Education and Information Technology Application*, pp. 746–752, doi: https://doi.org/10.2991/ifmeita-16.2016.136.

[5] M. Alatise and G. Hancke, "Pose Estimation of a Mobile Robot Based on Fusion of IMU Data and Vision Data Using an Extended Kalman Filter," *Sensors*, vol. 17, no. 10, p. 2164, 2017, doi: 10.3390/s17102164.

[6] M. Â. F. Torres, "Exploiting particle-filter based fusion in mobile robot localization," University of Coimbra, 2018.

[7] S. Thrun, D. Fox, and W. Burgard., *Probabilistic Robotics*. In: The MIT Press: The MIT Press, 2005.

[8] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *AAAI/IAAI*, vol. 1999, no. 343–349, p. 2, 1999.

[9] M. S. Grewal and A. P. Andrews, *Kalman Filtering: Theory and Practice Using MATLAB®: Third Edition*. Wiley, 2008.

[10] L. Marin, "Navegación De Un Robot Móvil De Configuración Diferencial Basada En Fusión Sensorial," 2011.

[11] N. A. Majid, Z. Mohamed, and M. A. Mohd Basri, "Velocity control of a unicycle type of mobile robot using optimal PID controller," *J. Teknol.*, vol. 78, no. 7–4, Jul. 2016, doi: 10.11113/jt.v78.9415.

[12] L. Joseph, *ROS Robotics Projects*. Birmingham, UK, 2017.

[13] Open Source Robotics Foundation, "Gazebo," 2014. http://www.gazebosim.org/.

[14] G. K. Fourlas, G. C. Karras, and K. J. Kyriakopoulos, "Fault tolerant control for a 4-wheel skid steering mobile robot," *CEUR Workshop Proc.*, vol. 1507, pp. 177–184, 2015.

[15] Open Source Robotics Foundation, "move_base," 2018. http://wiki.ros.org/move_base (accessed Jul. 20, 2020).