Control and Path Planning for an Unmanned Ground Vehicle in Simulation

Wesley Lowman[1], Chad G. Rose[2], and Vahid Azimi[3]

[1] Undergraduate Student, Department of Computer Science, Auburn University
[2] Assistant Professor, Department of Mechanical Engineering, Auburn University
[3]Staff Research Engineer, Controls, Gatik

With the prevalence of robotics in daily life expected to only increase, there exists many opportunities for the optimization of robotic system performance. One such area is the control and path planning of unmanned ground vehicles (UGVs). In robotics, path planning is used to generate a trajectory for a robot to move from one point to another in a 3D-reference coordinate system [2]. Essentially, this means creating a program to help the robot determine the most efficient way to travel from one point to another while accounting for all possible variables and obstacles. Control, on the other hand, ensures that the robot finds the most efficient way to remain on the planned path to get there; it acts as a "real-time" calibration method [7]. In addition to optimizing the performance of a UGV, developing and testing control and path planning solutions for it mitigates potential risks associated with both safety and cost.

This study designs a differential drive UGV in SolidWorks™ and configure it into a Unified Robot Description Format (URDF) file, configure a ROS2 package to host all of the project contents, design a simulation environment in Gazebo™, use OpenCV™ to design an overhead camera system with which the UGV will communicate, implement localization, mapping, path planning, and motion-planning programs in Python™, implement control in the ros2_control framework, and evaluate the efficacy of UGV path planning algorithms [3,5]. While the use of tools such as SolidWorks™, MeshLab™, and Blender™ provided a unique interdisciplinary experience in the completion of this project, the primary consideration is the control and path planning of the UGV itself. This study considers a UGV that is required to traverse from a known initial position to a desired target location while also navigating any potential obstacles it faces. For test settings, this study saw the creation of obstacle course environments for the robot to traverse. The first major step in enabling a path planning program for the UGV was to place a camera at the top of the Gazebo™ simulation environment (as shown in Fig. 1) and connect it to OpenCV™, providing the UGV with a utility that functions as a satellite camera.

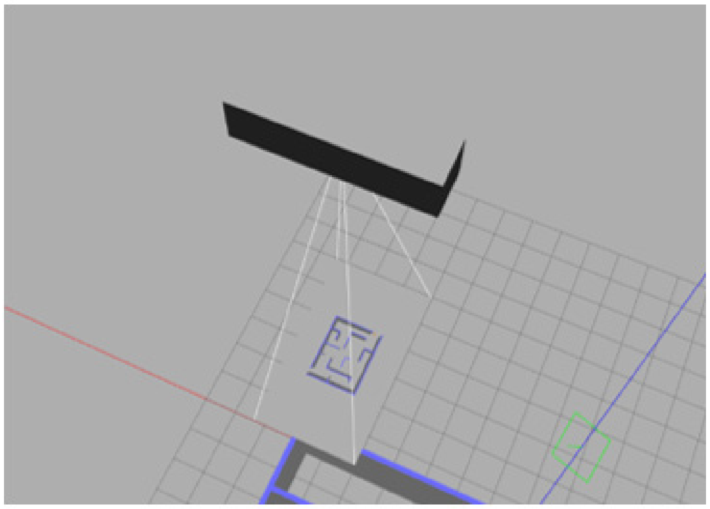


Fig. 1. Example of output from satellite camera inside the Gazebo™ world once OpenCV™ is connected.

After a given simulation environment was prepared, the next major component of the project was using Python™ and ROS2 libraries to develop programs to navigate the UGV. First, a localization program was developed that extracts the mask of all objects in the region of interest in the environment (as shown in Fig. 2) and then localizes the UGV through background subtraction and foreground extraction using an overlay image scan of the environment captured by the satellite camera [4].

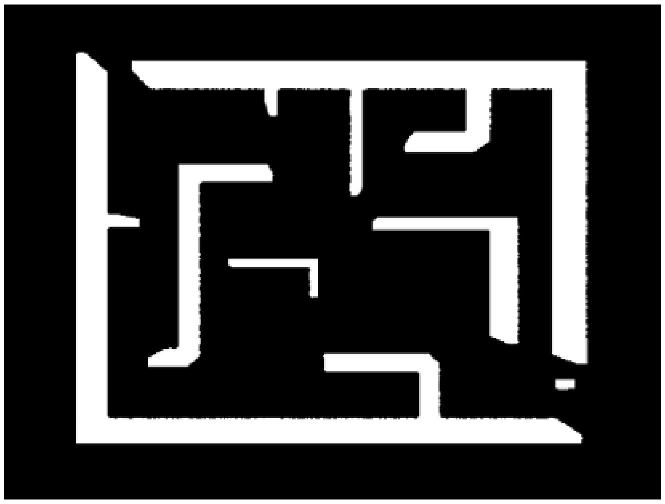


Fig. 2. Example of a region of interest mask output generated by this project's UGV localization program.

After this, this study implemented a mapping program for the UGV that would graphify the map by representing juncture points as nodes and connecting routes to such points as edges (as shown in Fig. 3). The program makes this possible by simplifying the environment itself into a grid and running a one-pass algorithm on itself [6]. The one-pass algorithm scans the grid from left-to-right and top-to-bottom and connects nodes that are neighbors, including all nodes that are connected diagonally.

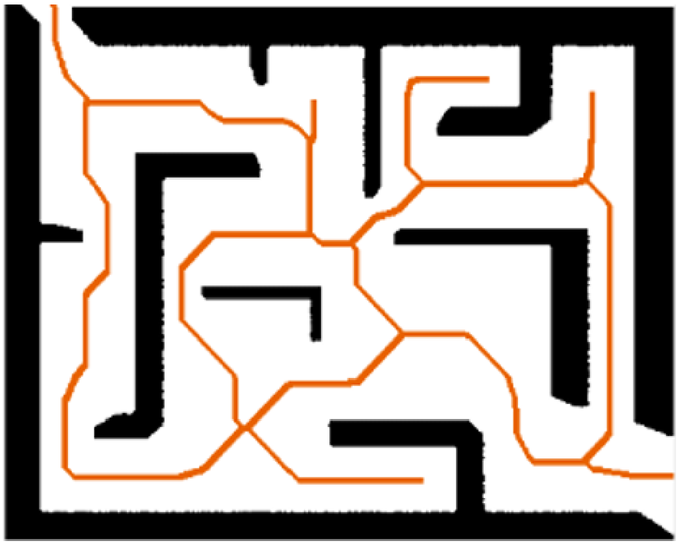


Fig. 3. Example of the map output generated after nodes in the graph of the simulation environment are connected.

Next, this study implemented 9 algorithms, including Dijkstra's algorithm, the A* algorithm, and A* variant algorithms into the UGV's path planning program. All these algorithms have the purpose of finding the shortest path in a graph, whereas the key difference between the Dijkstra's algorithm and the A* algorithm (and its variants) is that A* uses a heuristic function that gives priority to nodes that are more optimal. Last in the programming of the UGV was its motion planning program, which is a ROS2 program that commands the UGV to follow coordinate points on the shortest path (as shown in Fig. 4) generated by the path planning program.

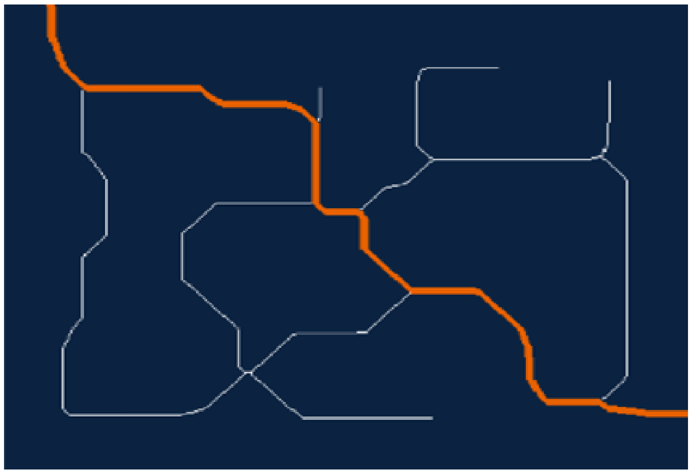
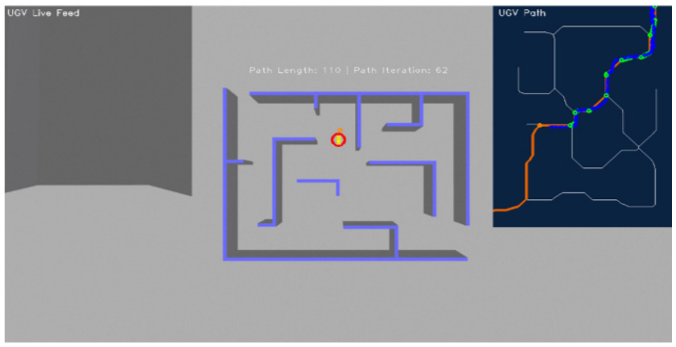


Fig. 4. Example of the shortest path output generated by the UGV's path planning program.

For control, the UGV is using a ROS2 framework known as ros2_control, which has allowed for a custom control system. This was done by creating a custom control file and coding in command interface objects for different joints of the UGV and connecting this file to the UGV's URDF file. All the previously mentioned programs are connected to a central navigator.py program, which is run in the Linux terminal to combine the many project components together; this program also provides a live feed, as shown in Fig. 5.



Path Length: 110 | Path Iteration: 62

Fig. 5. Example of the live feed output displayed by the UGV's navigator.py program.

This project has already yielded a viable system for the control and path planning of a UGV in a simulated environment. An accurate map, along with a program that generates the most optimal path was developed for a ROS2 UGV. After testing, it is confirmed that the path the UGV follows is both feasible and efficient.

The performance of the UGV in 5 maps of increasing complexity (with each map tested 20 times), based on average values for path length (PL), computation time (CT), nodes visited (NV), smoothness (S), path safety (PS), path continuity (PC), memory usage (MU), path deviation (PD), and search space visited (SV) indicates that, of the algorithms tested, Theta* provides the most efficiency in the navigation of a UGV from a start point to a goal point, as shown below in Table 1 [1].

Table 1: Performance Summary of Path Planning Algorithms

Algorithm	**PL**	**CT (s)**	**$\log_{10}$(NV)**	**S**	**PS**	**PC**	**MU (MB)**	**PD (%)**	**SV (%)**
Dijkstra's	79.35	0.08	2.32	57.99	8.71	33.73	282.69	0.00	98.55
A*	79.35	0.04	2.18	58.11	8.71	33.73	283.38	0.00	76.48
Theta*	79.35	0.03	2.18	58.11	8.71	33.73	283.55	0.00	76.48
Beam Search	79.35	0.03	2.32	58.11	8.71	33.73	283.74	0.00	99.32
Weighted A*	79.35	0.03	2.18	58.11	8.71	33.73	283.93	0.00	76.48
Iterative Deepening A*	79.35	24.81	6.91	57.72	8.71	33.73	285.60	0.00	100.00
Bandwidth Search	79.35	0.03	2.32	57.99	8.71	33.73	286.11	0.00	98.55
Bidirectional Search	91.64	0.06	2.25	67.59	8.12	39.39	286.59	12.45	86.81
Ant Colony Optimization	89.60	0.77	4.97	70.20	8.56	36.65	288.33	10.44	100.00

Future work includes the validation of the efficacy of path planning algorithms tested in simulation by testing them with a TurtleBot 4 lite robot. In summary, this work will lead to results that will allow researchers to answer questions related to the validity of path planning algorithms in both simulation and the real world.

Statement of Research Advisor

Wesley's independent investigation into path planning algorithms, as well as the development of a simulation environment, used industry standard tools and approaches which make the results generalizable and interesting to the robotics community. The entire project has been self-motivated, independent, and high-quality.

- Dr. Chad G. Rose, Mechanical Engineering, Samuel Ginn College of Engineering

References

[1]Ceballos, Nelson David Munoz, Jaime Alejandro Valencia, and Nelson Londono Ospina. "Quantitative performance metrics for mobile robots navigation." Mobile Robots Navigation. IntechOpen, 2010.

[2]Krowinska, M., "Motion Planning vs Path Planning." Retrieved June 28, 2022, from https://www.shaderobotics.com/posts/motion-planning-vs-path-planning, (2022).

[3]Macenski, S. et al., "Robot Operating System 2: Design, architecture, and uses in the wild," Science Robotics, 7(66) p. 6074, (2022).

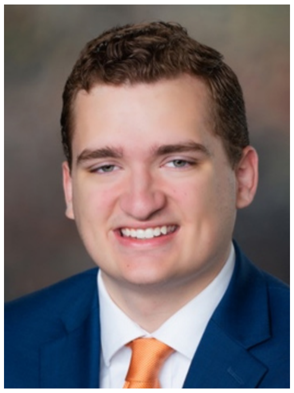
[4]OpenCV Authors, "Background Subtraction." Retrieved July, 12, 2022, from https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html, (2015).

[5]Patel, A., "Variants of A*. Retrieved November 10, 2022, from http://theory.stanford.edu/~amitp/GameProgramming/Variations.html, (1997).

[6]Pound, M., "Maze Solving - Computerphile." Retrieved July 25, 2022, from https://www.youtube.com/watch?v=rop0W4QDOUI, (2017).

[7]Sharma, M., "Introduction to Robotic Control Systems." Retrieved May 15, 2022, from https://towardsdatascience.com/introduction-to-robotic-control-systems-9ec17c8ac24f, (2020).

Authors Biography



Wesley Lowman is a junior-year student pursuing a B.S. degree in Computer Science at Auburn University. He has designed, developed this entire project. He specializes in algorithmic design, the primary focus of this project, but he also completed all mathematical and mechanical design components of the project as well.

Chad G. Rose, Ph.D. is an Assistant Professor in the Department of Mechanical Engineering. He holds a B.S. from Auburn University, with M.S. and Ph.D. from Rice University, all in Mechanical Engineering. Dr. Rose's research focus is on the design and control of robots to rehabilitate, augment, or assist human sensorimotor function.

Vahid Azimi received the M.Sc. and Ph.D. degrees in electrical engineering from the Georgia Institute of Technology in 2020. From 2020 to 2021, he was a Postdoctoral Research Fellow at Stanford University. From 2021 to 2022, he was an Assistant Professor at Auburn University. His research interests include robotics and autonomous driving. He is currently a Staff Research Engineer at Gatik, an autonomous driving company.