

Introducing ROS-Projects to Undergraduate Robotic Curriculum

Dr. Lili Ma, New York City College of Technology

Professor Lili Ma received her Ph.D. in Electrical Engineering from Utah State University focusing on autonomous ground vehicles. After that she did three-year post-doctoral training at Virginia Tech working with unmanned aerial vehicles (UAVs). Prior to joining the Dept. of Computer Engineering Technology at CUNY New York City College of Technology, she taught at Wentworth Institute of Technology for eight years. Her research interests are in designing coordinated control schemes for a group of autonomous robots. Her teaching interests are in designing robotic projects that promote undergraduate research and integrate interdisciplinary areas (robotics, artificial intelligence, IoT, electronics, and image processing).

Dr. Yu Wang, New York City College of Technology

Dr. Yu Wang received her Ph.D. degree in Electrical Engineering from the Graduate Center of the City University of New York in 2009. She is an associate professor in the Department of Computer Engineering Technology at New York City College of Technology. Her research areas of interest are engineering education, biomedical sensors, modeling real-time systems, embedded system design, and machine learning.

Dr. Chen Xu, New York City College of Technology

Dr. Chen Xu is an Associate Professor at the Computer Engineering Technology department at New York City College of Technology. She received her Ph.D. degree in Biomedical Engineering from the University of Connecticut. Her research areas of interest are biomedical sensors and instrumentation, image processing, signal processing, and non-invasive medical test.

Dr. Xiaohai Li, New York City College of Technology

Xiaohai Li received his M.S. degree in Electrical Engineering from Polytechnic Institute of New York University, New York, in 2004 and Ph.D. degree in Electrical Engineering from the Graduate Center of the City University of New York (CUNY), New York, in 2010. He worked as a Post-doc in the PRISM Research Center in the Department of Electrical Engineering at the City College of New York of CUNY in 2010. He is currently an Associate Professor in the Department of Compute Engineering Technology at NYC College of Technology of CUNY. He founded the City Tech Robotics Research Lab and is a co-founder of the City Tech Experiential Arts & Technology Lab (EAT Lab) at NYC College of Technology of CUNY. His current research interests include applied control systems, robotics, swarms, wireless sensor networks, computer vision and perceptual computing, and IoT/IoRT.

Introducing ROS-Projects to Undergraduate Robotic Curriculum

Abstract

This paper describes three MATLAB-ROS-based simulation projects developed for an undergraduate robotics course. The Robot Operating System (ROS) is an open-source framework that helps researchers and developers build and reuse code between robotics applications. Adoption of ROS in the undergraduate curricula is still rare due to its demanding requirements of C++/Python/Java programming skills and familiarity with Linux. Recently, MathWorks released its ROS Toolbox, making it easier to interact with simulators like the Gazebo and ROS-supported physical robots. The MATLAB-ROS-Gazebo simulation platform allows students to utilize other MATLAB Toolboxes, such as Image Processing, Computer Vision, Visualization, and Navigation Toolboxes, for fast algorithm development and testing.

The paper presents three projects for autonomous mobile robots on the MATLAB-ROS-Gazebo simulation platform. The first project is on sensing and perception of laser scan data and its post-processing of model-based fitting. The second project is on the path planning of an autonomous mobile robot implementing the Wavefront algorithm. The third project obtains closed-loop control of the robot's behavior based on visual hints. These three projects cover the fundamental components of controlling an autonomous mobile robot, including sensing, perception, decision-making, and low-level motion control. We believe these projects will help other educators develop ROS-based simulation projects as part of a course or a stand-alone course for teaching robotics.

Introduction

The Robot Operating System (ROS) has gained wide currency for creating working robotic systems, initially in the laboratory and then in industry. The primary programming environment for those working on ROS includes C++, Python, or Java. MathWorks recently released its Robotics Systems Toolbox and ROS Toolbox. Using MATLAB to interact with robotic simulators (such as Gazebo) and physical ROS-compatible robots (such as TurtleBot) becomes a new option. The existing toolboxes in MATLAB enable the development and verification of robotic control algorithms more quickly. Though it is still an ongoing evolution, the fundamental ROS principles of publishing and subscribing to topics, application-specific messages, invoking services, and sharing parameters remained constant. Due to the growing importance of ROS in research [1] and commercial robotics, educators began introducing ROS to enhance their robotic curricula [2–10].

Robotics is perceived in education as an excellent way to promote higher-quality learning by grounding theoretical concepts into reality. To maximize the learning throughput, the focus of any robotics software platform should be on ease of use, with little time spent integrating the components [11]. This paper describes the development of three projects on the MATLAB-ROS-Gazebo platform for a senior-level robotic course, which serves as a technical elective for the Computer Engineering Technology (CET) curriculum. Topics covered basic sensing and perception routines (laser sensing, image sequences from the robot's onboard camera, model-based data fitting, color-based feature extraction), decision-making (path planning), and low-level motion control (commanding the robot to move with a specified distance/angle or changing the robot's movement by varying its linear/angular velocities).

The context of this work is an undergraduate robotic course offered as a technical elective to senior-level students in the department of Computer Engineering Technology. This robotic course is structured to have a 2.5-hour lecture session and a 2.5-hour lab session each week. The course objectives include addressing fundamental subjects in Autonomous Mobile Robots and Robotic Manipulators and preparing students with the necessary skills in robotic programming, design, and system integration. The students will do lab/project exercises in the lab sessions to reinforce the knowledge, concepts, and algorithms learned during lectures. Project-based learning [12, 13] was adopted in the lab sessions to guide students through the implementations of fundamental algorithms discussed in the lecture sessions. We usually assign one project dedicated to autonomous mobile robots and another to robotic manipulators.

Before the COVID-19 pandemic, both the lecture and the lab sessions of this course were taught in person. In the lab sessions, students used physical robots for their experiments. These robots are built using the VEX robotic kits, which were consisted of a mobile base with a simple arm on top. Programming was using RobotC, implemented and tested on the physical robot directly without going through a-prior simulation step. During the pandemic, both sessions were taught online. We developed simulation-based projects to synchronize with the lecture sequences in the unified programming environment, i.e., MATLAB. To resemble those physical experiments, the MATLAB-ROS-Gazebo platform was brought into this course little by little each semester.

Recently, higher education has called for more in-person activities. In fall 2022, the lecture session was taught mostly online (except for midterm and final exams). The lab session returned to traditional in-person teaching. While huge desires are coming from the students demanding physical labs/projects on real robots, completely discarding the simulation projects, especially those with high fidelity, does not seem to be a wise solution. Instead, we have been looking for ways to integrate the best practices from both sides: simulation and physical.

Targeting undergraduate robotic courses, adopting the MATLAB-ROS platform can be a suitable arrangement due to a couple of factors. Firstly, programming on MATLAB is much easier than using other languages such as C++, Python, or Java. Secondly, MATLAB already has many other toolboxes dedicated to education and research, which would significantly shorten the learning curve. Students could start algorithm development and implementation likely after a two-week MATLAB tutorial. Students in the STEM field might have used MATLAB in other courses, such as math, controls, circuit analysis, and signal processing. Based on their existing familiarity with

MATLAB, our teaching experiences show that two introductory labs focusing on MATLAB programming, functions/sub-functions, and debugging can bring the majority of students to a satisfactory level of MATLAB programming. Thirdly, the MATLAB-ROS-Gazebo platform allows algorithms to be developed first with a simulator for quick prototyping. As a result, students can refine their algorithms outside of the classroom. Fourthly, the MATLAB-ROS interface can be connected to a wide range of ROS-supported hardware, such as the TurtleBot, allowing the algorithm developed on MATLAB to work with a physical robot.

Our ultimate objective is to use the MATLAB-ROS interface for the two projects, starting with the Gazebo simulator and moving on to the physical robot. This paper describes several options for the project on autonomous mobile robots using the MATLAB-ROS-Gazebo environment. Our next step is to use the MATLAB-ROS interface to connect to a physical mobile robot, either a home-built robot or a commercially available robot such as TurtleBot. Similar procedures are to be used to develop the project on a robotic manipulator, i.e., starting with the MATLAB-ROS-Gazebo environment and then moving to the MATLAB-ROS-physical-robot setting. These remain as directions for future investigations.

Setting Up the MATLAB-ROS-Gazebo Simulation Platform

The simulation system was successfully set up as shown in Fig. 1 by following the instructions on MathWork's website on how to set up the MATLAB-ROS-Gazebo platform [14]. Some details are given below:

Installation of required MATLAB toolboxes: When learning how to setup and use the MATLAB-ROS-Gazebo platform, we started by following the examples posted on MathWork's website. For those sample codes and commands, MATLAB will pop up a hint, indicating that certain toolboxes are required, and asking if you wish to install them. Follow those installation steps by saying yes. As with other MATLAB adds-on, installation is standard and straightforward.

Setting up the MATLAB-ROS-Gazebo simulation system:

- Download the VMware Player software. MathWork's website states version 16 at the time of writing. Versions 16 and 17 have been used. They both work well.
- Download the archive containing the virtual machine from MathWork's website [14]. Unzip the archive. Open it using the VMware Player. Start the virtual machine. Select "I copied it" if a dialogue shows up asking if you copied or moved the virtual machine.
- The virtual machine's IP address will be displayed. It may look like:

```
Virtual Machine IP
192.168.1.158 2600:4041:5a0d:4a00:2e8e:4d39:5841:212a 2600:4041:5a0d:4a00:a0cc:e6c:fb67:12b3
For administrative changes
- User name: user
- Password: password
```

We used the following commands to disable virtual machine's IPv6 address:

```
sudo sysctl -w net.ipv6.conf.all.disable_ipv6=1
sudo sysctl -w net.ipv6.conf.default.disable_ipv6=1
```

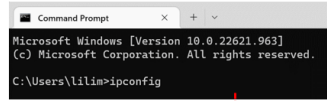
- Start the connection between the MATLAB ROS node and the Gazebo simulator:

```
roshutdown;
rosinit('192.168.1.158', 'NodeHost', '192.168.1.154');
```

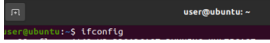
The first IP address is the IPv4 address of the virtual machine. The second IP address is the IPv4 address of the computer running MATLAB.

- Once connection is established, the command “rostopic list” on the MATLAB side will list all the available topics provided by the simulated robot.
- If the algorithm on the MATLAB side needs to create a ROS action client, the command “/start-turtlebot-move-action-server.sh” needs to be run on the virtual machine side.

The IP address of the computer running MATLAB. Can be obtained by typing ipconfig in the host computer's command window:

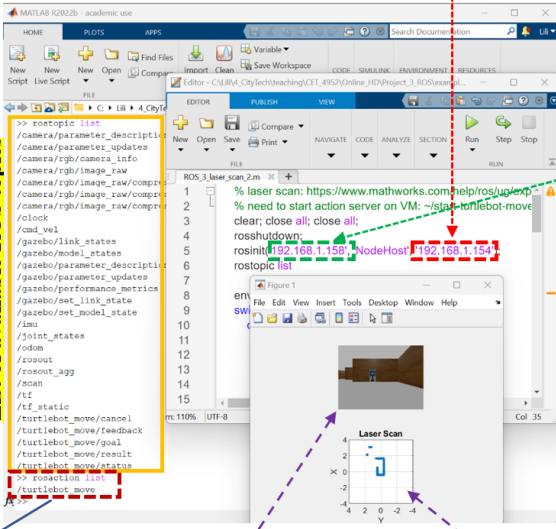


The IP address shown on the virtual machine:



If instead an IPv6 IP address is displayed, use the following commands on the virtual machine's terminal to disable the IPv6 IP address:

```
sudo sysctl -w net.ipv6.conf.all.disable_ipv6=1
sudo sysctl -w net.ipv6.conf.default.disable_ipv6=1
```



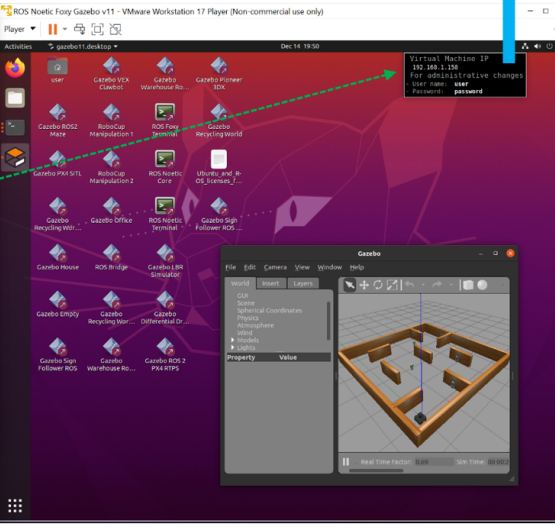
List of available ROS topics

ROS Action List

Image captured by the robot's onboard camera

360° laser scan data

MATLAB



Virtual Machine (Gazebo Simulator)

Figure 1: Setting up the MATLAB-ROS-Gazebo simulation platform.

Now the MATLAB-ROS-Gazebo platform has been setup. The MATLAB node will be able to receive messages published by the simulated robot (odometry, laser scan data, images), as well as sending commands to the robot to change its behavior (either by changing the robot's velocity or sending an action goal). More details will be presented when describing the three projects.

Our MATLAB-ROS-Gazebo simulation system was set up on the Windows operating system. We also installed the whole system on a Mac Pro Laptop, but not following the instructions particularly for the Mac Operating System, but used a Book Camp Assistant to allocate a Windows partition on the Mac computer's hard disk and then installed the Windows Operating System (Fig. 2 (a)). Once the simulator is up running, connection can be established between this simulator to more than one MATLAB node. Figure 2 (b) shows that MATLAB running on two different computers (the right two in (b)) can receive messages from the same simulated robot (the leftmost one in (b)).

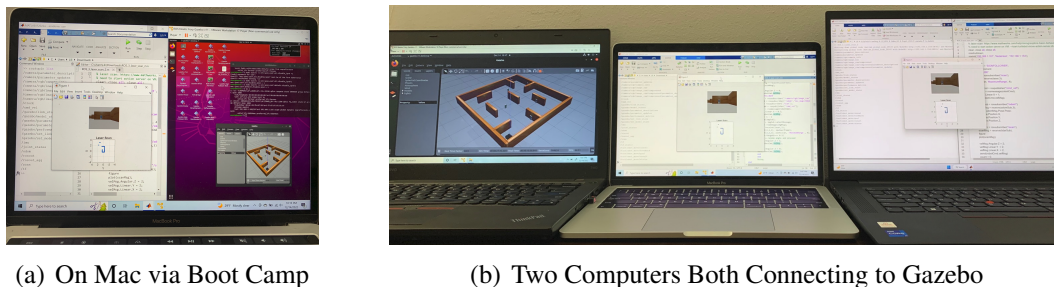


Figure 2: MATLAB-ROS-Gazebo.

Project 1: Model-Based Fitting of Laser Scan Data

This project exposes students to fundamental sensing and perception routines, including the collection of laser range sensor data, segmentation, and curve-fitting, as part of higher-level perception processes. This project is performed on the MATLAB-ROS-Gazebo platform using the “Gazebo Office” environment (Fig. 3) [14]. The robot is first commanded to wander around, scanning laser data to sense its environment. Data collected while the robot is at several different locations and orientations are accumulated together to have a better representation of the environment. The segmentation and curve-fitting (more specifically, circle-fitting) are described below.

Distance-based segmentation: Using distance as the criterion, laser points are grouped as one cluster if they are close to each other (i.e., the distance in between is less than a specified threshold). Figure 4 (a) presents the clustering result, where a totally nine clusters are identified, each plotted in a different color. The threshold of the minimal distance was 0.5 (m). For each new laser point, we first compute the minimal distance between it with all existing clusters. If the minimum of these distances is greater than the specified threshold, this point belongs to a new group, otherwise, it is assigned to the one closest to it. Setting the thresholds higher will result in fewer clusters.

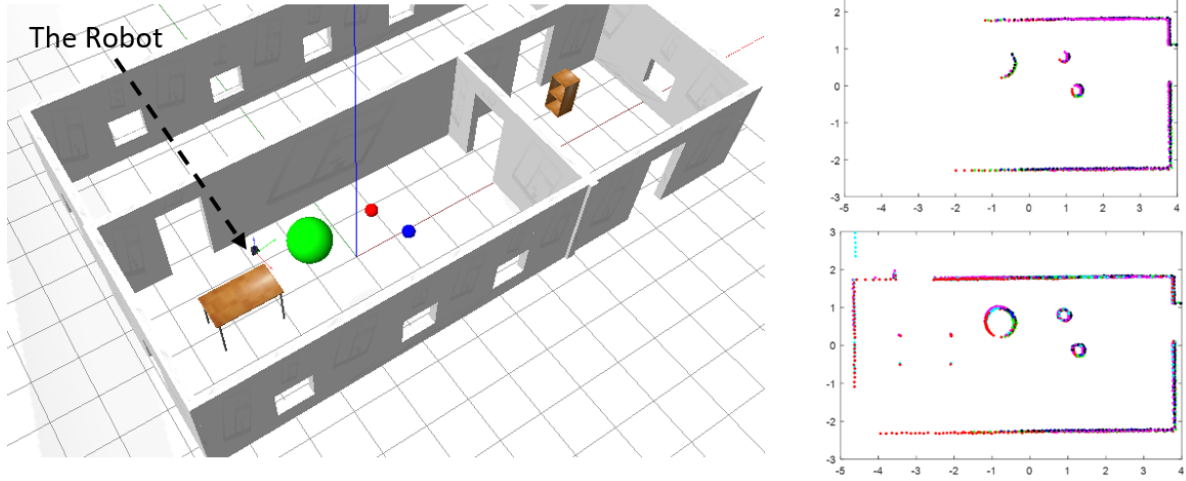


Figure 3: The Gazebo Office simulation environment.

Model-based fitting: After clustering and segmentation, a Least Square method can be applied to curve fitting. Using this method, the circle fitting problem is equivalent to solving

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix}}_A \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_X = \underbrace{\begin{bmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \\ \vdots \\ x_n^2 + y_n^2 \end{bmatrix}}_B, \quad (1)$$

where n denotes the total number of points, $a = 2x_c$, $b = 2y_c$, $c = \frac{\sqrt{r^2 - x_c^2 - y_c^2}}{2}$, (x_c, y_c) denotes the coordinates of the circle's center, and r is the circle's radius. Estimate of X , denoted by \hat{X} , can be computed using pseudo-inverse as $\hat{X} = A^T(AA^T)^{-1}B$. At this point, it only remains to calculate

$$x_c = \frac{a}{2}, \quad y_c = \frac{b}{2}, \quad r = \frac{4c - a^2 - b^2}{2}. \quad (2)$$

For laser points in clusters 3, 4, and 5, fitted circles are plotted in red as shown in Fig. 4 (b).

The MATLAB commands to retrieve laser scan data and then plot it are given below:

```
laser = rossubscriber('/scan');
scan = receive(laser, 3);
plot(scan, 'MaximumRange', 4);
```

Laser data collected at different locations and orientations are slightly different. This phenomenon shows that the simulator has taken into account the motion dynamics of the robot, thus resulting in a simulation platform of high fidelity.

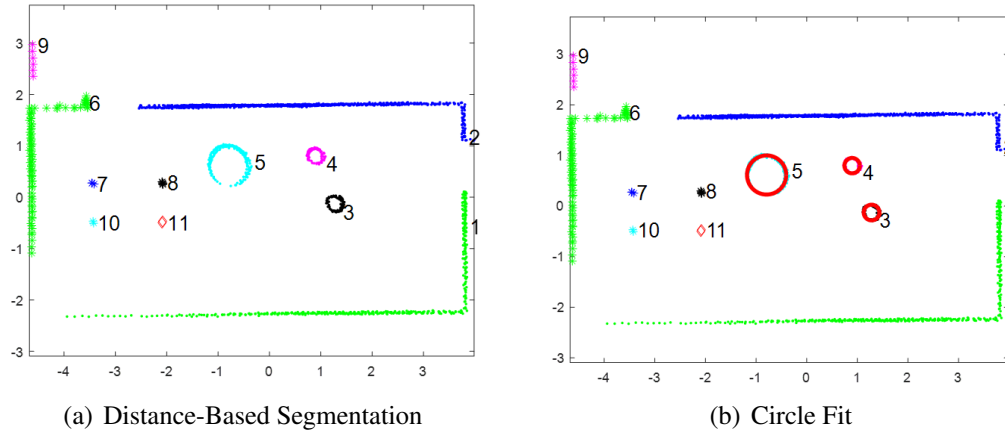


Figure 4: Model-based data fitting to laser scan data.

Project 2: Path Planning

In this project, students will perform path planning of an autonomous mobile robot working in an unknown environment. Students will go through the complete process of programming the robots to sense, perceive, make decisions, and take action. The project was implemented on the MATLAB-ROS-Gazebo platform using the “Gazebo Sign Follower ROS” environment. The assumption is that an autonomous mobile robot has an onboard laser range sensor. There are some unknown static obstacles in the robot’s workspace. The control task is to program the robot to detect objects in its workspace, utilize some path-planning algorithm to find a path to a specified location, and then let the robot follow the path. Path planning is to find a trajectory going from one point to another, given either completely known or partially-known information about the robot’s environment. The Wavefront algorithm is the most basic but still powerful approach.

Sensing the environment: The robot starts with no knowledge of its surroundings, so its first step is to wander around, collecting some information. Figure. 5 shows a sequence of laser scan data that are obtained by the robot at several different locations, starting with one scan of 360° as shown in Fig. 5 (a), three scans in Fig. 5 (b), and six scans as shown in Fig. 5 (c). Laser scan data indicate the locations of static objects that need to be avoided by the robot.

Setting up and propagation through the Wavefront map: Following the Wavefront algorithm’s convention by representing the robot’s current location by “R”, open spaces by “0”, spaces occupied by objects by “1”, and the goal location by “2”, the initial setup of the Wavefront map is shown in Fig. 6 (a), where grids without any value denote empty spots. Figure 6 (b) shows the Wavefront map after propagation. An open-space path is searched by performing “counting down”, starting from the robot’s current location and leading to the goal location. We highlighted the found path by changing the grids on it to green. A simulated robot was displayed to indicate the robot’s orientation upon execution. In Fig. 6, the neighbors of a grid include the one above, below, to its left, and right. The user specifies the goal location by clicking on the updated wavefront map.

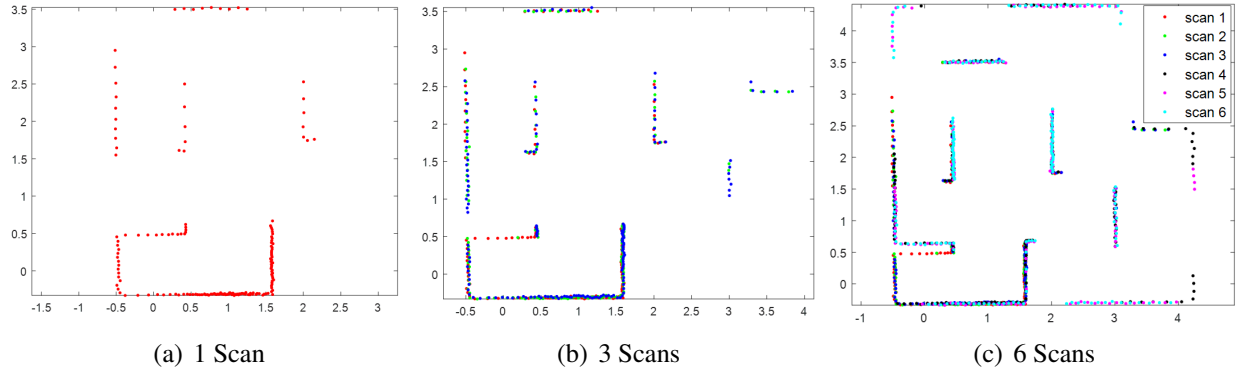


Figure 5: Integration of laser data.

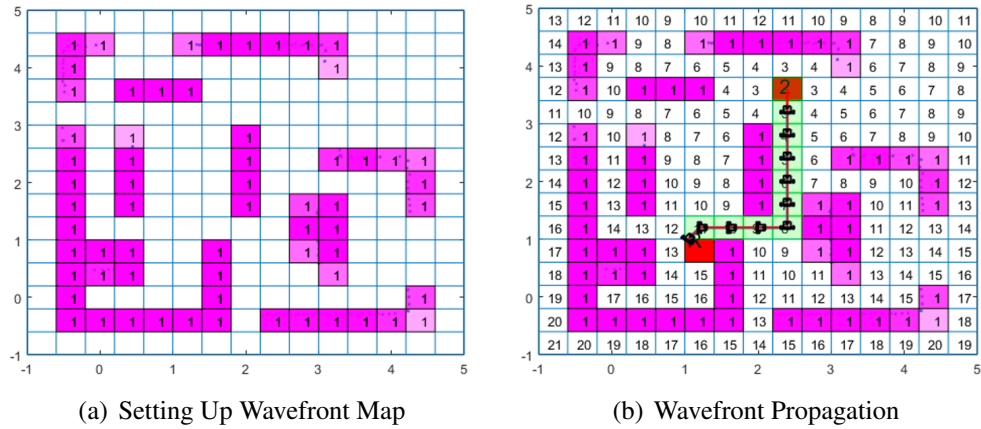


Figure 6: Path planning using Wavefront algorithm.

Commanding the simulated robot to execute the planned path: The path planning algorithm outputs a sequence of waypoints with x - and y - coordinates. This step commands the robot to navigate through this list of waypoints. For every two adjacent waypoints, the robot first spins in place to orient itself to face the second waypoint. It then translates toward it. The iteration continues till the robot reaches the goal location. Figure 7 shows several snapshots of the robot's movement.

The example demonstrated in Fig. 6 used a goal location close to the robot's initial position. Our experiences working with ROS show that the robot's orientation sensor accumulates errors quickly when turning. The accumulated error will make the robot's interpreted orientation less accurate. If the goal location is far away, the robot has to take more maneuvers to reach there. Due to its inaccurate orientation, the robot will likely run into objects, making the trajectory following hard to achieve. On the other hand, the scenario reveals the real issues that physical robots encounter, confirming that the ROS-Gazebo platform provides a high-fidelity simulation environment.

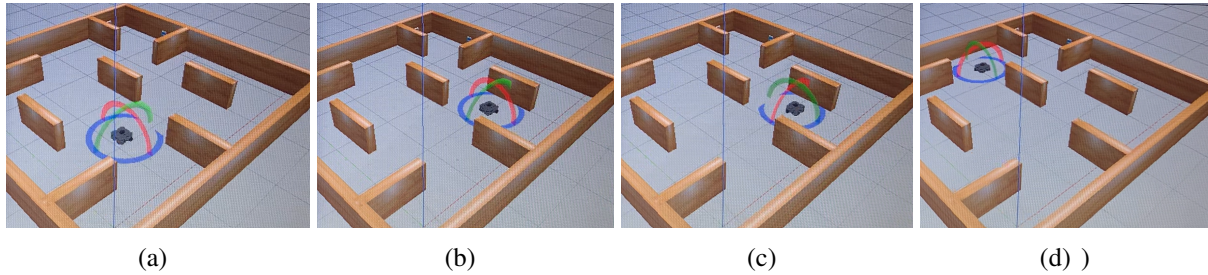


Figure 7: Snapshots of robot's movement.

Commanding the robot to rotate and translate are via the “sendGoalAndWait()” function:

- The translation command (i.e., moving forward with a distance “dist”) is achieved by:

```
goalMsg.ForwardDistance = dist;
goalMsg.TurnDistance = 0;
client.FeedbackFcn = [];
[resultMsg,~,~] = sendGoalAndWait(client,goalMsg);
```

- The rotation command (i.e., spinning in place for an angle “AOR”) is achieved by:

```
goalMsg.ForwardDistance = 0;
goalMsg.TurnDistance = AOR;
client.FeedbackFcn = [];
[resultMsg,~,~] = sendGoalAndWait(client,goalMsg);
```

In addition to commanding the robot's movement by sending translation and rotation commands, we can also control its movement by varying its velocity. For the autonomous robots working in the 2D Cartesian space:

- Its forward-moving velocity can be changed by:

```
velMsg.Angular.Z = 0;
velMsg.Linear.X = 0.3;
send(robotCmd,velMsg);
```

- Its spinning velocity can be varied by:

```
velMsg.Linear.X = 0;
velMsg.Angular.Z = 0.3;
send(robotCmd,velMsg);
```

In this project, the robot frequently needs to acquire its current pose (position and orientation). For example, the robot needs to know its gesture (both location and orientation) when integrating laser scan data collected at different locations, that is, to transform from the robot's body-fixed frame to the inertial world frame. The robot's onboard odometry sensor can provide such information, as shown below, where (x, y) denotes the robot's position and ψ (yaw) denotes the robot's orientation w.r.t. the positive x -axis.

```
odomMsg = receive(odomSub, 3);
pose = odomMsg.Pose.Pose;
x = pose.Position.X;
y = pose.Position.Y;
q = [pose.Orientation.X, pose.Orientation.Y, ...
     pose.Orientation.Z, pose.Orientation.W];
[a, b, yaw] = quat2angle(q);
```

Project 3: Vision-Based Control

This project is to close the loop by using information extracted from images captured by the robot's onboard camera to guide the robot's behavior, i.e., to achieve vision-based control. We implemented this project using the "Gazebo Sign Follower ROS" environment. In this simulated environment, there are some left-turn and right-turn signs. Following these signs, the robot will navigate through the maze.

One way to achieve vision-based control tasks is to utilize the information extracted from the image plane to directly control the robot's motion (i.e., image-based). Another approach is to transform this information into a 2D/3D world and use that information to command the robot (i.e., position-based). Researchers even investigated the combination of both. As a demonstration of vision-based control, we adopted the image-based approach in this project. Our control logic is quite simple. The robot turns right 90° when seeing a right-turn sign and turns left 90° upon seeing a left-turn sign. This simple logic fulfills the objective of exposing vision-based control to undergraduate students, illustrating the process of sensing, perception, decision-making, and execution.

As a starting point for image processing, a simple color-based feature extraction routine has been used that extracts and distinguishes the two signs simply by their color [15, 16]. For future work, we plan to adopt artificial intelligence/machine learning/Neural Network-based approaches to recognize these two signs by going through a pre-training process. Subscribing to receive images from the robot is given below:

```
imgSub = rossubscriber("/camera/rgb/image_raw", "sensor_msgs/Image");
receive(imgSub);
imgMsg = imgSub.LatestMessage;
frame = readImage(imgMsg);
```

Figure 8 illustrates the routine that distinguishes the signs by their unique color. Figure 8 (a) shows the original image captured by the robot. Figure 8 (b) shows the binary image after applying pre-specified thresholds (both lower and upper bounds) to the hue channel for the color of interest. We then performed contour detection, as shown in Fig. 8 (c). The contour with the maximum number of points represents the detected feature of interest, whose center is displayed using a cross.

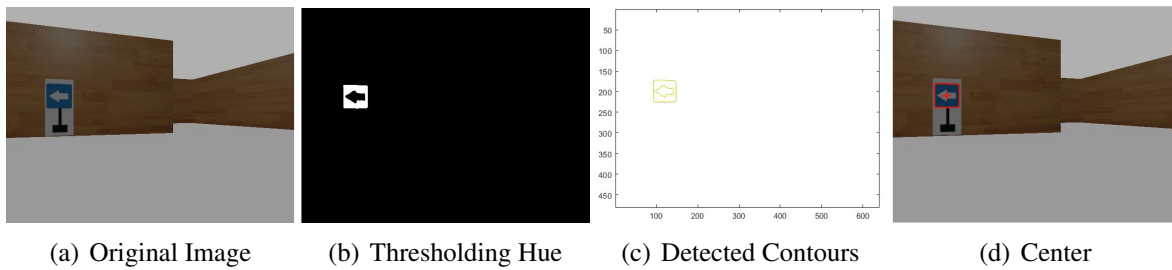


Figure 8: Color-based feature extraction.

Figure 10 shows the robot's motion. The robot turns 90° when the detected feature is big enough, indicating the robot is close to the sign. Otherwise, the robot moves forward with its current orientation. Changing the robot's orientation is achieved by varying its angular velocity about the vertical z -axis. By setting this angular velocity to be positive, the robot will rotate counter-clockwise; and clockwise otherwise. We constantly check the robot's orientation to see if the desired amount of rotation has been obtained. If so, we reset the robot's angular velocity to zero, and set its linear velocity to be positive so that the robot will move forward.

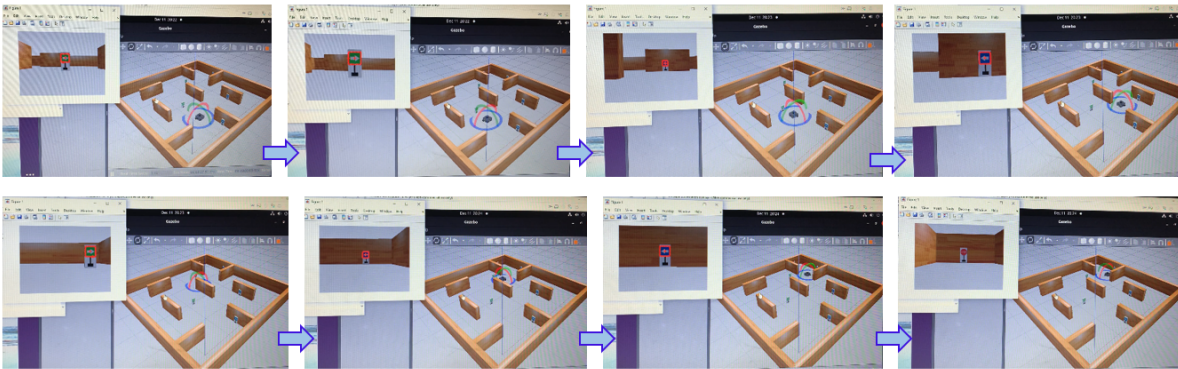


Figure 9: Snapshots of the robot's behavior during sign-following.

Inclusion in the Robotics Technology Course

The context of this work is an undergraduate robotic course (CET 4952: Robotics Technology) offered as a technical elective to senior-level students in the department of Computer Engineering Technology (CET). Driven by the transitioning from traditional in-person teaching to online due to the pandemic, a preliminary version of the MATLAB-ROS-Gazebo projects was introduced to this robotic course in Fall 2020 and Fall 2021. In each semester, four students successfully set up the simulation platform on their own computers, following the instructions on the MathWorks website under the instructor's remote supervision. Students were given sample codes. They executed the program, collected sensor data, and commanded the robot to move in the simulator. Due to time limit of two to three weeks, students were not required to develop their own algorithms.

The projects described in this paper are updated materials for introducing ROS. The instructor can decide how many projects to include in their course (one, two, or three), depending on how much focus they want to place on ROS. For our robotic course (CET 4952), we typically ask the students to complete three projects in one semester. The first project is on autonomous mobile robot (using physical robots while in-person), the second on robotic manipulator (also using physical robots while in-person), and the third on advanced robotic control tasks (using either physical robots or simulations). These MATLAB-ROS-Gazebo projects were intended to be used as one option for the third project (i.e., the final project).

The following summarizes the **timeline** of our offering of the ROS-projects:

- Fall 2020 (online): First time offering ROS-based simulation projects; a preliminary version; 4 out of 20 students successfully performed the ROS project; students installed the platform on their own computers.
- Fall 2021 (hybrid but primarily online): Second time offering ROS-based simulation projects; still a preliminary version; 4 out of 10 students successfully performed the ROS project; students installed the platform on their own computers.
- Fall 2023 (in-person for lab sessions): Third time to offer ROS-based simulation projects; an updated and more systematic version; the MATLAB-ROS-Gazebo simulation platform will be installed on lab computers, making the ROS-projects more accessible/available to students. The cap for this course is 20. Before the pre-pandemic, the class was usually full.
- In the future: We will continue offering the ROS-projects as one option for the final project.

In Fall 2020, we collected students' opinions, via anonymous zoom polls, on how this MATLAB-ROS project helped them to understand and implement more advanced robotic control algorithms. The survey question has four choices: "Absolutely help" (score 4), "Certainly help" (score 3), "Just help to some extent" (score 2), and "Does not help at all" (score 1). There were 19 students who participated in this survey. Students' responses are presented below. Considering the challenge of the MATLAB-ROS project, we think that the satisfaction rate of 68% is encouraging for bringing ROS into undergraduate robotic courses/curriculum.

# of Students Participated	Score (4)	Score (3)	Score (2)	Score (1)	Satisfactory (Score 3 or 4)	Unsatisfactory (Score 1 or 2)
19	26%	42%	7%	5%	68%	32%

Figure 10: Survey results of students' satisfaction of the MATLAB-ROS project (Fall 2020).

To continue evaluating the effectiveness of the ROS-projects regarding whether they successfully sustain and promote students' interests in robotics, we plan to conduct this type of surveys regularly each fall. The survey questionnaire may include questions such as:

- Does the MATLAB-ROS project help you understand data acquisition onboard the robot (acquisition of laser range data and image sequences)?
- Does the MATLAB-ROS project help you understand post-processing of acquire data (clustering, fitting)?
- Does the MATLAB-ROS project help you understand the concept/idea of closed-loop control (sensing, perception, and motion control)?
- Does the MATLAB-ROS project help you practice implementation of more advanced robotic control algorithms (waypoint navigation, obstacle avoidance, vision-based control)?
- Does the MATLAB-ROS project help to sustain/promote your interests in the STEM field?

Conclusions and Future Work

The **purpose** of this work is to introduce industrial and research-level practices for robotic programming and development on the open-source Robot Operating System (ROS). Our first step in bringing ROS into the undergraduate robotics curriculum is through simulations. This paper presents three simulation projects developed on the MATLAB-ROS-Gazebo platform. These projects cover the fundamental components needed to control an autonomous mobile robot, integrating sensing, perception, decision-making, and low-level motion control. Closed-loop control of the robot's behavior was obtained, demonstrating that other algorithms, such as localization and obstacle avoidance, could also be implemented.

The MATLAB programming environment, which students may have already used in earlier courses, also provides access to many other toolboxes (such as image processing, navigation, and artificial intelligence) that assist in fast algorithm verification and validation. It will significantly shorten the learning curve when compared with other programming languages such as C++, Python, and Java. The MATLAB-ROS-Gazebo environment already handled the complex visualization of the real-world environment and robots' dynamics. This leaves the users with the only role of algorithm development. These ROS-based simulation projects were designed as (one option) for the final project in our robotic course (CET 4952: Robotics Technology). We believe

they will also help other educators develop ROS-based simulation projects as part of a course or a stand-alone course for teaching robotics.

For future investigations, our next step is to bring in physical robots, demonstrating the complete picture of developing algorithms first on simulators and then deploying them to the physical robots. Another direction is to extend to robotic manipulators.

References

- [1] M. Galli, R. Barber, S. Garrido, and L. Moreno, "Path planning using MATLAB-ROS integration applied to mobile robots," in *IEEE International Conference on Autonomous Robot Systems and Competitions*, 2017, pp. 98–103.
- [2] W.-J. Tang and Z.-T. Liu, "A convenient method for tracking color-based object in living video based on ros and MATLAB/Simulink," in *International Conference on Advanced Robotics and Mechatronics*, 2017, pp. 724–727.
- [3] R. L. Avanzato, "Development of a MATLAB/ROS interface to a low-cost robot arm," in *ASEE Annual Conference and Exposition*, June 2020.
- [4] L. J. F. Rivera and B. Chandrasekaran, "ROS-MATLAB interface and setup for a fault tolerant robotic system using human robot interaction," in *Annual Computing and Communication Workshop and Conference*, 2020, pp. 0568–0575.
- [5] A. Araújo, D. Portugal, and M. S. Couceiro, "Integrating arduino-based educational mobile robots in ROS," *Journal of Intelligent and Robotic Systems*, vol. 77, p. 281–298, 2015.
- [6] A. Yousuf, W. Lehman, M. A. Mustafa, and M. M. Hayder, "Introducing kinematics with robot operating system (ROS)," in *ASEE Annual Conference and Exposition*, Seattle, WA, June 2015.
- [7] K. A. Khan and J.-C. Ryu, "ROS-based control of a manipulator arm for balancing a ball on a plate," in *ASEE Annual Conference and Exposition*, Columbus, OH, June 2017.
- [8] S. A. Wilkerson, J. Forsyth, and C. M. Korpela, "Project based learning using the robotic operating system (ros) for undergraduate research applications," in *ASEE Annual Conference and Exposition*, June 2017.
- [9] R. L. Avanzato and C. G. Wilcox, "Work in progress: Introductory mobile robotics and computer vision laboratories using ROS and MATLAB," in *ASEE Annual Conference and Exposition*, Salt Lake City, UT, June 2018.
- [10] Y. Wang, Z. Zhang, and Y. Chang, "A project-based platform for students' robot operation system (ROS) programming experience," in *ASEE Annual Conference and Exposition*, Minneapolis, MN, June 2020.
- [11] Y. Hold-Geoffroy, M.-A. Gardner, C. Gagné, M. Latulippe, and P. Giguère, "ros4mat: A Matlab programming interface for remote operations of ros-based robotic devices in an educational context," in *International Conference on Computer and Robot Vision*, 2013, pp. 242–248.

- [12] A. Shekar, "Project based learning in engineering design education: Sharing best practices," in *ASEE Annual Conference and Exposition*, Indianapolis, IN, June 2014.
- [13] E. H. Fini, F. Awadallah, M. M. Parast, and T. Abu-Lebdeh, "The impact of project-based learning on improving student learning outcomes of sustainability concepts in transportation engineering courses," *European Journal of Engineering Education*, vol. 43, 2018.
- [14] MathWorks, "Get started with Gazebo and simulated TurtleBot," [Online] <https://www.mathworks.com/help/ros/ug/get-started-with-gazebo-and-a-simulated-turtlebot.html>.
- [15] R. Raof, Z. Salleh, S. Sahidan, M. Mashor, S. Noor, F. Idris, and H. Hasan, "Color thresholding method for image segmentation algorithm of ziehl-neelsen sputum slide images," in *International Conference on Electrical Engineering, Computing Science and Automatic Control*, 2008, p. 212–217.
- [16] H. Jia, J. Ma, and W. Song, "Multilevel thresholding segmentation for color image using modified moth-flame optimization," *IEEE Access*, vol. 7, pp. 44 097–44 134, 2019.