Fusion of LiDAR and Computer Vision for Autonomous Navigation in Gazebo

***Abstract*—Simulating hardware continues to accelerate technology development to either provide initial proof of concept or function as a substitute when either hardware is either unavailable or damaged. Building an efficient simulator only requires basic real-world representations to provide feedback to software development. This report discusses and tests the application of object detection and autonomous navigation on a Turtlebot3 Waffle platform in a simulated Gazebo environment. Object Detection and mapping are supported using Turtlebot's Camera, Lidar, and IMU sensor. Testing for the first phase of development demonstrated the feasibility of simulation as applied to the IEEE Hardware Competition and successful detection and navigation between objects placed in a simulated, open environment using the methods proposed in this report. Follow-on work with this simulator includes posting to the IEEE SoutheastCON 2023 discord site, updating the world parameters to better model the specifications of the IEEE Hardware competition, and embedding the object detection algorithms directly into the navigation behavior tree.**

***Index Terms*—Gazebo, Autonomous Navigation, Turtlebot, LiDAR, YOLO, SLAM**

I. Introduction

The focus of this research will be on using a simulated environment to model the IEEE SoutheastCon Hardware Competition. Gazebo will be the simulation software used to support this. Simulation provides numerous advantages over real-world testing. Some of these advantages include proof of concept testing, reduced testing costs, and the ease that comes with sharing simulated testing results [1]. Simulation also provides the advantage of it being relatively easy to construct artificial testing environments that would otherwise be difficult to recreate in the real world.

At their core, robotics simulators are physics simulators used to examine how robots would interact with the world. Gazebo is one of the most widely used simulators today due to its compatibility with Linux-based operating systems, its open-source license, and compatibility with Robot Operating System(ROS). Gazebo's compatibility with ROS allows a simulated robot to receive commands in the same way it would in the real world. Because of this, there is often minimal difference between the programming of a simulated robot, and a real-world robot [2].

The IEEE SoutheastCon Hardware Competition models a duck garden that has been hit by a hurricane that must be rebuilt using an autonomous robot [3]. One possible solution to this problem is to have a robot map out the area and identify the objects within it. Once its surroundings are correctly identified and mapped, a robot should be able to easily move between destinations.

This method in effect would create two maps, one from LiDAR and the other from Computer Vision, that a robot must synchronize.

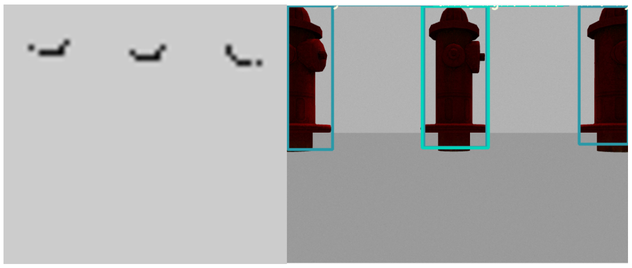

Fig. 1. A possible end state, where the robot is capable of creating a map with LiDAR data(Left), and identifying locations of objects(Right).

In the spirit of this competition, this paper will explore how LiDAR and computer vision technologies can be integrated into a Turtlebot3 Waffle robot to locate and identify objects in an enclosed area.

II. Theory

Using Gazebo, a world can be generated consisting of a Turtlebot3 Waffle and objects for the robot to move between. Commands will be published to Waffle using Robot Operating System 2 (ROS2) and python scripts. Information about the environment will be provided using Waffle's RGB Camera, LiDAR, and IMU sensor. Data collected from these sensors will be used in an algorithm to both name and locate objects.

ROS2 is a "set of software libraries that can be used to build robot applications" [4]. While the original ROS has seen wide applicability across a range of robotics applications, ROS2 is a newer version of the software that has yet to see as much widespread adaptation. ROS2 will be the framework that the robot discussed in this paper will be built on.

Turtlebot3 Waffle is a robot that is made available by the company Robotis, and is often used in experimental robotics. Of interest to this experiment, Waffle comes equipped with a camera, LiDAR, and IMU sensor [5]. These sensors will be the primary sensors used for a solution enabling object detection and pathfinding.

Using files provided by Robotis, a simulated version of Turtlebot3 Waffle will be generated in Gazebo. The simulated Waffle is modeled to have the same functionality as the robot in the real world, with factors such as its dimensions, LiDAR and RGB camera specifications, all being modeled after its real-world components.

A picture showing Turtlebot3 Waffle simulated in Gazebo is shown below in Figure 2. The blue fan surrounding waffle represents its LiDAR fan. The screen displayed in front of waffle represents the output of its RGB camera.

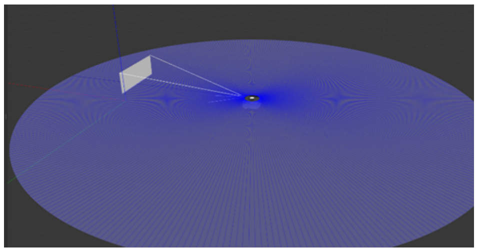

Fig. 2. Simulated Turtlebot3 Waffle

A. *LiDAR and SLAM Navigation*

The Turtlebot3 Waffle comes equipped with a LiDAR sensor which will be used for location finding and Simultaneous Localization and Mapping (SLAM). The LDS-01 LiDar sensor on Turtlebot3 collects range data in 360 degrees around the robot, with a maximum range of 120-3,500 millimeters. The LDS-01 has a maximum uncertainty of 5 percent, with a resolution of 1-degree [5]. This is significant as the LiDAR sensor will return a list of 360 values, where each value represents the distance until a collision, and the index of a value represents the angle that a collision is at.

Because the RGB camera is only capable of seeing directly in front of the robot, only the LiDAR data between the indexes of 45 and 135 will be considered in this paper.

Using the distance and the direction of a given object, a change in the X and Y position can be derived using trigonometry. The Equations for the change in X and Y positions are shown in Equations 1 and 2 respectively.

$$X = X_{original} + Distance \times cos(\theta) \tag{1}$$

$$Y = Y_{original} + Distance \times sin(\theta) \tag{2}$$

Along with determining the distances and directions of objects, the LiDAR sensor will also be used for SLAM Navigation.

SLAM is a method used in robotics to map out an environment and keep track of a robot's location as it moves around. SLAM Navigation simply refers to the process of using the generated map to enable movement between destinations. To enable SLAM Navigation, two ROS2 libraries will be used. The first is Cartography, which will allow Waffle to create a base map of its environment. This base map is required for the second library to function, which is Navigation2. Navigation2 will enable pathfinding and navigation using collision avoidance. This is done using Djikstra's algorithm to create and identify a path to a destination. Following this, the required commands are then published to the robot's wheel velocities via ROS2 [6].

The specific command that will be used in the Navigation2 package is ”Navigate to Pose with Replanning and Recovery”. Using this command, after initiating movement, Navigation2 will continue to update its path as it moves toward a destination to avoid any unexpected obstacles or unforeseen encounters. Navigation2 does this using a behavior tree. Behavior trees consist of a list of tasks that a robot must execute, along with the conditions for when each task must be executed. A robot will continue to test for conditions in the behavior tree until an end state is reached.

The behavior tree used for the Navigate to Pose with Replanning and Recovery is represented in Figure 3. In this behavior tree, there are subtrees. The first is the navigation subtree and the second is the recovery subtree. The navigation subtree is responsible for reaching the destination and the recovery subtree is responsible for dealing with any errors that the robot encounters along the way. When an error is encountered, the recovery subtree will respond by clearing its costmap. The costmap is a map SLAM creates wherein a value is assigned to each point on the map that represents how difficult it would be to move to or through that point. The robot will additionally spin and back up to remove itself from any direct collisions when the recovery tree is triggered.

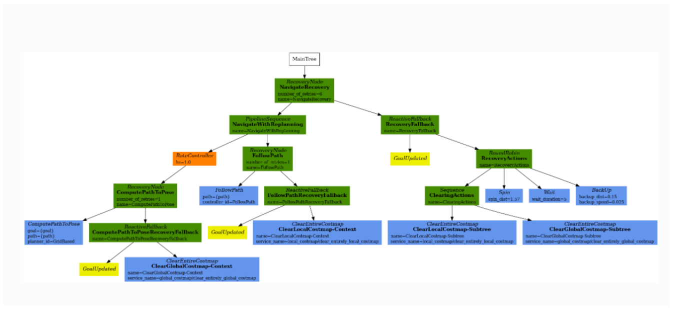
MainTree
RecoveryNode NavigateRecovery number_of_retries=6 name=NavigateRecovery
PipelineSequence NavigateWithReplanning name=NavigateWithReplanning
ReactiveFallback RecoveryFallback name=RecoveryFallback
RateController hz=1.0
RecoveryNode FollowPath number_of_retries=1 name=FollowPath
GoalUpdated
RoundRobin RecoveryActions name=RecoveryActions
RecoveryNode ComputePathToPose number_of_retries=1 name=ComputePathToPose
FollowPath path={path} controller_id=FollowPath
ReactiveFallback FollowPathRecoveryFallback name=FollowPathRecoveryFallback
Sequence ClearingActions name=ClearingActions
Spin spin_dist=1.57
Wait wait_duration=5
BackUp backup_dist=0.15 backup_speed=0.025
ComputePathToPose goal={goal} path={path} planner_id=GridBased
ReactiveFallback ComputePathToPoseRecoveryFallback name=ComputePathToPoseRecoveryFallback
GoalUpdated
ClearEntireCostmap ClearLocalCostmap-Context name=ClearLocalCostmap-Context service_name=local_costmap/clear_entirely_local_costmap
ClearEntireCostmap ClearLocalCostmap-Subtree name=ClearLocalCostmap-Subtree service_name=local_costmap/clear_entirely_local_costmap
ClearEntireCostmap ClearGlobalCostmap-Subtree name=ClearGlobalCostmap-Subtree service_name=global_costmap/clear_entirely_global_costmap
GoalUpdated
ClearEntireCostmap ClearGlobalCostmap-Context name=ClearGlobalCostmap-Context service_name=global_costmap/clear_entirely_global_costmap

Fig. 3. Navigate to Pose with Replanning and Recovery Behaviour Tree

By using Navigation2, the difficulties that come with pathfinding, navigation, and collision avoidance will largely be simplified. Navigation2 will only require an X and Y coordinate for it to move a robot to a destination.

B. *Camera Data and YOLO*

To allow for object detection, computer vision will be used to identify objects. Data from the included RGB camera will be processed using YOLOV5. The version of YOLO being used comes equipped with a preloaded COCO data set. This dataset is capable of identifying many objects; however, for applications requiring a more tailored data set, this can also be done [7]. While the IEEE Hardware competition specifies that ducks will be used in its environment, the COCO data set is not trained to recognize ducks so for this paper, fire hydrants will be used instead. Figure 4 shows a sample output from YOLOV5 being used to identify a fire hydrant.

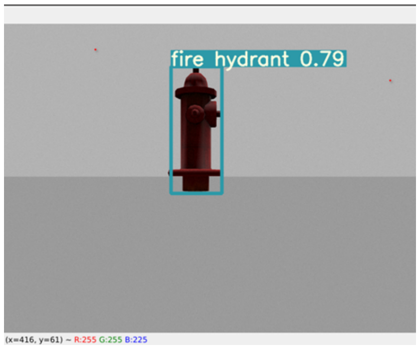

Fig. 4. Sample YOLOV5 Output with a fire hydrant. YOLO is 79% confident that it sees a fire hydrant and draws a box around where it thinks the fire hydrant is. The location of this bounding box will later be used to associate names with the locations of objects.

YOLO works by taking in data from an RGB camera and passing that data through its neural network to identify any objects that it is capable of recognizing. YOLO returns its results in the form of an object name, a confidence, and a bounding box, signifying the position of the object in the camera window.

YOLOV5 defines the bounding boxes by labeling the corners of the box. If the X coordinates of the corners are averaged to find the center of the bounding box, the robot then has an accurate idea of the center angle that a given object is at. In this experiment, Waffle will keep track of the locations of these bounding boxes, along with the names of the objects within them.

C. Integration of LiDAR and Computer Vision Data

While there has been extensive research on both SLAM Navigation and computer vision, there have been few papers documenting the integration between the two.

For the robot to complete its goal, it needs to be able to associate the names of objects found in YOLO with their X and Y coordinates found with the LiDAR calculations.

To do this, each YOLO detection will be matched with a LiDAR detection using an equation such that the average X-coordinate given by the YOLO bounding box can be related to the average LiDAR detection index. The equation used for this is shown in equation 3.

$$Lidar.Index = (YOLO.Index - 330) \times \frac{27}{330} \tag{3}$$

The result of this equation will be a directory of objects detected, each with a name, an X coordinate, and a Y coordinate.

Equation 3 represents a direct translation between LiDAR and camera coordinates. It is important to note that this equation will only work as applied to this scenario because both the focal length and principle point of the camera used are 0. If a different camera were to be used, or these parameters were to be changed in Gazebo, a pinhole model of a camera would need to be used [8]. This would result in a more complex version of equation 3.

III. Test Procedure and Equipment

To test the theory, a world was created in Gazebo with three fire hydrants and a Turtlebot3 Waffle robot. The waffle was spawned at the world origin, (0,0,0), and the fire hydrants were placed in front of Waffle such one was center-line, another was at the left extreme of the camera field of view, and the last was at the right extreme of the camera field of view.

Following these emplacements, the LiDAR data and computer vision data were collected. This data was used to calculate an estimated position for each of the three fire hydrants using equations 1 and 2, along with the estimated LiDAR angle using equation 3. In addition to the calculations, the position of each of the fire hydrants was measured in Gazebo using the built-in measurement tool. Taking the true position data allows for a comparison between the real and estimated positions.

For the purposes of this paper, these calculations will only be performed once, when the robot is initially spawned into Gazebo. In a more complex simulated or real-world environment, Waffle could be directed to repeat these calculations as it is moving to a destination, in order to correct for prior miscalculations or potentially improve accuracy.

After the setup was complete, the LiDAR and Computer Vision data were analyzed to confirm that the number of detections matched the number of fire hydrants, Waffle was told to move to each of the three fire hydrants from left to right using the navigate to pose Navigation2 feature. Screen captures were taken after each movement to determine the judge the accuracy of this.

IV. Results and Error Detection

The LiDAR data was returned as a list of 90 values. If there was a LiDAR detection at a given index a number was returned, and if not, the value "inf" was returned.

For each detection, Table 1 shows the average angle and distance to each detection identified by the LiDAR sensor

Detection	Average Angle	Average Distance
1	-27	2.036
2	45	1.888
3	26.5	2.0385
Units	Degrees	Meters

TABLE I
Average Angle and Distance of each Hydrant.

Using Equations 1 and 2, a calculated X and Y position for each hydrant was found. This can be compared with the actual measured X and Y coordinates. This is shown in Table 2. The percent errors associated with each X and Y calculation are additionally shown in Table 3.

Fire Hydrant	X Pos	Y Pos	Calc. X Pos	Calc. Y Pos
1	2	-1	1.82	-0.91
2	2	0	1.88	0
3	2	1	1.82	0.91

TABLE II
Calculated X and Y positions vs. Measured.

Percent Error	X	Y
Hydrant 1	9.77%	10.15%
Hydrant 2	5.92%	0%
Hydrant 3	9.62%	9.99%

TABLE III
PERCENT ERRORS BETWEEN CALCULATED AND MEASURED X AND Y COORDINATES.

The data from the LiDAR sensor and subsequent calculations represent the robot's estimation of the number detections along with their locations. These results can be combined with the results from YOLO to identify what each detection is. Figure 5 displays an image of the camera output after YOLO processing.

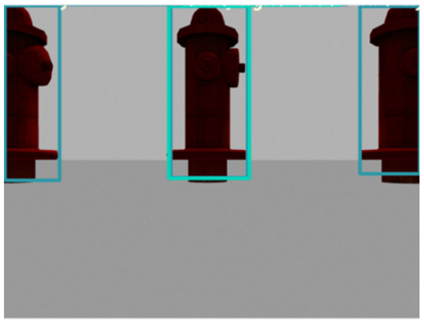

Fig. 5. YOLO Output from the RGB Camera during testing.

In Figure 5, three bounding boxes are drawn around each of the fire hydrants. The X coordinates of the bounding boxes for each of these detections, from left to right, are 43, 303.5, and 595. Further, each of these detections was correctly identified as a fire hydrant. Table 4 below shows the results of equation 3 as applied to these detections.

YOLO Calculated Angle	LiDAR angle	Difference in Angle
21.72	26.5	4.78
-2.17	5.92	8.09
-23.5	-27	-3.5

TABLE IV
YOLO CALCULATED ANGLES VS. LIDAR CALCULATED ANGLES

The results from Table 4 show that YOLO was able to correctly associate each of the LiDAR detections to each of the YOLO detections. There is some percent error as a result of these calculations; however, this is still largely insignificant as applied to this paper. This will be discussed further later on.

Following the calculations, three commands were published to Waffle telling it to visit each of the three fire hydrants in order. After Waffle reached each hydrant, a screen capture was taken showing the results. These are shown in Figures 6 through 8.

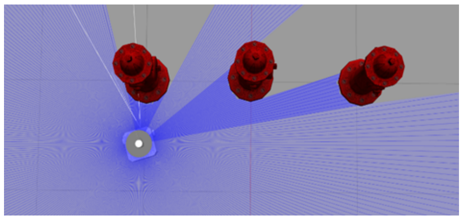

Fig. 6. Waffle after visiting the left hydrant.

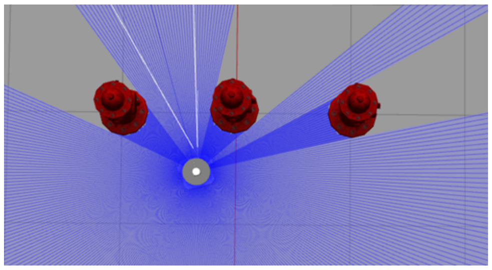

Fig. 7. Waffle after visiting the middle hydrant.

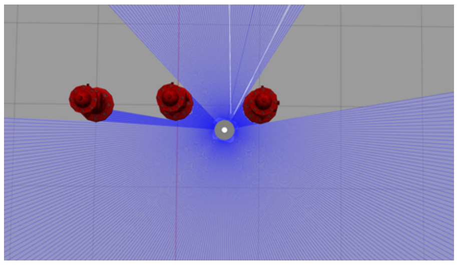

Fig. 8. Waffle after visiting the left hydrant.

Figures 6 through 8 show that Waffle could move to each of the hydrants after calculating the location of the hydrants, using SLAM Navigation.

The results from this experiment show that in a simulated environment using Gazebo, a Turtlebot3 Waffle was able to successfully calculate the location of, and move to three fire hydrants.

A. Parametric Error

Parametric error was present in this experiment in several forms. The first form was when the difference between the actual distance and the calculated distance with LiDAR data was considered. This was because the LiDAR sensor measured the distance between the robot and the nearest edge of each fire hydrant, while the actual location was listed as the center of where the fire hydrant was placed.

This error most prominently presented itself in tables 3 and 4. Measuring the actual location as the center of each hydrant was something done for simplicity, knowing that the true measure of error would be whether or not Waffle could move to each hydrant. To eliminate this error, the true x and y coordinates for each point along the circumference of each fire hydrant would have had to been measured. For this reason, this error was seen as largely acceptable.

B. Measurement Error

Measurement error in this experiment was likely minimal, but still present. The sensors in Gazebo were used to take all measurements. The instruments in Gazebo were designed to model their real-life counterparts; however, there is still likely some difference between modeled and real sensors. Additionally, some error was intentionally introduced to the LiDAR sensor in the form of Gaussian noise. This was done in the hopes of more closely modeling a real-world testing environment.

C. Modeling Error

Modeling error was present in this experiment as Gazebo was used as a simulator to model the real world. There will likely be some differences between the results of this experiment and an identical experiment conducted in the real world; however, the results of the simulation will still very likely closely model what would have happened in the real world.

V. Discussion

This experiment was able to show that autonomous detection on a Turtlebot3 Waffle model was possible by using LiDAR and Computer Vision data in a simulated environment. Further, Waffle was able to move to each of its detections when told to do so using SLAM Navigation.

While the results of this paper do not model a complete solution to a competition such as the IEEE Hardware competition, they do set a solid theoretical foundation for this development environment. Further with the aid of a robotic arm or another instrument, Waffle could be capable of moving and manipulating objects as is required in the IEEE Hardware Competition.

VI. Future Research

In the future, more work should be done to expand upon the results of this experiment, such that it more closely models the situation given in the IEEE Hardware competition or other similar competitions.

Applied to the IEEE Hardware competition, the next step forward should be to test the theory outlined in this paper in a simulated Gazebo environment that more closely models the parameters of the competition. Such a simulated world would look similar to the Gazebo world shown in Figure 9. The primary reason that the simulated world shown in Figure 9 was not used in this paper was that the COCO data set was not trained to recognize ducks. Training a data set to recognize ducks along with the various other objects that exist in the IEEE Hardware Competition is an essential next step to enable testing in a world such as this.

In addition to the IEEE Hardware competition, this simulator and supporting code will further be released to the IEEE Conference Discord site to encourage others to experiment with it and solicit feedback.

In addition to creating a more accurate world and updating the Computer Vision with a more applicable data set, future research should explore incorporating the methods explored in this paper directly into the Navigation2 behavior tree. If Waffle were able to classify and locate objects from the Navigation2 behavior tree, this could provide a more streamlined solution to the problem. In addition, nodes could be incorporated into the behavior tree that accounts for misclassification, something that the model explored in this paper does not do.

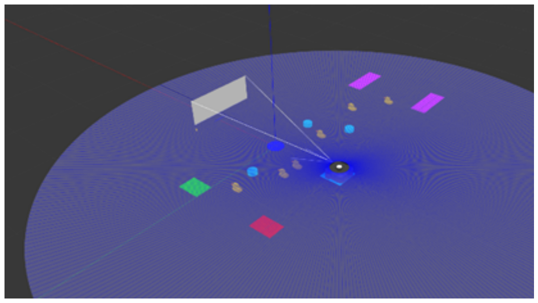
Fig. 9. Gazebo World Simulating Parameters Given in IEEE SoutheastCon Hardware Competition

Incorporating the object detection algorithm into the behavior tree is likely a step best saved for once the model is fully verified alone, as creating and adding nodes to behavior trees is a separate topic that is capable of producing errors independently of a detection algorithm.

VII. Conclusion

This paper proved that it is possible to simulate autonomous detection and navigation on a Turtlebot3 Waffle model in a Gazebo environment.

By first simulating a solution to the problem of object detection and navigation, this paper was able to provide proof of concept in a timely and cost-efficient manner. The steps highlighted in the future research section will be taken in the future to more accurately model the scenario given in the IEEE Hardware Competition, with the end state of having a real-world solution eventually built.

References

[1] "3 Advantages of Using a Robot Simulator," www.onlinerobotics.com, May 09, 2022. https://www.onlinerobotics.com/news-blog/3-advantages-using-robot-simulator (accessed Dec. 09, 2022).

[2] "Gazebo," gazebosim.org. https://gazebosim.org

[3] S. Hopkins, "IEEE SoutheastCon 2023 Hardware Competition Hurricane Alley 'CAT5 Robots Rebuild the Magic at Duck Gardens' Regina Hannemann, R3 SAC Hardware Competition Consultant," IEEE SoutheastCon 2023 Hardware Competition Revision, vol. 1, no. 4, 2022, Accessed: Nov. 20, 2022. [Online]. Available: https://ieeesoutheastcon.org/wp-content/uploads/sites/392/IEEE-SoutheastCon-2023-Hardware-Competition-Rules-v1.4.pdf

[4] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "ROS 2 Documentation — ROS 2 Documentation: Foxy documentation," docs.ros.org, May 07, 2022. https://docs.ros.org/en/foxy/index.html

[5] T. Lim, Y. Pyo, W. Son, and G. Kijong, "TurtleBot3," GitHub, Nov. 28, 2021. https://github.com/ROBOTIS-GIT/turtlebot3

[6] S. Macenski et al., "Nav2 — Navigation 2 1.0.0 documentation," navigation.ros.org. https://navigation.ros.org/

[7] G. Jocher, "ultralytics/yolov5," GitHub, Aug. 21, 2020. https://github.com/ultralytics/yolov5

[8] R. van den Boomgaard, “1.2. The Pinhole Camera Matrix — Image Processing and Computer Vision 2.0 documentation,” staff.fnwi.uva.nl, 2016. https://staff.fnwi.uva.nl/r.vandenboomgaard/IPCV20162017/LectureNotes/CV/PinholeCamera/PinholeCamera.html (accessed Dec. 19, 2022).