Portfolio Report: Perception in Robotics: Pursuit Evasion Game

Siddharth Nilesh Joshi
Robotics and Autonomous Systems (AI)
Ira A. Fulton Schools of Engineering
Arizona State University
snjoshi2@asu.edu

I. Introduction

Computer vision is one of the most interesting domains in robotics that uses digital images, videos, and tools like camera, lidars and radars to fuse data together and 'make-sense' of the real world environment. This domain is at the intersection of robotics, mathematics, and artificial intelligence. Through perception, a robot is able to localize and map the environment and he is then aware of the cause-effect of its interaction with the world. SLAM refers to Simultaneous Localization and Mapping which relates to the concept of knowing 'where you are' in the map and how does the map look like. Numerous algorithms and tools have been developed for localization, mapping, object detection, object recognition, etc. The perceptions layer of any robotics system consists of the initial segment of the pipeline. A robotics system usually works on the 'Sense-Plan-Act' strategy, wherein perception contributes to the 'sense' and perceive.

The idea for the pursuit evasion game in the course is about an autonomous robot that plays the role of a pursuer attempts to detect and follow the human – which plays the role of an evader that moves randomly and hides in the environment. The robot could be a TurtleBot3 Burger or Waffle, equipped with a camera sensor and the evader is capable to move in a pseudo-random way. The human starts moving randomly in the environment, and after some time the robot is activated to move autonomously and catch the human. The detection and follower algorithms need to be programmed onto the robot. Once the robot catches up to the human and contacts it, the simulation ends. To set-up this scenario, the idea was to build an environment in Gazebo along with the human and TurtleBot3 added. Setup the teleoperation and the sensor mount for the robot. Program robot perception and sensor nodes to communicate over the ROS framework. Implement the gmapping ROS package for Simultaneous Localization and Mapping (SLAM). And lastly, implement an object recognition and detection algorithm for the robot using the RGB-D camera sensor.

II. Solution

Robotic Operating System (ROS) is the middleware that is used as an additional layer between software and hardware. It allows developing nodes that allow sending and receiving messages over channels determined by the topic names. There are multiple protocols such as client/server, publisher/subscriber, etc. Built along with the ROS framework is the Gazebo Simulator that allows creating worlds, creating, and deploying robots and analyzing their interactions.

A. Creating a Gazebo World

The environment, which forms the 'playground' for our simulation is created in the Gazebo simulation software. The terrain, building along with the robots, and other dynamic and static elements can be developed using the Gazebo tool. The plan view of the environment is visible in the editing window. The walls, windows, doors, and pathways can be created in the plan view of the Gazebo. Once the basic layout of the building is ready, various objects like table, chair, human, trees, etc. can be added. The aesthetics of the house can be improved using colored walls and objects.

We designed a 2 bedroom, a kitchen and a living room house as shown in Fig. 1. An simpler variation of the world A tricycle was placed in one of the rooms, a bookshelf in the other room, a few abstractly colored-objects in the third room and the fourth room was left empty. The living room had a drone object, a table, and a human along with the TurtleBot3 robot. The world file consisted of the static objects and the house. The launch file was later created to load the world in the Gazebo environment and also spawn the instance of TurtleBot3. TurtleBot3 is a wheeled device with an ability to add the camera and other sensors on itself. Initially the TurtleBot3 could be teleoperated and maneuvered using the basic keyboard keys. This robot node created, would be later programmed to autonomously detect the human and follow it. The initial segment of the project concluded when the world was successfully loaded in the gazebo, and an instance of the robot with a camera and teleoperation capability was spawned through a launch file.

B. Creating nodes for robot, sensors and human

The next segment of the project comprises of creating ROS nodes for communicating with the simulation hardware. These nodes will be designed using ROS **[1]** packages and will majorly facilitate SLAM **[2]** and autonomous maneuvering. The TurtleBot3 **[3]** is equipped with a camera, which is used to map the environment through teleoperation once the environment is launched from the terminal, in Gazebo. The scanned environment obtained shows object and walls, halls and pathways that are accessible to the robot as shown in Fig. 2.

The nodes are created for the entities in the environment such as robot and the human. The pursuer (robot) has a camera node that will publish the stream of images over the camera topic, to which any required node can subscribe to. When the TurtleBot3 is teleoperated in the house space, it will generate a bare-bone structured-map of the environment.

The nodes designed centric to the robot were mainly camera (RViz) **[4]** node, teleop node, and tracker node. The camera module integrated with the robot is RViz and the teleop package and move_base package was used to execute the teleoperated controls of the same. The tracker node was

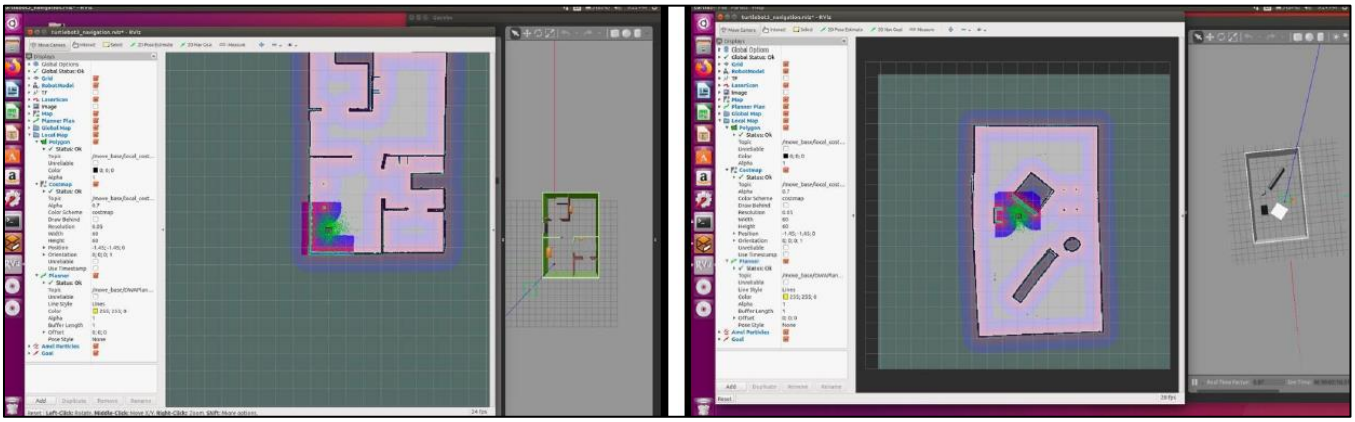

Figure 1 : World instances created in gazebo environment

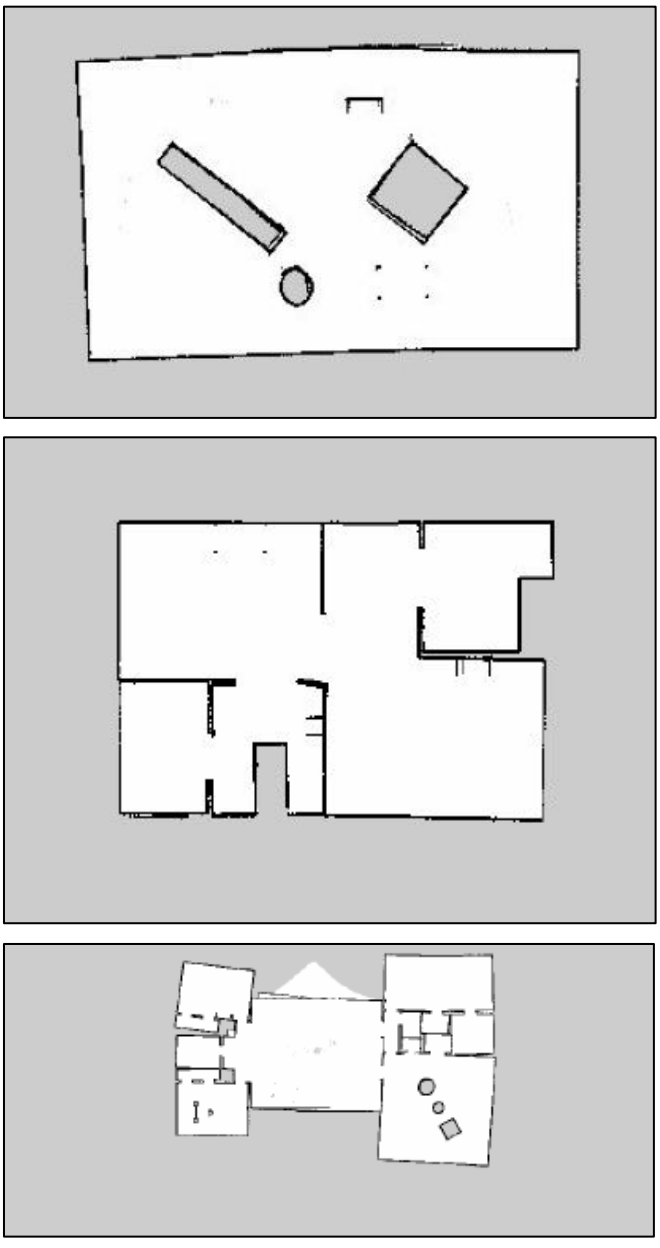

Figure 2 : Map layout scanned by the TurtleBot3

implemented through the help of cv_bridge ROS package that was able to bridge the messages from sensor node to the OpenCV compatible format. An object detection node analyzes the stream of images and detects the object (human, evader in our case). The object detection is implemented through a deep neural-network algorithm known as YOLO **[5]** darknet. Based on the image and LiDAR data input, the position and distance of the human is detected and traced using a bounding box. By tracing the position of a point on the human and comparing the distance between that point and the center of the bounding box, the location of the box always envelops the human. Hence, the robot is aware of the position and distance from the evader. Through image segmentation, the evader was perceived to be distinct from the other objects detected in the image frame. With the known position and distance of the human, the robot can trace the location precisely through a control node.

The human (evader) is programmed to move along a certain trajectory, however, this path is unavailable to the robot. This could be characterized as a pseudo-random movement of the human with respect to the robot. The position of the bot was accessed by subscribing to the amcl_pose topic, which was also available on the goal topic under move_base_simple. Based on the received data about the location and distance of the human the robot will follow the human, and the simulation will end once the robot reaches the human. The results obtained over ten runs were averaged to derive substantial conclusions.

III. RESULTS

The results obtained from the simulation were primarily targeted at the pursuit time, environment mapping and accurate object detection algorithm. The simulation was ended as the pursuer reached the evader. The environment mapping generated a minimal layout, which was obtained through teleoperating over *move_base* package. The object detection algorithm implemented was YOLO, which triggered the motion of the robot based on a PID controller if a human of detected as an object as shown in **Fig. 3**.

The *launch file* scripted was capable to launch the world in the gazebo environment. A few other python scripts were developed to control the movement of the bot in the environment. In this world, robot, human and other dynamic and static elements spawned that were also executed through the launch file. Once the environment was operational, the

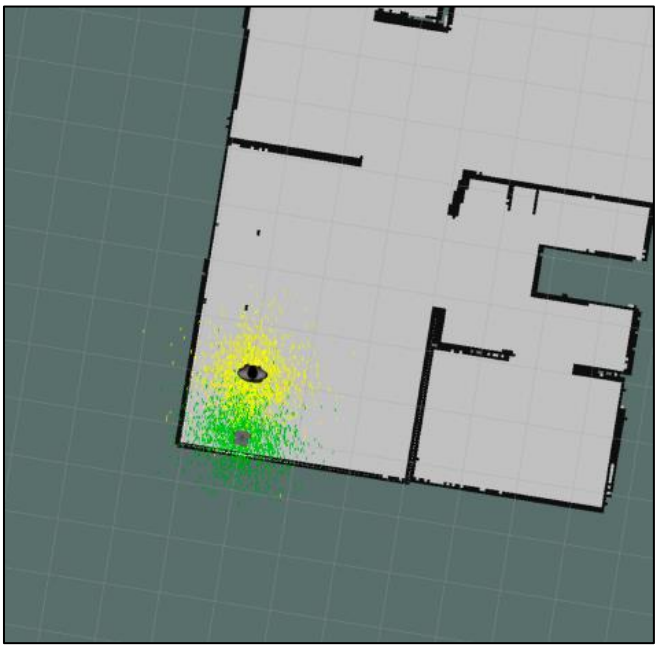

Figure 3 : Human detection through TurtleBot3

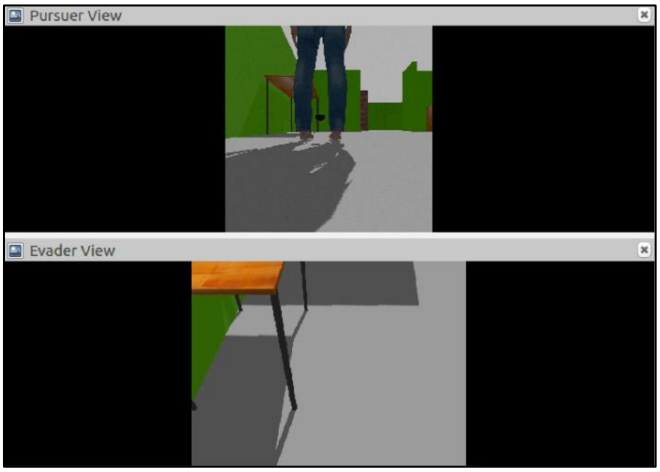

Figure 4 : First Person Views from TurtleBot3 and human

teleoperation of the bot was controlled through the terminal. Execution and communication between other nodes were made possible through the terminal window in Ubuntu 16.04 with ROS Kinetic. The project, however, was executed successfully and the performance-based results obtained were convincing.

IV. CONTRIBUTIONS

We all contributed equally to the execution of the project. My teammates are Karan Shah and Gautam Sharma. The course involved numerous other projects that served as the basis to develop a platform to undertake the current pursuit-evasion game project.

My contribution to the project were as follows:

• I developed several iterations of the Gazebo worlds (buildings) traversed by the TurtleBot3. An optimal selection of the environment was made from the same.

• I collaborated on scripting the *launch file* that allowed loading of the Gazebo environment, and spawning of the robot, human and other elements.

• I was involved with mounting the camera sensor onto the TurtleBot3 robot. RViz was setup and subscriber/publisher protocol were implemented to publish the stream of images and made available a visual feed for First Person View of the bot and the human as shown in **Fig 4**.

• I studied and implemented the controller to move the bot as per the location coordinates available to the bot. The path traced by the robot was according to the error term between center of the evader and the center of bounding box. This way, the bot would move linearly towards the evader.

• I was involved with Karan to partially contribute in implementation of YOLO darknet for object detection. I also drafted the project report.

Gautam Sharma majorly scripted the python scripts to move the bot accordingly and help create and setup the package to execute the *launch file.* Karan Shah was majorly involved in implementation of YOLO for object detection and executing SLAM for mapping the environment.

V. SKILLS AND KNOWLEDGE ACQUIRED

The project provided with an in-depth exploration of perception and controls' concepts.

• I learnt and exploited the ROS framework to develop a middleware between the simulated entities. This could be transferred easily to a real-world bot.

• I learnt to setup the ROS environment along with necessary packages and dependencies and how to use the same as per the use-case. This included the *gmapping package*, *move_base package*, etc.

• I was made familiar with mounting a simulated sensor onto the bot and how to stream the data to specific nodes and allow to use the same and maneuver around the objects without collision.

• The tuning of controller would result into optimal movement trajectory of the bot. I was able to correlate the perception and control layers of the robotics pipeline.

• I was introduced with key perception concepts and algorithms per se SLAM and YOLO. Learnt how to exploit neural networks in optimizing the object detection, object recognition task.

• I learnt how the script and develop the *launch file* and other python scripts to control the bot in a simulated world.

VI. REFERENCES

[1] "Robot Operating System (ROS)" [Online].Available: https://www.ros.org/

[2] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., … Leonard, J. J. (2016). Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age. *IEEE Transactions on Robotics*, *32*(6), 1309–1332. https://doi.org/10.1109/TRO.2016.2624754

[3] Robotis – TurtleBot3 : https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[4] Kam, H., Lee, S.-H., Taejung Park, & Kim, C.-H. (2015, October). RViz: a toolkit for real domain data visualization. Retrieved April 14, 2021, from ResearchGate website: https://www.researchgate.net/publication/275524447_RViz_a_toolkit_for_real_domain_data_visualization

[5] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR).
https://doi.org/10.1109/cvpr.2016.91