Executive Summary: VIP Lunabotics Software Team (Spring 2023)

The Objective

The objective of the VIP Lunabotics software team was to develop a FastSLAM 1.0 algorithm to localize the robot within the lunar environment and build a map of the surroundings. To achieve this, we require data collection from multiple sensors. It was also necessary to design a basic simulator/environment to test if our algorithm works properly.

The objective of developing a senior design project for Gazebo robot simulation is to create a realistic and accurate simulation environment that can be used for testing and development of robot control algorithms, navigation systems, and other robotic applications. The simulation will provide a platform to test and refine software before deploying it onto a physical robot. This will help in reducing the cost and time associated with physical testing and provide a safer and controlled environment for testing.

A Note on Continuance from Last Semester (Fall 2022):

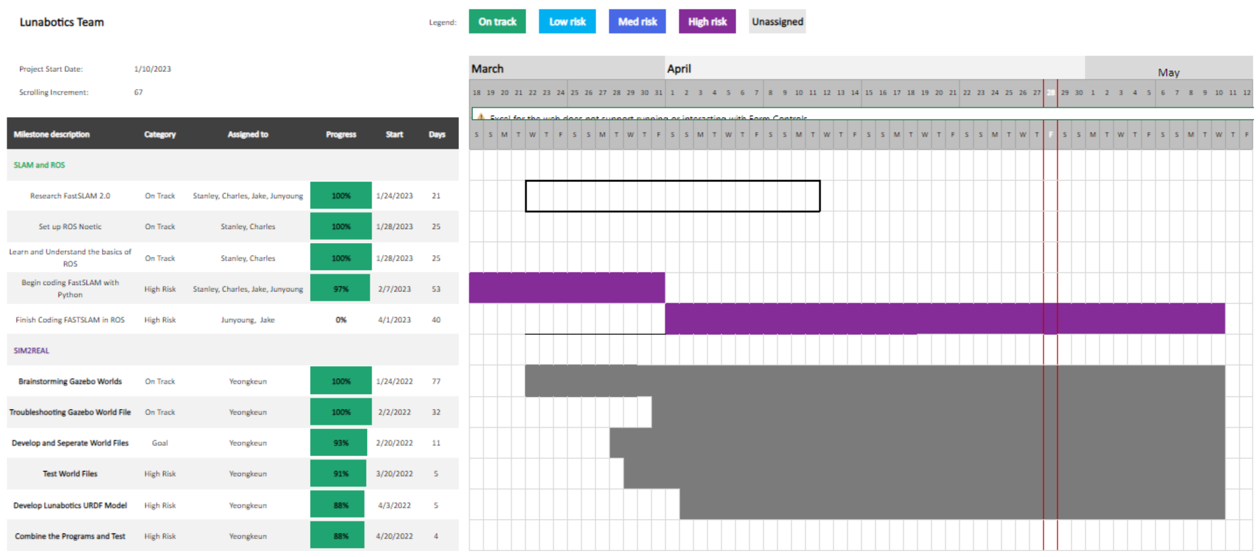
> The objective of the SLAM software subteam last semester (Fall 2022) was relatively the same as this semester. However, due to the complicated nature of SLAM, the objective morphed into focusing on researching and building a proper foundation of understanding for how SLAM algorithms work. This task took the entire semester to complete and bled into this semester (Spring 2023) due to the onboarding of new team members and the switch from pure EKF SLAM to FastSLAM 1.0. Thus, the primary objective this semester was to finish researching the topic and code the algorithm in Python.
>
> The objective of my senior project last semester (Fall 2021) was relatively similar to that of this semester, but my focus was primarily on researching the Gazebo platform to effectively simulate the physical components of the robot in the real world. Through my research and simple coding implementations during the last semester, I was able to successfully complete the sim2real project.

The projects we worked on this semester are as follows:

Project	Contributors
FastSLAM 1.0 Algorithm	Junyoung Kim, Jake Stall, Stanley Kim, Charles Allison
Senior Design: Sim2Real	Yeongkeun Kwon

This is the link to the FastSLAM 1.0 GitHub:
https://github.com/jun0kim3722/RLunabotics

This is our final updated Gantt chart for this semester:



Lunabotics Team

Legend: On track | Low risk | Med risk | High risk | Unassigned

Project Start Date: 1/10/2023

Scrolling Increment: 67

Milestone description	Category	Assigned to	Progress	Start	Days
SLAM and ROS					
Research FastSLAM 2.0	On Track	Stanley, Charles, Jake, Junyoung	100%	1/24/2023	21
Set up ROS Noetic	On Track	Stanley, Charles	100%	1/28/2023	25
Learn and Understand the basics of ROS	On Track	Stanley, Charles	100%	1/28/2023	25
Begin coding FastSLAM with Python	High Risk	Stanley, Charles, Jake, Junyoung	97%	2/7/2023	53
Finish Coding FASTSLAM in ROS	High Risk	Junyoung, Jake	0%	4/1/2023	40
SIM2REAL					
Brainstorming Gazebo Worlds	On Track	Yeongkeun	100%	1/24/2022	77
Troubleshooting Gazebo World File	On Track	Yeongkeun	100%	2/2/2022	32
Develop and Seperate World Files	Goal	Yeongkeun	93%	2/20/2022	11
Test World Files	High Risk	Yeongkeun	91%	3/20/2022	5
Develop Lunabotics URDF Model	High Risk	Yeongkeun	88%	4/3/2022	5
Combine the Programs and Test	High Risk	Yeongkeun	88%	4/20/2022	4

LINK:
https://purdue0-my.sharepoint.com/:x:/r/personal/kim3722_purdue_edu/_layouts/15/doc2.aspx?sourcedoc=%7BB98F68A7-F6EA-4C1D-AE52-9A2591D9AF59%7D&file=team_gantt_chart.xlsx&action=default&mobileredirect=true&DefaultItemOpen=1&cid=4deb08fa-9645-4bb0-918b-eaca4ac2032e

Project: FastSLAM 1.0 Algorithm

Research

FastSLAM is an algorithm for simultaneous localization and mapping (SLAM) tasks. SLAM is a complex problem that requires a robot to create a map of its surroundings while also estimating its own position within the map. It tackles this problem by using a particle filter to represent the probability distribution over the robot's pose and a map of the environment.

The particle filter is a probabilistic algorithm that uses a set of particles to represent possible states of the system. It is typically used to estimate the robots' positions and the location of landmarks in the environment, and uses a set of particles to represent the possible robot and landmark positions and updates their weights based on the sensor measurements and motion models.

Before programming the FastSLAM algorithm, we needed a good understanding of the FastSLAM algorithm so that we are able to implement the algorithm. The topics that we have researched includes the Monte Carlo Localization (MCL), resampling particles, and gmapping.

gmapping:

Gmapping is an existing and established FastSLAM 1.0 algorithm that was used to help understand the foundations of FastSLAM.

The gmapping, in summary, is a probabilistic algorithm used for creating a map of an unknown environmental dn estimates the robot's location within that environment using data from the sensor data. The five steps are as follows:

1. Initializes the occupancy grid map, particle filter, and sensor models
2. Receives the sensor data from the robot's sensors
3. Updates the particle filter using the sensor data
4. Updates the occupancy grid map using the particle filter, and estimates the robot's pose
5. Publishes the map and pose estimate for use by other components of the robot system

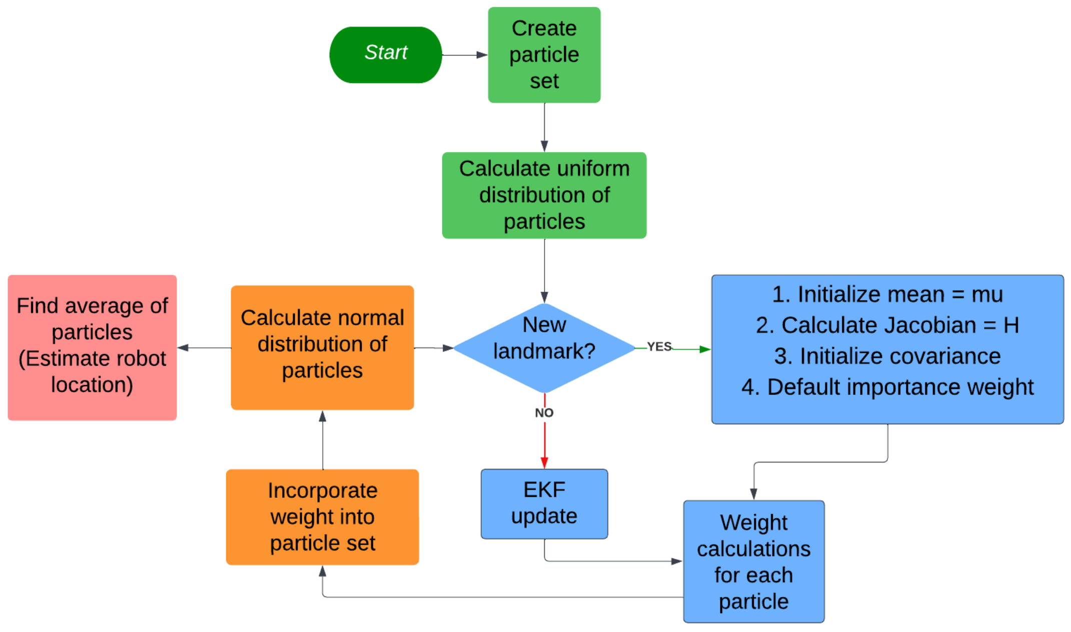
Monte Carlo Localization (MCL):

The Monte Carlo Locationaliztion (MCL) estimates the pose of the mobile robot in an unknown environment using sensor measurements. It also uses the particle filter approach to represent the robot's possible poses with set of particles, and updates the particle weights based on sensor measurements.

FastSLAM and the Particle Filter

For the project, we created a Python simulation of FastSLAM to test its performance on a variety of scenarios. The simulation includes a robot equipped with sensors and a map of the environment. The robot is tasked with navigating the environment while simultaneously building a map and localizing itself within that map. Our simulation includes several different scenarios, such as navigating a maze or following a path through a complex environment. We also varied the number of particles used in the particle filter to examine how this affects the performance of the algorithm.

Below is a flowchart that visualizes the FastSLAM 1.0 algorithm that we created:


Start
Create particle set
Calculate uniform distribution of particles
New landmark?
YES
NO
1. Initialize mean = mu
2. Calculate Jacobian = H
3. Initialize covariance
4. Default importance weight
EKF update
Weight calculations for each particle
Incorporate weight into particle set
Calculate normal distribution of particles
Find average of particles (Estimate robot location)

Resampling Particles:

The resampling of particles is a step in a particle filter algorithm that selects a new set of particles from the previous set. The purpose is to focus the computation on the most likely particles and discard the less likely ones in order to improve the accuracy of the particle filter algorithm. The two most common techniques for resampling particles are the roulette wheel and stochastic universal sampling. The Roulette Wheel divides up the wheel into slices proportional to the weights of the particles. It uses the binary search method to compute the most likely particle, which means that the particles with higher weights have a higher probability of being selected. Stochastic universal sampling, on the other hand, is a variation of roulette wheel selection that uses a sampling process to select individuals from the population. In this method, a set of evenly spaced pointers are created on the roulette wheel, and particles are selected by iterating through the wheel and selecting individuals whose slice intersects with the pointers. For the Python simulation, we have decided to select the stochastic universal sampling, as it is easier to implement.

Particle Filter

The particle filter is a probabilistic algorithm that uses a set of particles to represent possible states of the system. It is typically used to estimate the robots' positions and the location of landmarks in the environment, and uses a set of particles to represent the possible robot and landmark positions and updates their weights based on the sensor measurements and motion models.

EKF:

Since Particle Filter is for Robot’s motion, we are using EKF for the landmark's location. EKF(Extended Kalman Filter) uses Jacobian to predict non-linear curves and find out the uncertainty of the prediction. By using this EKF, we are able to calculate position of landmarks and uncertainty everytime robot observes the landmark. At the EKF update part, EKF will recalculate landmark position and uncertainty to have a more accurate map and weight calculation for particle filter.

Conclusion

Through our simulation, we found that FastSLAM is a highly effective algorithm for SLAM tasks. It was able to accurately localize the robot within simulation and create an accurate map of its environment. Additionally, we found that increasing the number of particles in the particle filter improved the accuracy of the algorithm.

Project : Senior Design: Gazebo Simulation, Sim2Real

Research

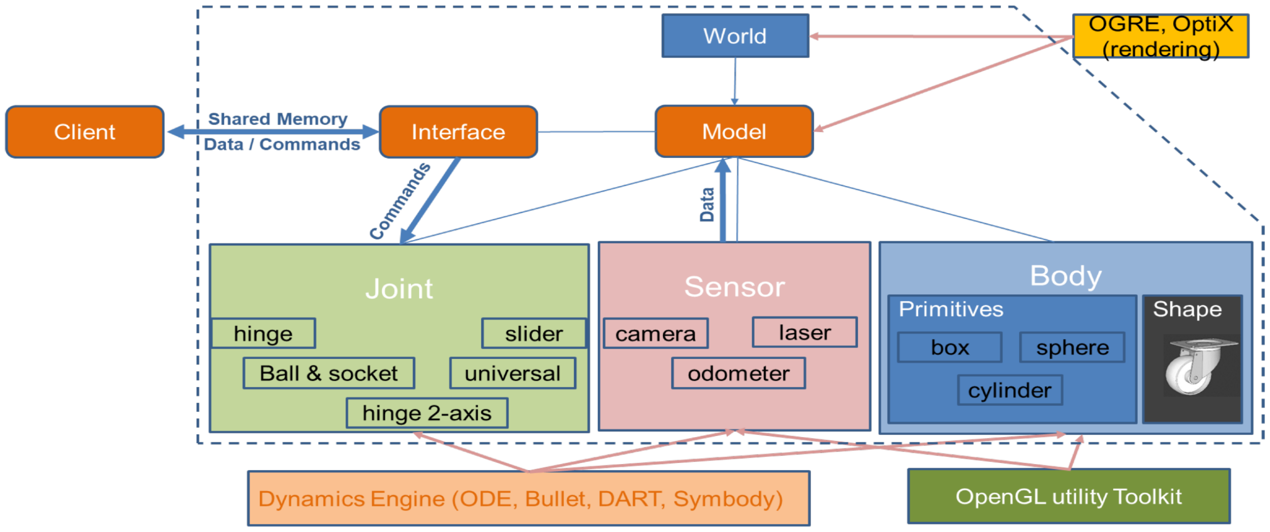
Gazebo is a robot simulator that provides a 3D environment to test robots and their sensors in a virtual environment before deploying them in the real world. It is an open-source project developed by the Open Source Robotics Foundation (OSRF) and is widely used in the robotics community for research, development, and testing purposes.

One of the popular robots simulated in Gazebo is the TurtleBot3. It is a mobile robot platform designed for education, research, and development. The TurtleBot3 is equipped with various sensors such as LIDAR, cameras, and IMU, which can be used for localization, mapping, and navigation tasks.

To simulate the Lunabot in Gazebo, one needs to have the robot's 3D model, control and sensor plugins, and a launch file that starts the simulation. The control and sensor plugins allow the robot to interact with the simulated environment and provide data to the robot's software. The launch file starts the simulation and loads the robot's software.

Gazebo also provides a plugin for ROS (Robot Operating System), which allows the user to control the simulated robot and access its sensor data. This integration with ROS makes it easier for developers to test their robot's software in a simulated environment and then deploy it on the real robot.

In summary, Gazebo is a powerful robot simulator that enables developers to test and validate their robot's software in a safe and controlled environment. The integration with ROS and availability of pre-built robot models like TurtleBot3 make it a popular choice among the robotics community, and the Lunabot simulator is composed of several file formats that work together to create a realistic simulation environment. The following are the main components and the flow chart.



URDF (Unified Robot Description Format)

For the simulator, I created a URDF file for the Lunabot to describe the robot’s specifications. Unified Robot Description Format (URDF) is an XML file format used in robotics to describe the structure and kinematics of a robot. The purpose of URDF is to provide a common language for robots so that different software packages can communicate with each other seamlessly. It specifies the robot’s physical dimensions, mass properties, joints, links, and sensors.

In our URDF file, the Lunabot is described as a tree structure consisting of a root link and a series of child links connected by joints. Each link can have its own visual and collision properties, and joints define the relationship between links, such as fixed or revolute joints. The properties of each link and joint can be defined using different attributes, such as size, shape, and position.

URDF also allows for the specification of sensors that are mounted on the robot, such as cameras, lidars, and force sensors. This information is critical for robot simulation and control, as it allows software packages to accurately model and simulate the robot's behavior in different environments.

URDF is a key component for the robot simulator, as it enables easy and standardized sharing of robot descriptions among different software tools, facilitating the development and deployment of robot applications, and I built the file in xml format.

SDF (Simulation Description Format)

SDF stands for Simulation Description Format and is an XML-based file format used to describe models for use in robot simulations. It is commonly used in conjunction with the Gazebo simulator, which uses SDF files to create virtual environments and simulate robot behavior.

Like URDF, SDF allows the user to specify the physical properties and visual appearance of a robot model, as well as its collision properties and joints. However, SDF also includes additional features such as support for multiple physics engines, custom sensors, and environment settings.

In the Lunabot simulator, the SDF files are used to define the physical properties of the robot model and its environment, which are then used by the Gazebo simulator to simulate the robot's movement and interaction with the environment. The SDF files for the Lunabot include a description of the robot's wheels, sensors, cameras, and other components.

Gazebo Model File

The Gazebo Model File is a critical component of the robot simulator like Turtlebot3, as it defines the robot's physical properties and behavior in the simulation environment.

In our robot simulator, Gazebo Model File is used to define the visual and physical properties of a robot model. It is written in SDF format and contains information about the robot's geometry, inertial properties, and visual appearance.

In the Lunabot simulator, the Gazebo Model File defines the geometry and appearance of the robot's various components, including the chassis, wheels, sensors, and other attachments. It specifies the physical properties of these components, such as their mass, moments of inertia, and friction coefficients, which are important for simulating the robot's movements accurately.

Launch File

In the Lunabot simulator, the launch file is used to launch the simulation environment with all the necessary components and settings. It is written in XML format and consists of several tags and parameters that define the nodes and their interactions. The launch file is used to start the Gazebo simulator with the robot model, sensors, controllers, and other components.
The launch file in the Lunabot3 simulator is typically followd named from the turtlebot3 simulator liie "turtlebot3_{model_name}.launch” because the Turtlebot3 fram was applied to the Lunabot development.

The launch file is divided into several sections, each defining a different component of the simulation. The first section defines the Gazebo simulation parameters, such as the world file to be used, the physics engine to be used, and the time step size. The next section defines the robot model, including the URDF file, the joint states publisher, and the robot state publisher. The third section defines the sensor models to be used in the simulation, such as the lidar and camera sensors. The final section defines the controllers to be used, such as the joint position controllers.

The launch file also includes parameters for the simulation, such as the robot model name, the initial position and orientation of the robot, and the simulation time limit. These parameters can be modified to customize the simulation for different scenarios.

Conclusion

All of these file formats work together to create a complete simulation environment for the Lunabot simulator. The URDF file defines the robot model and its properties, the SDF file defines the simulation environment, the Gazebo model file defines the physical properties of each object, and the launch file is used to launch the simulation with the desired settings. By using these file formats together, developers can create and test their code in a realistic simulation environment before deploying it on a real robot, and the Lunabot simulator was developed successfully.

Individual Teammate Contributions

Jun Kim

- Researched on FASTSLAM, EKF, and Particle filter.
- Wrote a basic frame of particle filter.
- Wrote EKF, resampling code.
- Wrote a python simulation to test our FastSLAM code.

Jake Stall

- Researched FastSLAM, EKF, the particle filter, and gmapping
- Investigated and found ways to implement our custom FastSLAM algorithm with a ROS gazebo turtlebot simulation
- Created the landmark class and implemented it into the FastSLAM algorithm
- Helped debug landmark update section of the FastSLAM algorithm

Charles Allison

- Researched SLAM and FastSLAM, along with EKF and particle filter.
- Researched the basics of ROS and got a workspace set up.
- Wrote the initial EKF update step code.
- Helped debug landmark update and EKF update code.

Stanley Kim

- Researched and documented the FastSLAM, Resampling Particles, Gmapping, MCL
- Assisted in debugging the landmark updates and the EKF update code
- Implemented the calculation normal distribution of particles
- Implemented the Jacobian calculations and the weight calculations for each particle

Yeongken Kwon

- Everything about the Sim2Real project
- Research about Gazebo, ROS, and Turtlebot3
- Utilize the Turtlebot3 frame to enhance the robot simulator performances.
- Develop testing environments
- Develop the Lunabot simulator