EECE 5550 Mobile Robotics - Fall 2019
Project Definition

Deadlines

- Team formation: Friday October 18th, 7pm, by email
- Project proposals: Monday October 28th, 23:59pm, by Blackboard.
- Project progress: deadline TBA
- Project final due: Friday December 13th, 23:59pm.

Proposal

The project proposal should be 2-page document including the following:

- The project goals
- An abstract of the project you want to do
- Detail of steps that need to be taken to achieve the goals:
    - A timeline should be included for how long each step can take
- A brief literature overview and some references about your project.

One important factor in writing proposals is the doability factor; how achievable your goals are. I suggest evaluating your capabilities realistically and set your goals slightly higher. This can help you be realistic while at the same time pushing yourself to learn more.

The member 1 of the team (link to teams here) is responsible for uploading the proposals in the Blackboard. If you want to change the member 1 please edit the link above.

As explained in the class, there exists a basic project definition which comprises only up to 80% of the total project grade. To achieve the rest 20%, you have to go farther and implement an algorithm from scratch or design a complex system. Your proposal does not need to necessarily include the basic project if it is complex and involving enough.

If you want to use a robot platform rather than Turtlebots, your proposals should reflect that. Focus on capabilities of the robot you want to use and indicate why you need those capabilities to achieve your goals.

Project Deliverables

- Progress report presentation
- Final presentation
- Detailed technical document
- A final video highlighting your achievements

Standard Project: Autonomous Robot Navigation (up to 80% of your project grade)

Introduction

In this project, you will program a mobile robot so that it can navigate in an unknown environment autonomously while mapping its environment. You will focus on high-level mapping and navigation tasks for the robot to create a world model of the environment and then travel through it.

Objectives

Upon successful completion of this project, you will be able to:

1. program a mobile robot for waypoint navigation.
2. program a mobile robot for mapping its environment.
3. program a mobile robot for obstacle avoidance.
4. program a mobile robot for path planning in a (dynamic) world map.

Requirements

1. Robot must operate fully autonomously within the experimental environment setup.
2. Robot must effectively avoid obstacles and walls within its environment.
3. Robot must generate a local map of its environment.
4. Robot must generate a global map of its workspace.
5. Robot must be capable of planning an admissible trajectory to move within its workspace.
6. Robot must be capable of estimating its position and orientation in world coordinates.
7. Robot must be capable of navigating to a given waypoint. The waypoint will be considered reached when any part of the robot covers the waypoint marker. The waypoints will remain stationary within robot's workspace and will be marked by a circle with diameter equal to the Turtlebot's diameter. Robot must navigate at least 4 out of 5 waypoints.
8. Robot must return to the starting point (base) by planning a path that will cover all the open areas in the map generated. The robot must update the world map while returning to the base. Note that the location of the obstacles on the return trip may change and the robot must adjust the map and plan accordingly.

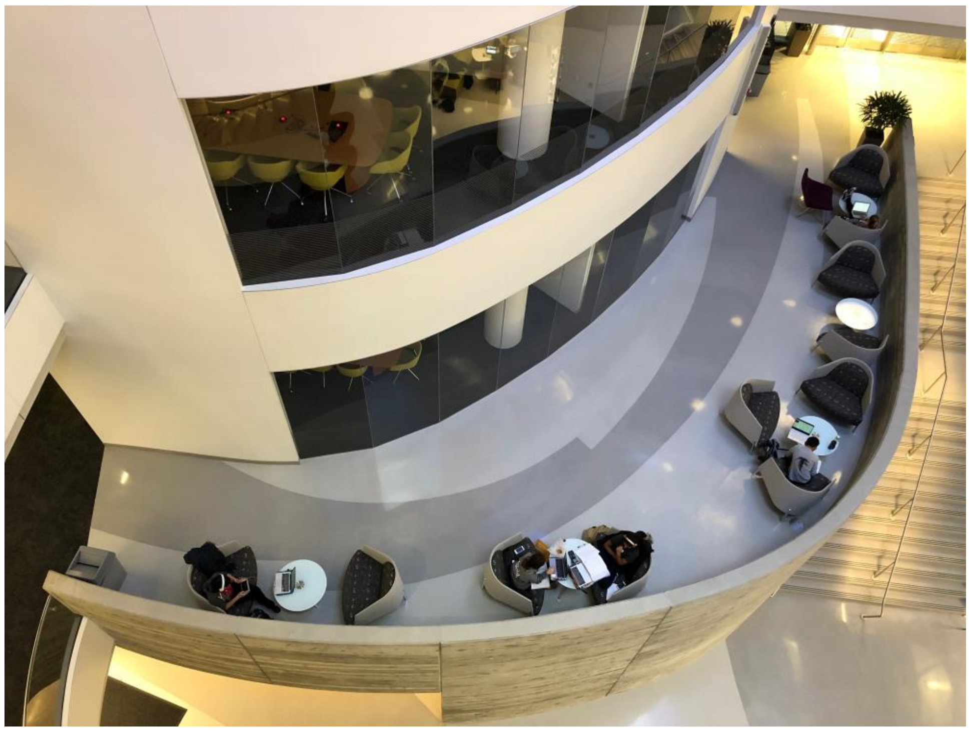
Figure 1 - An example world for autonomous robot navigation

A non-inclusive list of suggestions for proposals

- Implement an algorithm from scratch for robot localization, path planning, etc. Examples:
    - Path planning algorithms like A*, D*, RRT, trajopt (some not taught in the class)
    - Collaborative mapping (by multiple robots)
    - Visual odometry
    - 3D mesh reconstruction
    - …
- Implement a working complex system. Example:
    - A robot that gets an image of a location from a cell phone in an environment, go search and find the location.

- (partial bonus) Using AWS RoboMaker for your projects
- (partial bonus) Using docker for development

Issues on Robots

For reporting robot problems please file issues at the following link:

- https://github.com/sharif1093/eece5550_master/issues

The title of the issue should have as title "[NAME OF THE ROBOT] A BRIEF INFORMATIVE TITLE". Teams are responsible for reporting and troubleshooting the issues (as this is part of a robotic career), but our support team may also help.

Development Workflow

Access to the physical robot is limited. The robots can be accessed through checkouts, i.e. marking a calendar for the time you need the robot (usually no longer than ~4 hours periods; the process to be announced later in detail). The access to the physical robot might be different based on the platform you are using as well (Turtlebot, ROSbot, HSR). So instead of trying all pieces of your code on the physical robot from the beginning, the following more convenient way is suggested: simply create a working simulation environment, develop your codes in simulation, then try the final code on the physical robot. This is a workflow used by roboticists all over the world. Fortunately, for all our robot platforms this simulation environment already exists, although you might need to add your own objects to the environment. Simulations in ROS are usually done in Gazebo, which is a physics modeling environment to model dynamics, collisions, sensors, etc. The modeling of robots is done through Universal Robotic Description Format (URDF) XML description. To make modeling of robots more modular there also exist a macro language named XACRO which helps make the URDF files more modular. As an example of XACRO and URDF file see the "turtlebot_description" package:

https://github.com/turtlebot/turtlebot/tree/kinetic/turtlebot_description/urdf

The modularity that ROS provides through using "topics" makes transferring between simulation and reality very easy, only the interface should be kept the same between the real robot and the simulation. For our current platforms this uniform interface between simulation and real robot is already implemented.

There exists a tool named RVIZ in ROS, which is a very useful tool for robot visualization. This tool should not be confused with Gazebo, as it does not provide any physics modeling whatsoever (kinematics, dynamics, collision, etc.). It is JUST a visualization tool which can be both used with simulation or the physical robot.

References

- Some basic useful instructions for turtlebots:
  - https://github.com/sharif1093/eece5550_master/wiki
- ROS navigation:
  - http://wiki.ros.org/navigation
  - http://wiki.ros.org/navigation/Tutorials
  - http://wiki.ros.org/gmapping
- Modeling robots:
  - http://wiki.ros.org/urdf/Tutorials
  - http://wiki.ros.org/xacro
  - http://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File
- Visualization tools:
  - http://wiki.ros.org/rviz
- April tags: https://github.com/RIVeR-Lab/apriltag_ii_ros
- Visual odometry:
  - https://github.com/MIT-SPARK/Kimera-VIO
- Trajectory planning:
  - https://github.com/ros-industrial-consortium/trajopt_ros