2.12: Introduction to Robotics
Final Contest: Docking Simulation*

Spring 2021

Instructions:

1. **This handout describes a software tool that you may use to test out your docking algorithm. There is nothing that you have to submit for this.**

1 Introduction

Part of the challenge for the final project will be for you to autonomously or semi-autonomously to navigate the mobile robot in the designated environment. The provided simulation tool uses ROS and Gazebo allow you to test out your navigation algorithms in a virtual environment. This handout will:

1. Walk you through setting up the tool.
2. Explain a bit about how the tool is designed.
3. Explain how the simulated environment is designed.
4. Explain how you can interact with the simulation.
5. Explain how you can modify the simulation.

You will complete run everything in this handout in your virtual machine (VM).

2 Setup

2.1 Download Files

Download all of the files that you will need for the simulation. In the next step you will place these files at different locations within your VM's file-structure.

```
git clone https://github.com/mit212/testNavigation.git
```

*

1. Version 1 - 2021: Phillip Daniel

2.2 Re-Locate Files

Once you have downloaded this repo:

- Place 'start_gazebo_testStage.desktop' on your desktop
- Place the directory 'building_editor_models' in /home
- Place 'turtlebot3_testStage.launch' in opt/ros/melodic/share/turtlebot3_gazebo/launch
  - You can get to the 'opt' directory by going to 'Files', and then clicking 'Other Locations > Computer > opt.'
- Place 'turtlebot3_testStage.world' in opt/ros/melodic/share/turtlebot3_gazebo/worlds
- Place 'start-gazebo-testStage.sh' in /home
- Place the directory 'simple_controller' in /home/user/catkin_ws
- Place the directory 'turtlebot3_description' in /home/user/catkin_ws
- Open a terminal, navigate to the catkin_ws folder, and run **catkin_make**
- You should now be able to run the tool:
  - Click on 'start_gazebo_testStage.desktop' on your desktop. This will open the simulated environment.
    - You may need to right-click on this shortcut, go to "Permissions", and change it to trusted.
  - Open a new terminal and run the below to update your path:

```
source /opt/ros/melodic/setup.bash
. ~/catkin\_ws/devel/setup.bash
```

  - Now run the controller template:

```
rosrun simple\_controller controller.py
```

3 Tool Structure

This tool has many different file-types. I will briefly explain some of them.

3.1 Loading the Simulated Environment and Starting ROS Communication

The file 'start_gazebo_testStage.desktop' is effectively a short-cut to the file 'start-gazebo-testStage.sh'. 'start-gazebo-testStage.sh' runs a few '.launch' files that load the stage and mobile robot into Gazebo, and starts publishing and subscribing via ROS so that you can later interact with the simulation.

'turtlebot3_testStage.launch' loads the robot using '.xacro' files. This is an XML macro file. This tells Gazebo what the robot's geometry is, and also loads a plugin that allows ROS to move the robot as if it was omni-directional. The simulated room's designed is stored in 'turtlebot3_testStage.world'.

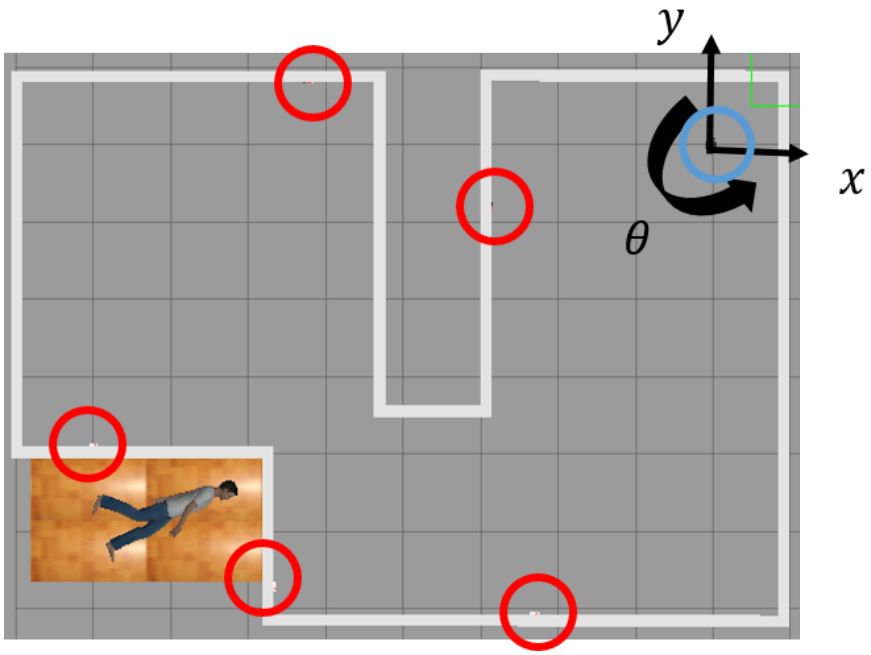

Figure 1: This is a top-down view of the simulated environment. The April tags are circled in red, and the mobile robot is circled in blue. The mobile robot in the simulated environment is the "Turtlebot: Burger" (source: `https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/`

3.2 Making a Controller That Interacts With the Simulation

The controller portion of this tool is described in the file 'controller.py'. This file receives information from Gazebo about the robot's state, does some computation based on this information, and then sends commands to Gazebo to move the robot in some way.

This controller is based on the tutorial shown here: `https://www.theconstructsim.com/ros-qa-053-how-to-move-a-robot-to-a-certain-point-using-twist/`. Feel free to review it if you would like more information about how to interact with the model.

This tutorial assumes some initial knowledge with ROS. The first six brief tutorials here are extremely helpful (`http://wiki.ros.org/ROS/Tutorials`), and should be enough background for you to understand how the controller code was put together.

The controller is written in Python. If you need to look things up to understand the code, we suggest that you reference the website: `https://www.w3schools.com/python`.

4 Design of the Simulated Environment

Here is a description of the simulated environment.

4.1 The Room

The room is shown in figure 1. The April tags in the simulated environment are visualizations that match the hard-coded location of the tags in the 'controller.py' file. The robot is not able to drive through the walls of the room. The global coordinate system is also shown in the figure.

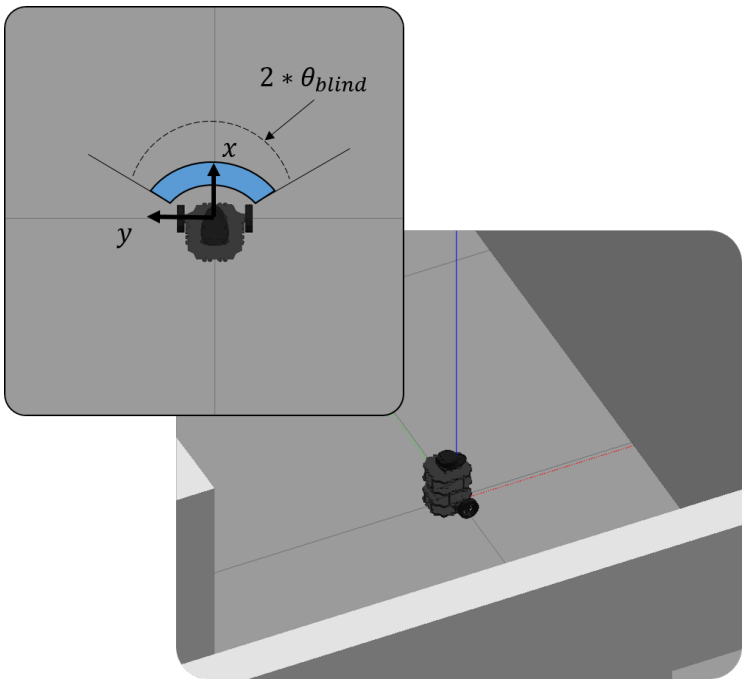

Figure 2: This is a top-down and perspective view of the mobile robot. The figure also shows the robot's blind spot, as well as its body fixed coordinate frame.

The mobile robot is shown in figure 2. The figure also shows the robot's blind spot, as well as its body fixed coordinate frame.

The 'controller.py' file determines when the robot can see a tag based on the location of the robot and the direction that it is pointing, see figure 3.

5 Writing Your Own Controller Code

We have provided 'controller.py' as a template for you to use in writing your navigation algorithm. You can place your code after the line'while not rospy.is_shutdown()' in this file, as this is the start of the main loop. The code is commented, so you are invited to read through the code to understand what is going on.

5.1 Robot Pose

You will be given a map of the contest stage. This map will describe the shape of the room, as well as the final location of each of the April tags.

For the actual mobile robot, you should be able to measure its orientation with respect to the global coordinate system ($theta$) as well as its relative position from each of the April tags that it identifies (e.g. $tagInfo.relPos1.x, tagInfo.relPos1.y$). Thus, these form a set robot pose measurements that you can use in formulating your control algorithm.

5.2 Robot Control

You send velocity commands to the robot by writing a number between -1 and 1 to the variables 'xDot', 'yDot', and 'thetaDot'. The 'setSpeeds' function scales this range to the maximum velocity

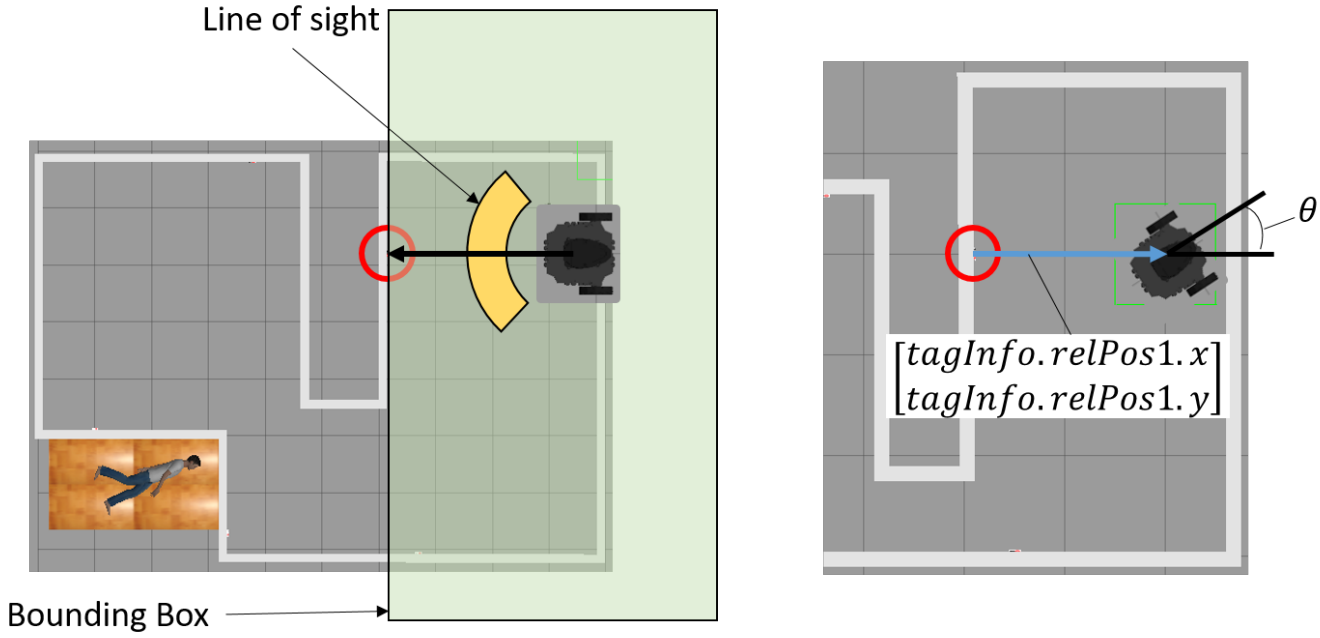

Figure 3: (Left) This shows how 'controller.py' determines what April tags are visible. The robot needs to be inside of a tag's bounding box, and the tag needs to be in the robot's line of sight in order for the tag to be "seen". The bounding box of each tag is defined by the variables 'tag1Vis.xMin,' etc. (Right) This shows a set of robot pose measurements that you should be able to compute on the hardware in real time.

of the robot for you. These values are sent to the robot with the command $pub.publish(speed)$.

6 Edit the Gazebo world

If you want to edit the room, you can do so by navigating to the directory 'opt/ros/melodic/share/-turtlebot3_gazebo/worlds' and then running the command:

```
gazebo turtlebot3_testStage.world
```

Once you are finished making changes, use 'save as' to replace the old '.world' file. This tutorial describes how you can make some simple changes to the world: `http://gazebosim.org/tutorials?tut=model_editor`