**Enabling Scripting Workflows for eCornell Autonomous Mobile Robots**

**Maps:** In order for the simulator to be a useful tool, the robot must be able to detect its environment (the map) and give sensor readings that reflect the presence/absence of various environment features.

Original Design:

* Environment is represented by a map file generated using MapMakerGUI
* Loading the map in the map in the simulator does three things
  + Parses the map files
  + Creates data structures describing the locations of maps, lines, beacons, and virtual walls
    - Passes these to the CreateRobot object (the object stores the map data)
  + Plots the maps using the data structures created above
* When a sensor method from the robot is called (genLidar, genBump, etc), the robot accesses its internal map data structures and processes them to return accurate sensor reading with the environment accounted for

Problem:

* The robot needs the map data structures to give accurate sensor readings, but these data structures are only created by the “Load Map” functionality of the simulator app
* This is why you often need students to open the simulator, load the map, and then generate a sensor reading

Refactor:

* Robot currently has properties representing itself and the environment, remove properties representing the environment and the getMap and setMap methods for getting and setting these properties (make the robot just a robot)
* Created a static CreateRobot method map = loadMap(mapFileName) that parses a map file and returns a struct containing the walls, lines, beacs, and vwalls informatin
* Adjusted the sensor methods (genLidar, genBump, findDist, findCollisions) to take in the map struct as an input argument, updated function comments for help usage
* Adjusted the push\_load\_map callback of the simulator to use the static loadMap method
  + Store the map struct as properties of the app
  + Uses the map struct to plot the map
* Implemented fast and responsive click workaround for “Set Position” functionality in app

The above represents the heaviest lift for refactoring the simulator and CreateRobot class to enable scripting workflows. This has the benefit of also enabling MATLAB Grader usage for at least some sandbox activities.

Our decision to separate the map data structures from the CreateRobot class follows from MathWorks’s approach to this problem in Simulink. The scene and the robot are separate but can exchange data as necessary for motion/sensing/rendering. By passing the map data structures to the sensor methods when they are called, we can replicate the physical robot operation where it does not know what the environment looks like until it senses.

**Remaining Work for eCornell to complete:**

There are a few remaining tasks to fully divorce the CreateRobot class from the SimulatorGUI app. MathWorks has not completed these since we believe they fairly straightforward and we are not sure whether this functionality is needed in the eCornell course.

1. Move the updateSensorVisualization method from the CreateRobot class to the app
   1. Function header should be updateSensorVisualization(app)
2. Refactor the ButtonSensorsRoomba CreateRobot method
   1. Add robot properties represented the state of the “Play” and “advance” buttons on the robot
   2. Add simulator callbacks to the toggle buttons to update the state values of the robot when these buttons are pressed.
3. Refactor the setLEDsRoomba CreateRobot method
   1. Add robot properties for the on/off state of the buttons, the color (pColor) and the intensity (pInt)
      1. setLEDsRoomba just sets these values
   2. Add code to the updateSensorVisualization method to read the above values from the robot object and update the appearance of the text\_play, text\_advance, and text\_power GUI elements
4. Update the CreateRobot\_refactor constructor method to remove setting the handlesGUI property and accepting the app as an input argument.

Right now, everything is in a working state, Simulator\_refactor references CreateRobot\_refactor and you can load maps and manually drive the robot in the simulator. You can also instantiate the robot object, load maps, and generate sensor readings outside of the simulator using Live Scripts. Whether you proceed with the 4 tasks listed above is up to you to decide.

**Findings:**

Redundant methods:

* VirtualWallSensorCreate (genVWall)
* ReadSonar, ReadSonarMultiple (genSonar)
* LidarSensorCreate (genLidar)
* ReadBeacon (genCamera)
* RealSenseDist (genRSDepth)
* RealSenseTag (genCamera)
* CameraSensorCreate (genCamera)
* OverheadLocalizationCreate (genOverhead)

The methods in parentheses (gen\* methods) represents the core methods for performing the computations needed to simulate and return a sensor reading. The listed methods utilize these functions but simulate a communication delay (from a config file) between requesting the reading and getting the readings. If the communication delay is not important, then the listed methods can be removed.

Potential Bugs:

* In updateSensorVisualization: realSenseDepth x\_sensor and y\_sensor are both computed using cameraDisplaceX and don’t use cameraDisplaceY
  + genRSDepth does the same thing
  + genCamera does the same thing
* I don’t know if this is correct or not, but wanted to bring it to your attention since in my mind this did not fit the expected pattern and the cameraDisplaceY property of the robot is never referenced in the CreateRobot class

AutoCheck:

* There are a number of methods in the CreateRobot class that begin by calling autoCheck(). This function checks whether autonomous control has been enabled (i.e. that the method is being called from a control function running in the simulator). If you want these methods to be usable outside of the simulator, these methods need to be refactored to remove this check.
  + Method names: ctrl+F autoCheck, appears to be roughly 35 methods, many likely aren’t used in the eCornell course