# Using the refactored simulator

The primary change made to the simulator and the CreateRobot class was to refactor how maps were handled to enable scripting workflows for exercises that involved obtaining sensor readings but did not involve autonomous control functions. The rationale for this is given in the document “SimulatorRefactor\_work” that I shared with you along with the refactored simulator in March.

# Scripting Workflows for Obtaining Sensor Readings

To better understand what the refactored simulator and CreateRobot class enable, let’s look at the plot\_LIDAR\_range exercise. The goal of this exercise is for learners to obtain and plot the raw LIDAR data from the robot in the presence of obstacles like walls. In the original version of the exercise learners had to complete the following steps before they could obtain and plot the LIDAR data:

1. Open the Simulator
2. Click the “Load Map” button to load a map.
3. Click the “Set Position” button to set the robot’s pose.
4. Click the “Read Sensors” button to export the robot object from the simulator’s scope (the simulator’s workspace in MATLAB parlance) to the live script’s scope (the base workspace in MATLAB parlance).

Once the learner has done this, they can then proceed to the task at hand, which is to use the genLidar function to obtain the LIDAR reading, and then plot it using the plot function.

Let’s now look at plot\_LIDAR\_range\_refactor, which is the same exercise, but can now be completed without ever opening the simulator! The workflow now is all script based, so all setup steps that a learner previously needed to do themselves can now be given to them, and learners can just jump right to obtaining and plotting the LIDAR data. The first section of the live script completes all the setup steps by:

1. Instantiating a CreateRobot object
2. Loading a map
3. Setting the robot pose (with some random noise, can also be sliders)
4. Plotting the map with the robot pose so that learners can see where the robot is on the map

The learner needs to just run this section, and then they can proceed to the next section the get the LIDAR reading and plot it. The primary change now is that when the call any of the sensor functions (gen\* functions like genLidar), they should supply the map struct as an argument. For example, the old way was:

distLidar = genLidar(robotObj) or distLidar = robotObj.genLidar()

and the new way is:

distLidar = genLidar(robotObj,map) or distLidar = robotObj.genLidar(map)

**Note:** the two syntaxes shown are equivalent in MATLAB. The first syntax (function form) is our recommended syntax as many of MATLAB users are not familiar with OOP and the idea of an object method. The second syntax (object methods) conforms more the expectations of uses familiar with OOP from working in other languages.

# Change to Writing Autonomous Control Functions

Changes to the autonomous control functions are minimal. In general, changes mostly are in response to the fact that the map representation has been separated from the robot. To illustrate this, let’s look at backupBump\_refactor.mlx. The old function signature was

function backupBump(robotObj)

and the new function signature is

function backupBump(robotObj,map)

to reflect that the function will need to know what the environment looks like (the map) to sense it properly. In tandem with this, any attempts to collect sensor data will need to pass the map struct, e.g. on line 22

bump = genBump(robotObj,map); OR bump = robotObj.genBump(map);