Lab Guide

Environment Interpretation

Background

The objective of the environment interpretation lab guide is to teach students about how an occupancy grid mapping algorithm and LiDAR inverse measurement model can be implemented. Prior to starting this lab guide, please review the following concept reviews:

- Concept Review – Occupancy Grid Mapping
- Concept Review – LiDAR Inverse Measurement Models
- Concept Review – Frames of Reference

Considerations for this lab guide:

This lab uses the SDCS roadmap presented in the vehicle control skills activity for generating trajectories for the car to follow when exploring the environment. How this works can be found by reviewing the lab guide for the vehicle control skills activity.

This lab will ask you to complete 2 pipelines as shown below:

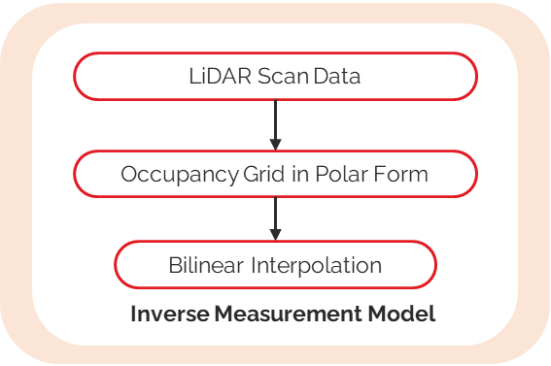

Figure 1: Inverse Measurement Model Pipeline

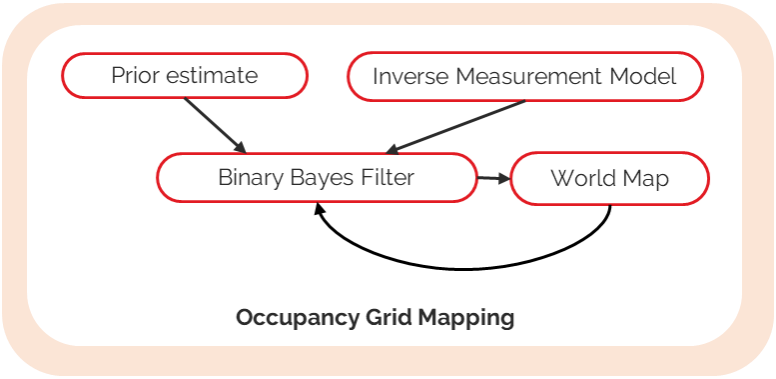

Figure 2: Occupancy Grid Mapping Pipeline

- **Inverse Measurement Model** is designed to give students insights into how LiDAR data can be interpreted to make predictions about the surrounding environment.
- **Occupancy Grid Mapping** shows students how successive LiDAR scans can be integrated over time to construct a complete map of the environment.

Skills Activity - Setup

The physical setup for the lab will require the SDCS roadmap, Review Setup Guide – Instructor for information on the size of the roadmap. The default structure of this map will rely on the large map configuration.

environment_interpretation.py Is the main file that needs to be edited and executed for interacting with this skills activity. Experiments can be configured by modifying the following parameters, located near the top of **environment_interpretation.py**

```
# ================ Experiment Configuration ================
# ===== Timing Parameters
# - tf: experiment duration in seconds.
# - startDelay: delay to give filters time to settle in seconds.
# - controllerUpdateRate: control update rate in Hz. Shouldn't exceed 500
tf = 100
startDelay = 1
controllerUpdateRate = 100

# ===== Vehicle Controller Parameters
# - enableVehicleControl: If true, the QCar will drive through the specified
#   node sequence. If false, the QCar will remain stationary.
# - v_ref: desired velocity in m/s
# - nodeSequence: list of nodes from roadmap. Used for trajectory generation.
enableVehicleControl = False
v_ref = 0.3
nodeSequence = [0, 20, 0]

# ===== Occupancy Grid Parameters
# - cellWidth: edge length for occupancy grid cells (in meters)
# - r_res: range resolution for polar grid cells (in meters)
# - r_max: maximum range of the lidar (in meters)
# - p_low: likelihood value for vacant cells (according to lidar scan)
# - p_high: likelihood value for occupied cells (according to lidar scan)
cellWidth = 0.02
r_res = 0.05
r_max = 4
p_low = 0.4
p_high = 0.6
```

Skills Activity 1 – LiDAR Inverse Measurement Model

The objective of this section is to focus on the inverse measurement model for interpreting LiDAR scan data in the context of an occupancy grid map. Using the concept reviews:

- Concept Review – Frames of Reference
- Concept Review – LiDAR Inverse Measurement Models

Students develop an intuition on how an efficient LiDAR inverse measurement model can be designed and implemented.

Common setup –

Within the **environment_interpretation.py** file set the **enableVehicleControl** parameter to **False** to prevent the QCar from driving.

Physical setup –

Run **environment_interpretation.py** in a terminal on the QCar directly to start the state estimation skills activity.

Virtual setup -

Inside Quanser Interactive Labs navigate to Citcyscape Lite workspace prior to starting the skills activity. In a terminal session, navigate to the folder environment interpretation skills activity.

Step 1 – Developing a Polar Occupancy grid in the Lidar Frame

Inside **environment_interpretation.py** the **OccupancyGrid()** class has been created and partially implemented. **SECTION A** defines the unique sections required for generating the polar grid used by the occupancy grid class. Step 1 will require you implement the **update_polar_grid()** function. The generated polar grid should be stored in the variable called **self.polarPatch**.

Note: When implementing your solution, consider the OccupancyGrid **__init__** and **init_polar_grid** functions to ensure your solution is consistent with the rest of the class.

```
        # ==============  SECTION A - Polar Grid ====================
    def init_polar_grid(self, r_max, r_res):
        # Configuration Parameters for polar grid
        fov = 2*np.pi
        self.phiRes = 1 * np.pi/180
        self.r_max = r_max
        self.r_res = r_res

        # Size of polar patch
        self.mPolarPatch = np.int_(np.ceil(fov / self.phiRes))
        self.nPolarPatch = np.int_(np.floor(self.r_max/self.r_res))
```

```
        self.polarPatch = np.zeros(
            shape = (self.mPolarPatch, self.nPolarPatch),
            dtype = np.float32
        )

    def update_polar_grid(self, r):
        # Implement code here to populate the values of self.polarPatch
        # given LiDAR range data 'r'.
        # - r is a 1D list of length self.mPolarPatch
        # - All range measurements are equally self.phiRes radians apart,
        #   starting with 0

        # Implement Your Solution Here

        pass
```

Codeblock A – Polar Grid Implementation

Results:

When executing, a scope window will open. The upper-left scope axis should resemble something like what is shown in Figure 3.

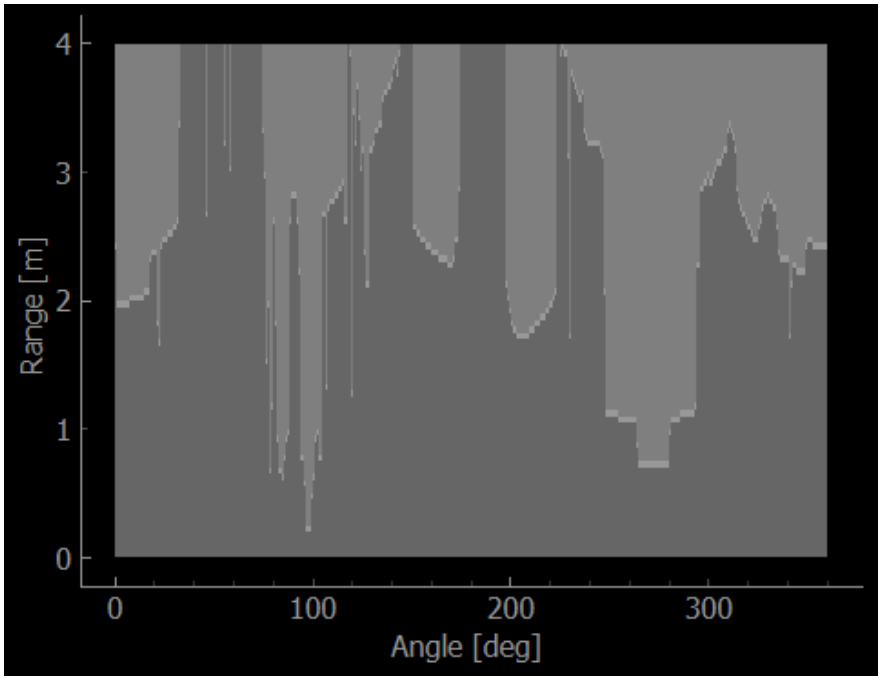

Figure 3: Occupancy grid from a single LiDAR scan in polar form

Note: your results will likely look different but should reflect the environment where your QCar is located. You can easily test if it's working correctly by putting your hand in front of the LiDAR at various positions.

Step 2 – Bilinear interpolation

In this step, we will focus on **SECTION B** where you will need to complete the implementation of the **generate_patch** function. This function will make use of the variable **self.polarPatch** calculated in step 1 and maps it to a cartesian grid using bilinear interpolation. When the bilinear interpolation is complete use the variable **self.patch** to generate a square cartesian grid with the LiDAR located at its centre.

```
# ==============  SECTION B - Interpolation ====================
def init_patch(self):
    self.nPatch = np.int_(2*np.ceil(self.r_max/self.cellWidth) + 1)
    self.patch = np.zeros(
        shape = (self.nPatch, self.nPatch),
        dtype = np.float32
    )

def generate_patch(self, th):
    # Implement Your Solution Here
    pass
```

Codeblock B – Bilinear Patch Generation

Results

When executing, a scope window will open. Use Figure 4 as a reference for what the bottom-left scope axis should look like

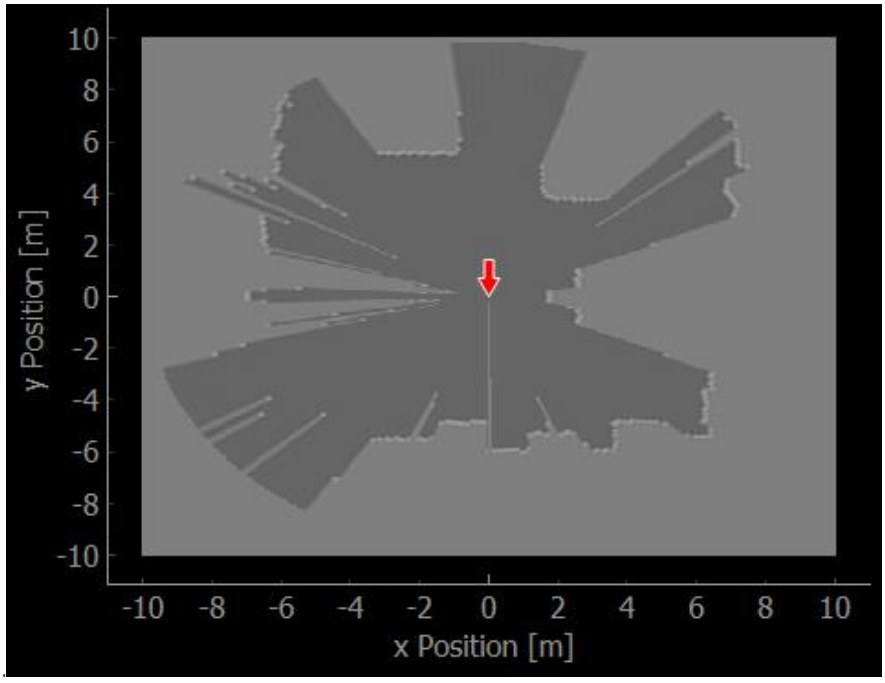

Figure 4: Occupancy grid from a single LiDAR scan in cartesian form

Note: your results will likely look different but should reflect the environment where your QCar is located. You can easily test if it's working correctly by putting your hand in front of the LiDAR at various positions.

Skills Activity 2 – Occupancy Grid Mapping

The objective of this section is to build a complete map of the environment over time as the QCar drives around. This section will make use of the following concept reviews:

- Concept Review – Frames of Reference
- Concept Review – Occupancy Grid Mapping

Students will develop an intuition on how occupancy grids can be used as a means of interpreting the environment around a vehicle and how LiDAR scan data can be integrated over time.

Step 1 – Occupancy Grid Map Update

This skills activity will focus on **SECTION C** where you will need to complete the implementation for the **update_map** function. You will make use of:

- **self.patch** representing a single LiDAR scan developed in Sills Activity 1
- **self.map** variable representing the global occupancy gird**.**

To apply a binary bayes filter to update the map of the environment as each new lidar scan is registered.

```
    # ==============  SECTION C - Occupancy Grid Update  ====================
    def init_world_map(self,
            x_min = -4,
            x_max = 3,
            y_min = -3,
            y_max = 6,
            cellWidth=0.02
        ):

        self.x_min = x_min
        self.x_max = x_max
        self.y_min = y_min
        self.y_max = y_max
        self.cellWidth = cellWidth
        self.xLength = x_max - x_min
        self.yLength = y_max - y_min
        self.m = np.int_(np.ceil(self.yLength/self.cellWidth))
        self.n = np.int_(np.ceil(self.xLength/self.cellWidth))

        self.map = np.full(
            shape = (self.m, self.n),
            fill_value = self.l_prior,
```

```
            dtype = np.float32
        )

    def xy_to_ij(self, x, y):
        i = np.int_(np.round( (self.y_max - y) / self.cellWidth ))
        j = np.int_(np.round( (x - self.x_min) / self.cellWidth ))
        return i, j

    def update_map(self, x, y, th, angles, distances):
        self.update_polar_grid(distances)
        self.generate_patch(th)

        # Implement code here to update self.map using self.patch
        pass
```

Codeblock C – Global Occupancy Grid Map Generation

Hint: you may find the **find_overlap** function in **pal.utilities.math** helpful for identifying which specific cells of **self.map** need to be updated during each iteration.

Common setup –

For this activity, set the **enableVehicleControl** parameter to **True** to enable the QCar to drive around the environment, following the node sequence specified by the **nodeSequence** parameter.

Physical setup -

Run **environment_interpretation.py** in a terminal on the QCar directly to start the environment interpretation skills activity. The car will move during this phase so ensure the environment is clear and safe for the car to drive around.

Virtual setup -

Inside Quanser Interactive Labs navigate to the Cityscape Lite workspace prior to starting the skills activity. In a terminal session, navigate to the folder containing the environment interpretation skills activity. Run **environment_interpretation.py** to start the skills activity.

Results for Occupancy Grid Mapping:

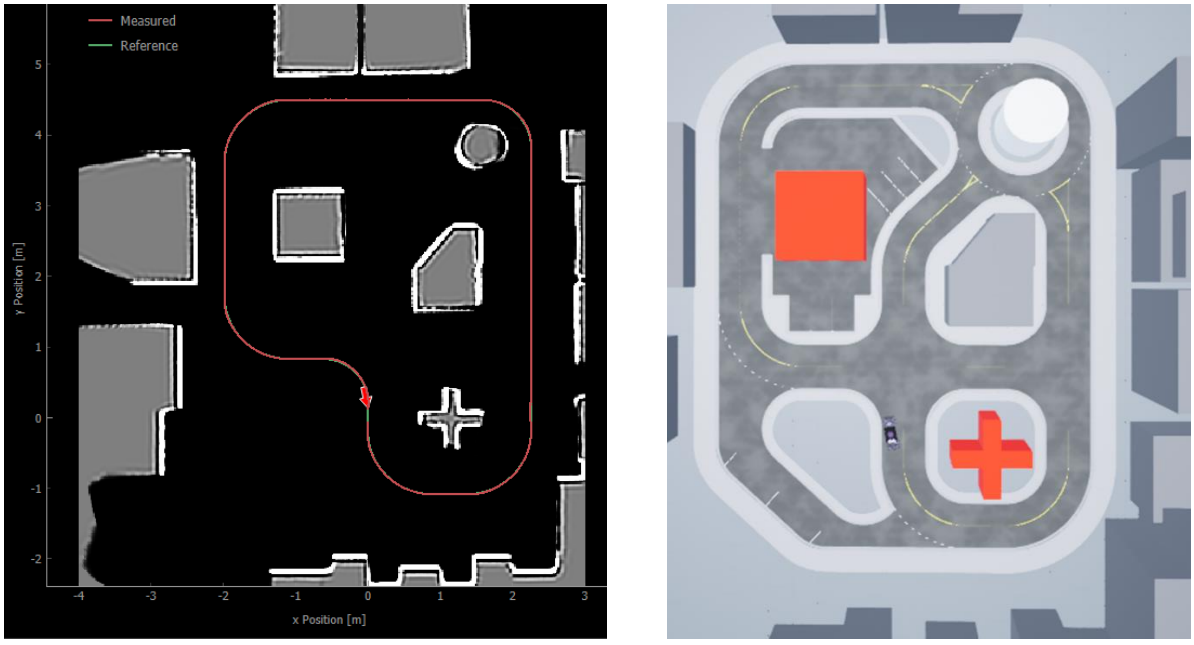

Figure 5: Generated occupancy grid map (left); Environment where mapping was performed (right)