JEM 455
MATLAB Programming Assignment 1:
Lidar Mapper

Due: 2/8/2024 @ 11:59pm

Save all MATLAB files necessary to complete this project into a single folder. Compress that folder into a zip file and submit to Moodle by the due date. All necessary code files to run your script file should be included in the folder. In other words, I should be able to unzip the folder, open your script in MATLAB, and run your code without any errors. Any errors will result in points counted off. I advise you to zip it, send it to another computer, and run it to make sure all your files are present. You do not need to include data files (.mat files) that I give you in your submission.

Background

A differentially wheeled robot is programmed to navigate through any arbitrary room with any arbitrary number of obstacles located in the room. While the robot navigates, it collects data concerning its current pose (w.r.t some global coordinate frame), and lidar data detecting the location of objects and walls with respect to the lidar equipment mounted at the robot's center. It is your job to process that data and construct a total map layout of the room in which the robot has traversed.

Specifics

Write a collection of MATLAB scripts and functions that can do the following.

- Load in the data collected by the robot
- Map the lidar data from raw collection to local point locations to the robot
- Transform the local point locations to global point locations
- Overlay the global point locations on an image to construct a room map

Data

All data that you must process are in .mat files. They are named "Run_xx.mat" in which the "xx" represents the identifier number for that run. These files contain 2 variables; `lid_data` and `poseTimeLine`.

`lid_data`

The variable `lid_data` is a $361 \times T$ matrix that contains the lidar data collected during the run, where T is the number of time stamps collected. The first row of this matrix is the time stamp (in seconds) of when that column of data was collected since the beginning of the run. The remaining 360 rows contain the distance (in meters) an object was detected at each of the 360 degrees surrounding the robot (row 2 is objects directly in front, row 92 is objects directly to the left, row 182 is objects directly behind, and row 272 is objects directly to the right). If a location did not detect an object (eg. Objects were beyond the range of the lidar), it will contain the "Not a Number" value NaN.

`poseTimeLine`

The variable `poseTimeLine` is a $3 \times T$ matrix that contains the location and orientation data of the robot with respect to the global coordinate frame (which is simply just where the robot was at the beginning of the run). Row 1 contains the x location (in cm), row 2 contains the y location (in cm), and row 3 contains the angle (in radians) of the robot. The following figure shows the coordinate frame to use for describing the pose of the robot. Assume that the origin of the robot's reference frame is centered on the lidar.

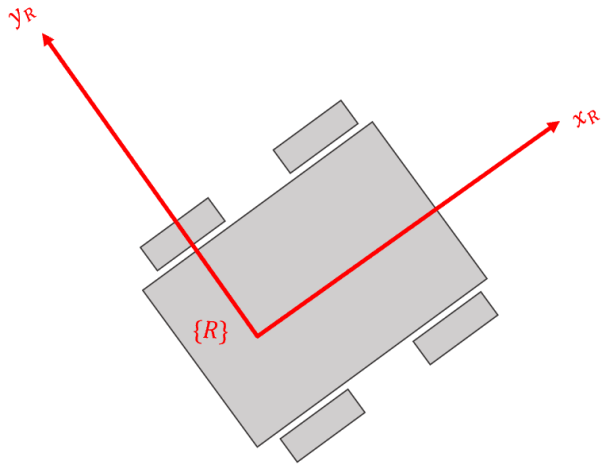


Figure 1: Local coordinate frame of the robot.

Functions

You must write the following MATLAB functions.

```
function [x, y] = lidar2location(lidarSnap)
```

This function must do the following.

- Process the lidar data from a single time point and transform that data into a collection of 2D points w.r.t the robot's local coordinate frame.
- Check to ensure that the input data is of the correct size and shape. If not, it returns blank vectors for the outputs (eg. `x=[]` and `y=[]`).
- Ignore all data points that contain the NaN such that the length of the output vectors are N which represents the number of data points out of the 360 that were not NaN.

Inputs:

- `lidarSnap` is a 360x1 vector that contains the lidar data from a single time stamp in m.

Outputs

- `x` is an $N \times 1$ vector that contains the local x location of the points detected by the lidar in cm.
- `y` is an $N \times 1$ vector that contains the local y location of the points detected by the lidar in cm.

`function [pc] = local2global(poseTimeLine, lid_data)`

This function must do the following.

- Process the pose data of the robot over time in conjunction with the lidar data over time to create a point cloud of all points detected over time by the robot with respect to the global coordinate frame.
- Ensure the pose data and the lidar data have the same amount of time stamps collected or it returns an empty pc variable as `pc=[]`.
- Use the lidar2location function defined above.

Inputs:

- `poseTimeLine` is a $3 \times T$ matrix that contains the pose information of the robot (see data section above).
- `lid_data` is a $361 \times T$ matrix that contains the raw lidar data (see data section above).

Outputs:

- `pc` is a $K \times 2$ matrix that contains the x (column 1) and y (column 2) locations, in cm, of all global points collected during an entire run.

`function [room_img] = createBlankRoom(pc, res)`

This function must do the following.

- Process the 2D point cloud created by the `local2global` function to create a blank image just big enough to fit all points of the room detected by the robot according to a pixel resolution. The blank image must be a $Y \times X$ matrix with all values equal to 1 (eg. white-painted pixel).
- Be able to account for the case in which the point cloud is empty.
- Be able to account for the case in which the desired resolution is 0 (eg. no image is created and a blank variable is returned).

Inputs:

- `pc` is a $K \times 2$ matrix that contains the point cloud created by the `local2global` function.
- `res` is a scalar value that contains the desired resolution of each pixel within the image to be created in cm/pixel.

Outputs:

- `room_img` is a blank $Y \times X \times 1$ image (a matrix of that size in which all values are equal to 1).

Note: When calculating the amount of pixels necessary to fit all detected points in the point cloud within the image, make sure that your extreme cases (largest and smallest x and y points) will exist on the very edge of the image and not outside of the image.

`function [paintedRoomImg] = paintObjects(blankRoomImg, pc, res)`

This function must do the following.

- Process the 2D point cloud created by the `local2global` function and paint the pixel corresponding with every point black (eg. a value of 0 in the matrix) according to the pixel resolution.
- Be able to account for a blank blankRoomImg variable and/or a blank pc variable.
- Be able to account for the case in which a point exists outside of pre-created blankRoomImg by returning and error.

Inputs:

- blankRoomImg is a $Y \times X \times 1$ image created by the createBlankRoom function.
- `pc` is a $K \times 2$ matrix that contains the point cloud created by the `local2global` function.
- `res` is a scalar value that contains the desired resolution of each pixel within the image to be created in cm/pixel.

Outputs:

- `paintedRoomImg` is a $Y \times X \times 1$ image that contains the pixels painted associated with the edges of objects. See the test cases below to ensure you are correctly painting these images.

Script

Write a script that can load in any .mat file that contains run data and create a room map and plots the room map. You will want to use the function `imshow` to do this with the image. Make sure that you have the "Image Processing and Computer Vision" toolbox in MATLAB to do this.

Grading Rubric

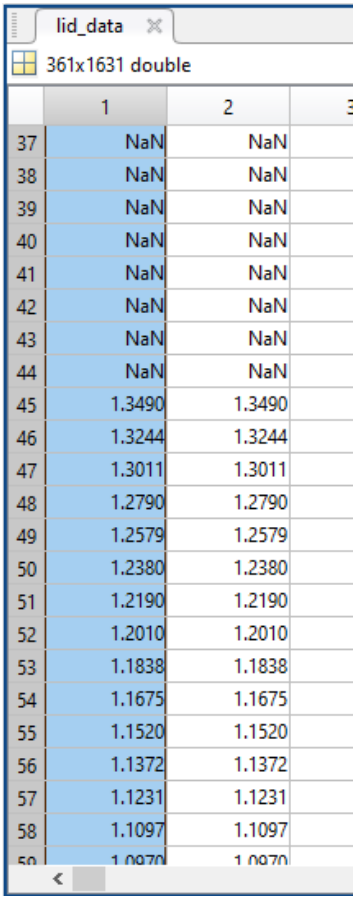
Script compiles without intervention	10 pts
lidar2location function	25 pts
Accurate calculation of local x and y points	*17 pts*
Input variables correct shape	*3 pts*
NaN's ignored	*5 pts*
local2global function	25 pts
Accurate calculation of global x and y points	*14 pts*
Detects different amount of time stamps in inputs	*3 pts*
Uses the lidar2location function	*3 pts*
Correct calculation of pose matrix	*5 pts*
createBlankRoom function	15 pts
Image created of correct size	*10 pts*
All values set to 1	*2 pts*
Returns empty if the pc if empty	*1 pt*
Account for a resolution of 0	*2 pts*
paintObjects	15 pts
Paints correct pixels	*10 pts*
Account for blank image and blank pc	*2 pts*

	Returning an error for pc value outside the image	*3 pts*
Script		10 pts
	Loads the .mat file	*2 pts*
	Creates the room map	*4 pts*
	Plots the room map	*4 pts*

Example

This will be an example run for a collection of data not given to you to show you the types of data you should receive and to help you better visualize the intent of this project. Not all figures that I show in this example are figures that you are asked for but may just be used to help you visualize the data better. Check the script description above to see what figures are asked for you to plot in your program.

After loading the data from the .mat file I ran local2global to receive the global points extracted from all lidar data. Within local2global, I called lidar2location in which the first timestamp has transformed the data in figure 2 into the data plotted in figure 3.



	1	2
37	NaN	NaN
38	NaN	NaN
39	NaN	NaN
40	NaN	NaN
41	NaN	NaN
42	NaN	NaN
43	NaN	NaN
44	NaN	NaN
45	1.3490	1.3490
46	1.3244	1.3244
47	1.3011	1.3011
48	1.2790	1.2790
49	1.2579	1.2579
50	1.2380	1.2380
51	1.2190	1.2190
52	1.2010	1.2010
53	1.1838	1.1838
54	1.1675	1.1675
55	1.1520	1.1520
56	1.1372	1.1372
57	1.1231	1.1231
58	1.1097	1.1097
59	1.0970	1.0970

Figure 2: Raw lidar data.

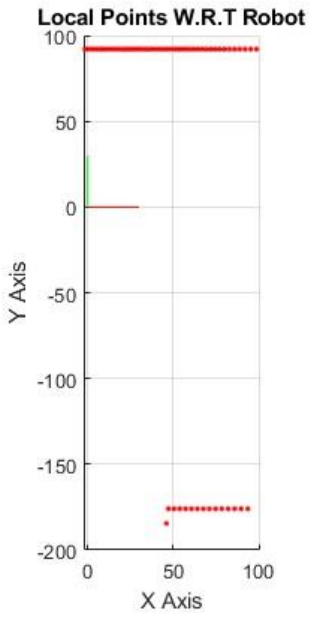


Figure 3: Local points with respect to the robot's local coordinate frame. Red and green axes represent the robot.

Each of these data was transformed to global coordinates where they were compounded for each time stamp. The following figure shows all points transformed to a single grid. Notice, I overlaid the x and y location of the robot over time to demonstrate its path.

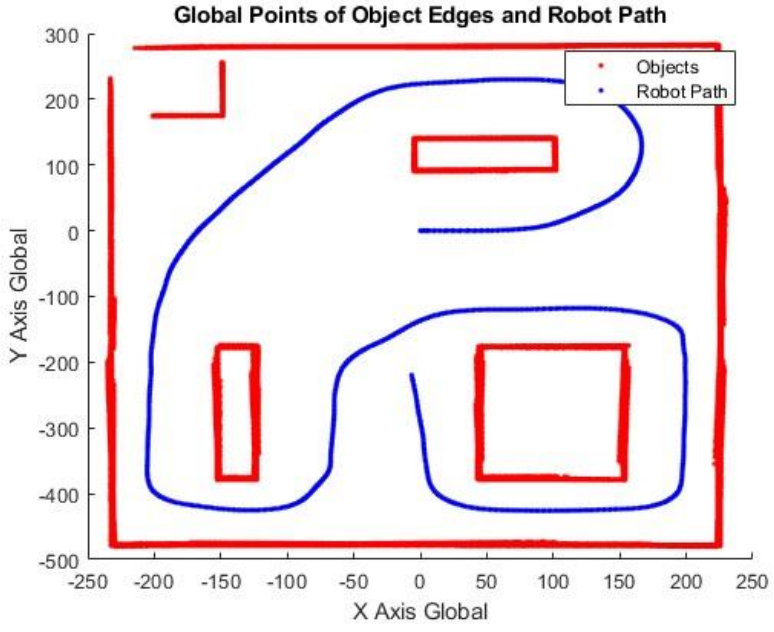


Figure 4: Global points of objects overlaid with the global position of the robot over time.

Lastly, I ran createBlankRoom and paintObjects with a resolution of 2 cm per pixel, plotted the image, and received figure 5.

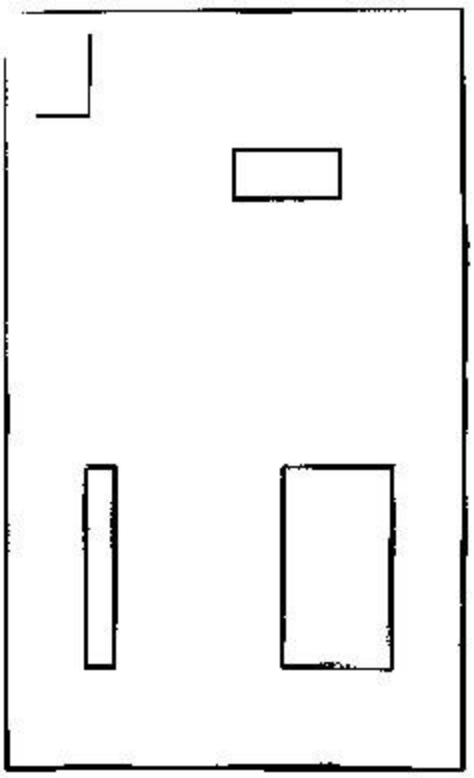
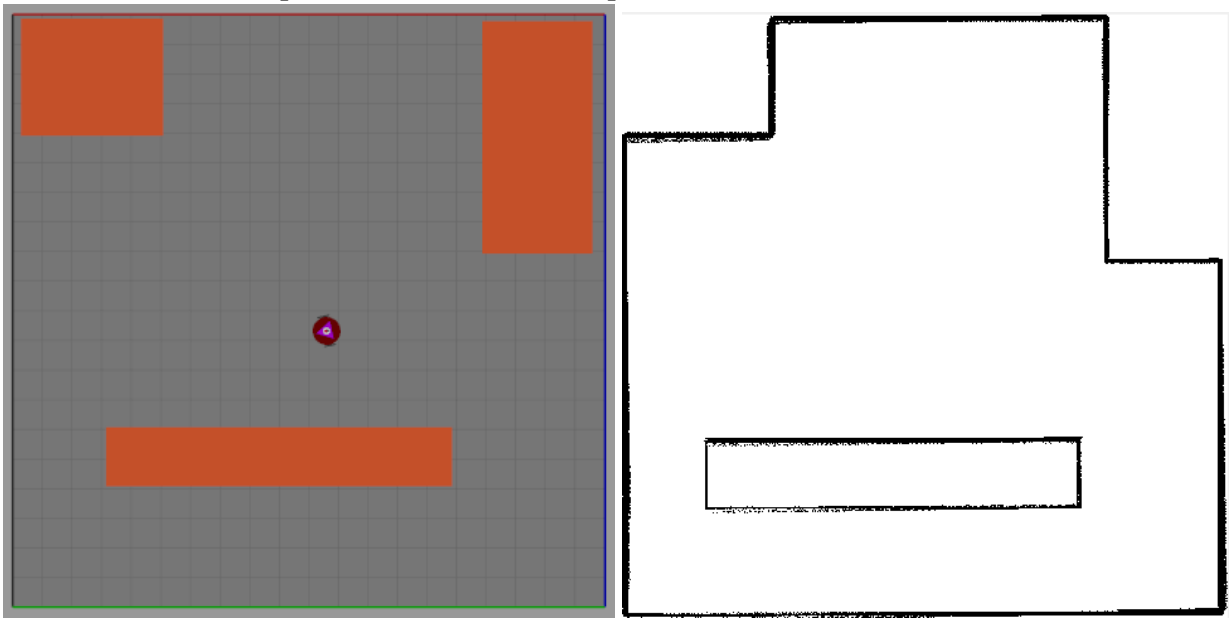


Figure 5: Room map of all objects and walls.

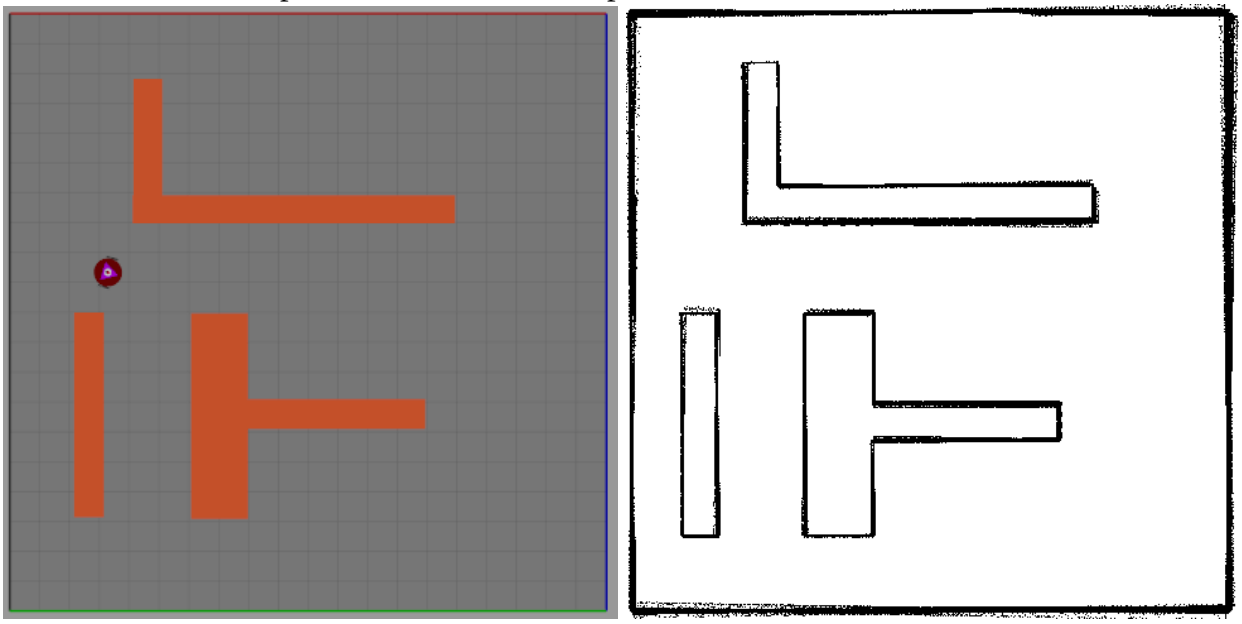
Test Cases

These data were collected in a virtual environment in which I had control to design the room, the obstacles in them and run the robot through the environment. Below is a list of the test runs and the results you should receive with a pixel resolution of 2 cm per pixel.

Run_01: Virtual map then created room map from lidar data



Run_02: Virtual map then created room map from lidar data



Run_03: Virtual map then created room map from lidar data. Note, this one suffers from drift issues but can still approximate the layout.