

# Navigation Workflow

ROB 102: Introduction to AI & Programming

Lab Session 4

2021/10/08

# Administrative

Project 2 is out, due **Monday, October 25<sup>th</sup>, at 11:59 PM.**

Watch the rest of the potential field navigation lecture before Monday.

**Monday's activity:** Potential field navigation on a robot.

There will be new team assignments for Project 2.

# Today...

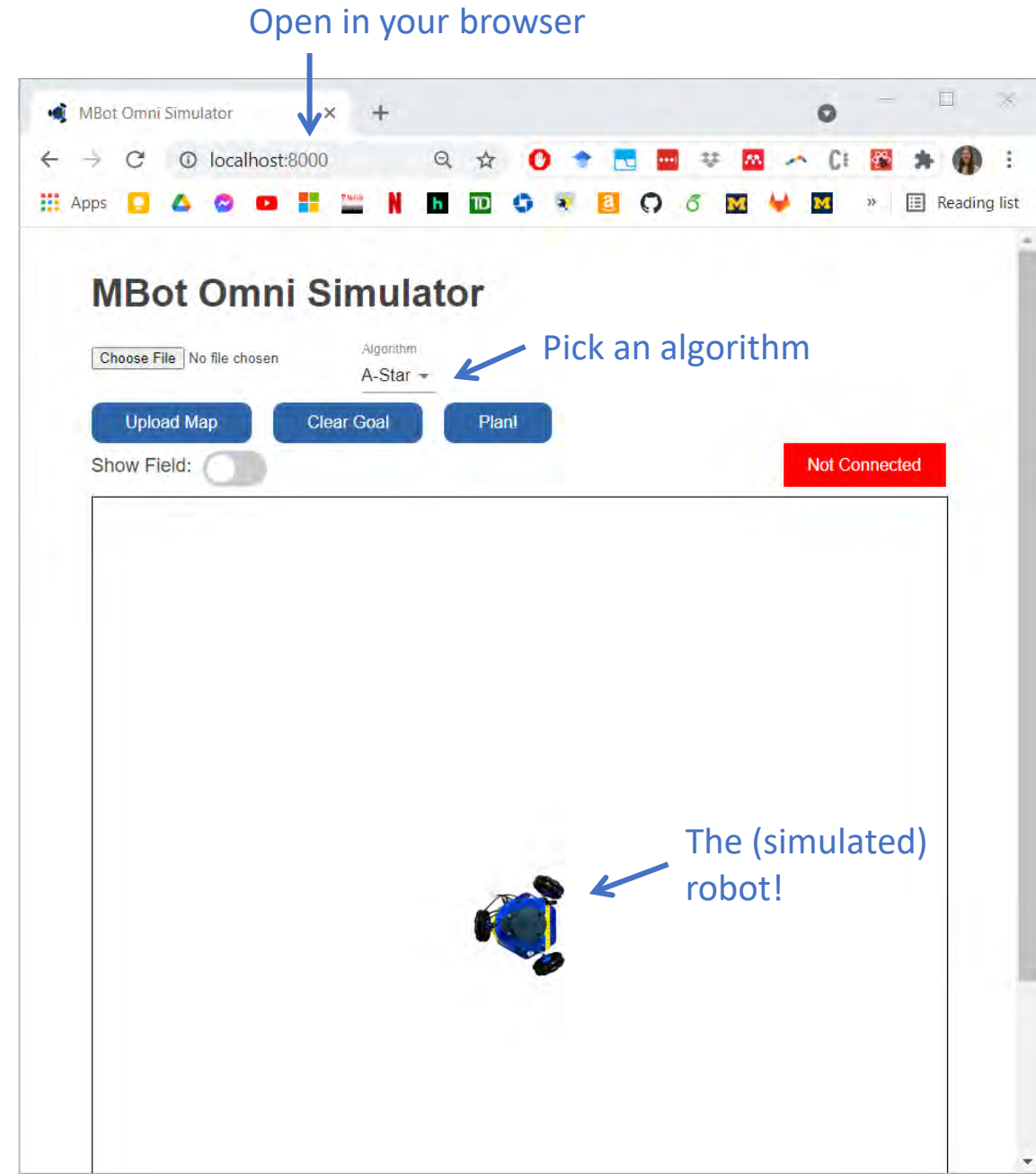
1. Navigation workflow & how to use the navigation web app.
2. Autonomous navigation code structure
3. Storing a map in C++ code
4. Building a map on the robot

## **TODO Today:**

1. Clean up robots from Project 1
2. Get Docker and the web app running
3. Build a map on the robot

# The Navigation Web App

We built a web app to help you visualize maps, your potential fields, and paths.



# Running the Web App

The web app must be run in the provided Docker container.

In a terminal open in the repository directory:

1. Build the Docker image:

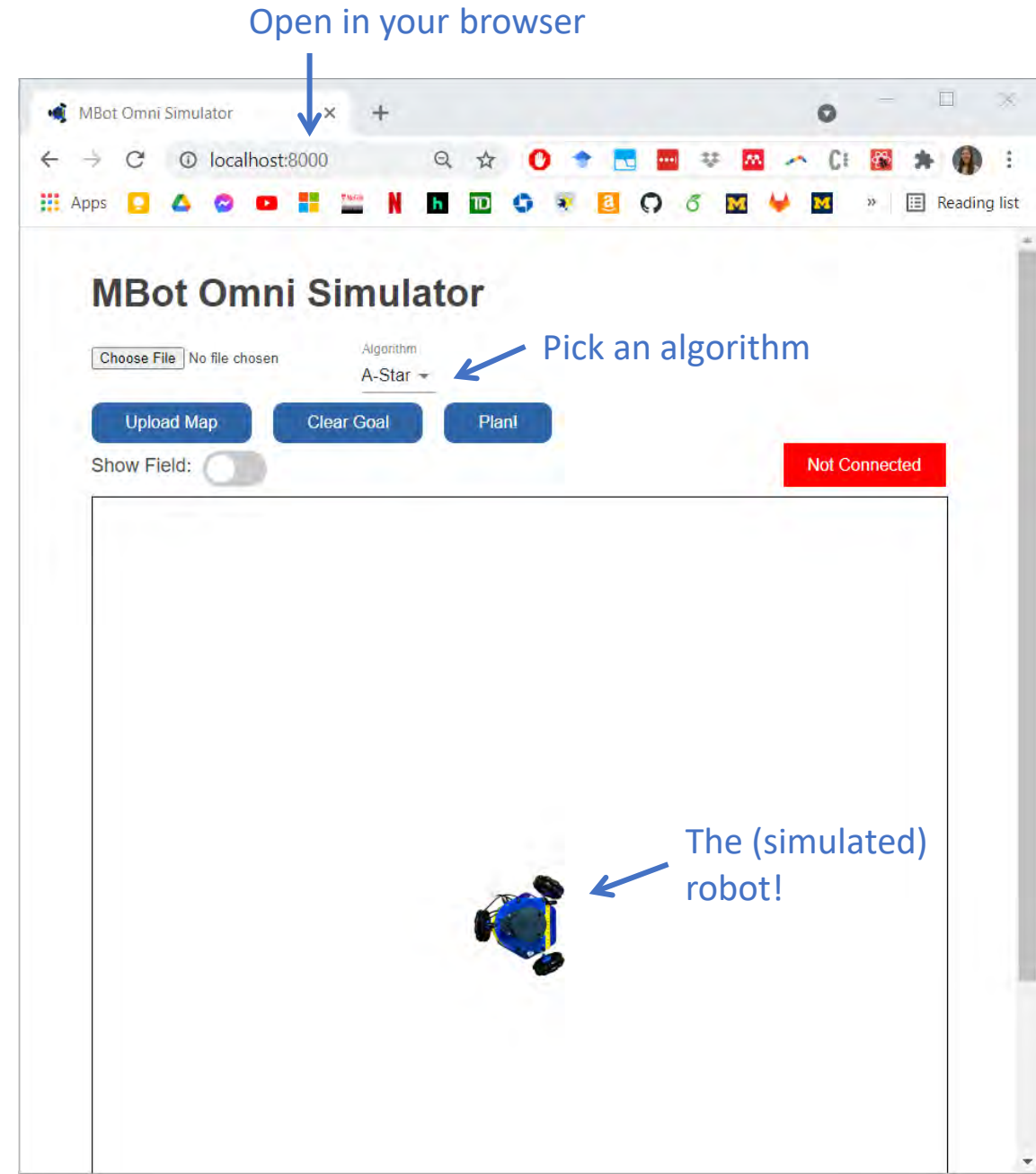
```
./docker_build.sh
```

2. Run the Docker container:

```
./docker_run.sh
```

3. Open a browser and go to:

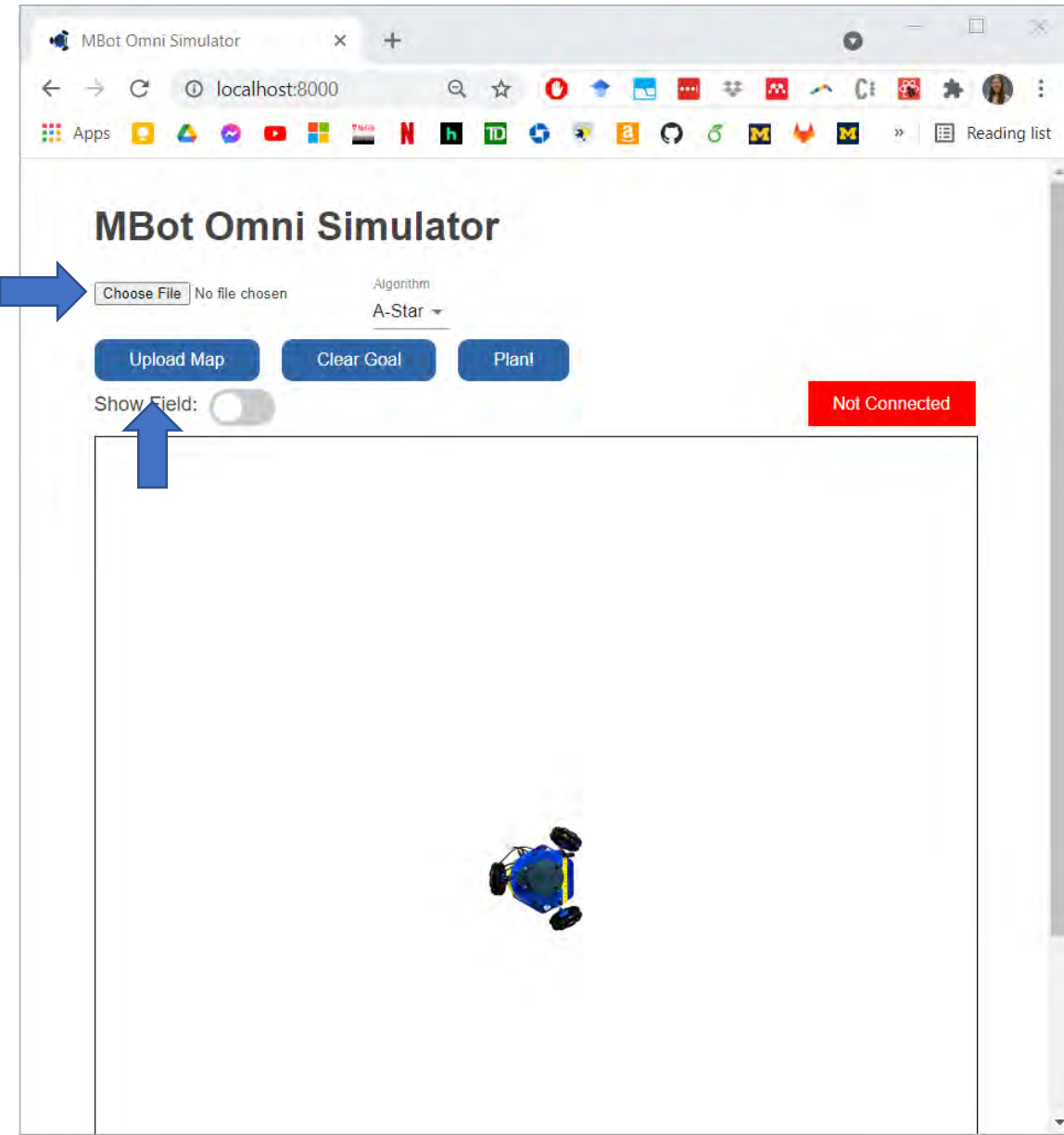
```
http://localhost:8000
```



# Using the Web App

To upload a map:

1. Click the “Choose File” button and navigate to one of the files that ends in “.map” in the “data” folder of the repository.
2. Click “Upload Map”

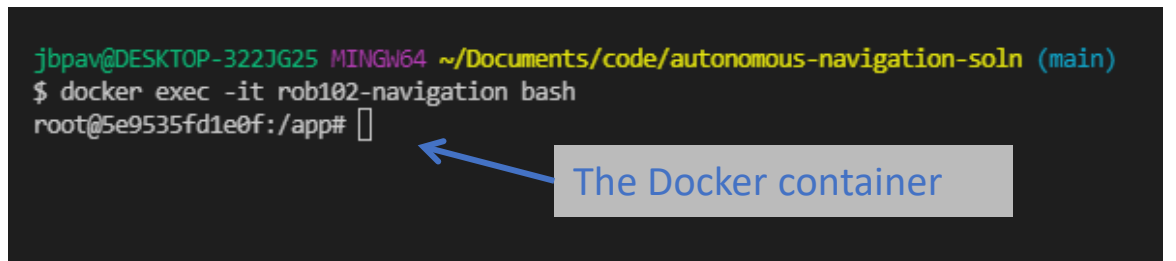
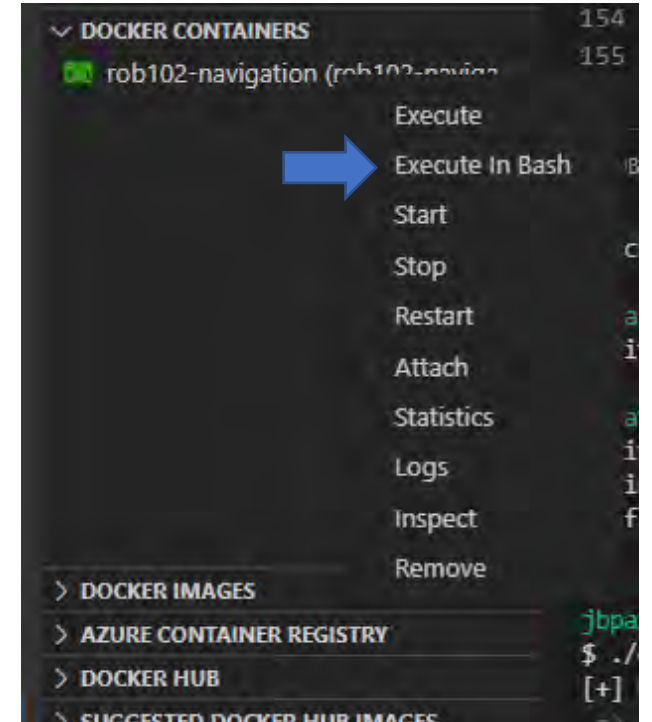


# Testing C++ Code

We have written a C++ interface between the functions you will write and the web app.

1. Open a terminal in the Docker from the Docker Explorer in VSCode.
2. Compile and run the code in the Docker:

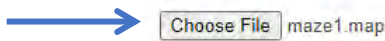
```
cd /code/build
cmake ..
make
./nav_app_server
```



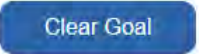
# Testing C++ Code

## MBot Omni Simulator

Choose a map file



Algorithm  
Potential Field



Show Field:



The status is "connected" if the C++ server is connected

Upload the map (to the web app and the server)





# Testing C++ Code

## MBot Omni Simulator

Choose File maze1.map

Algorithm  
Potential Field ▾

Upload Map

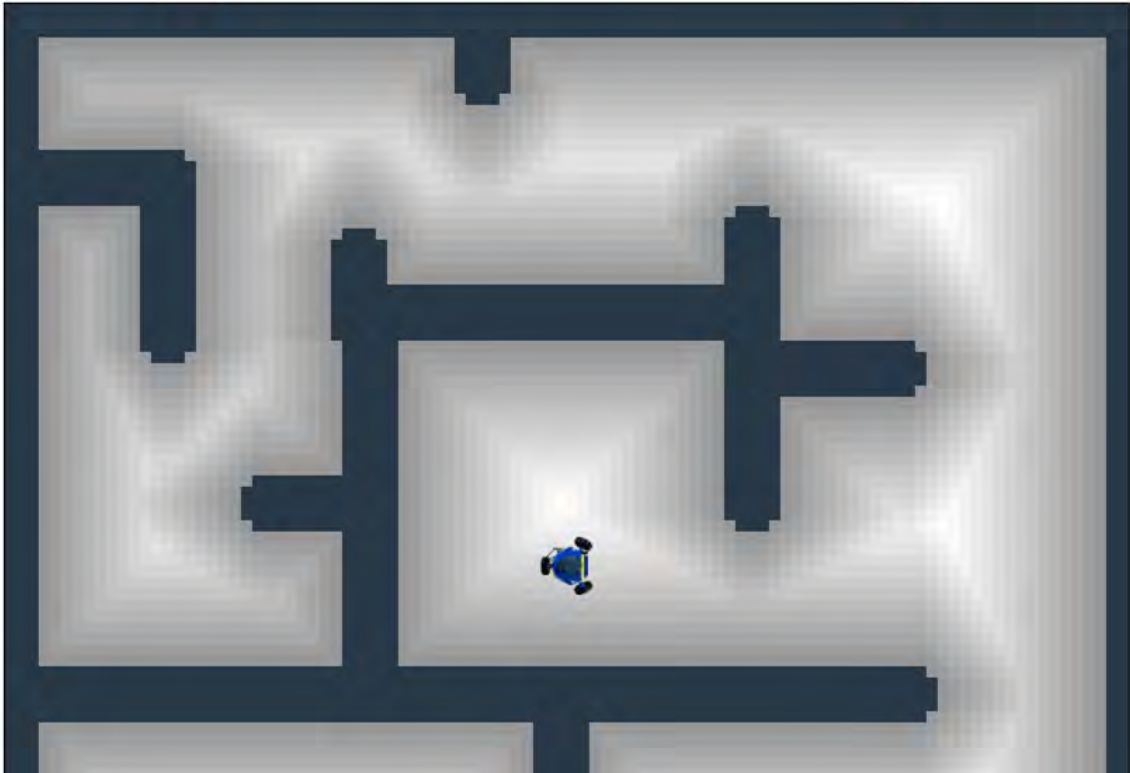
Clear Goal

Plan!

Show Field:

Connected

Show the distance transform (computed in C++)



# Code Structure

The screenshot shows the GitHub interface for the repository 'robotics102 / autonomous-navigation'. The repository is private and has a 'main' branch with 1 branch and 0 tags. The commit history shows a recent commit by 'janapavlassek' titled 'Add find neighbors function solution' with commit hash '96ac0e1' and 4 commits. The file list includes folders for 'autonomous\_navigation', 'build', 'data', 'omnibot\_msgs', and 'scripts', and files for 'CMakeLists.txt', 'Dockerfile', 'README.md', 'docker\_build.sh', and 'docker\_run.sh'. Blue arrows point from text annotations to these specific elements.

File/Folder	Commit Message	Time
autonomous_navigation	Add find neighbors function solution	29 minutes ago
build	Init template files	5 hours ago
data	Init template files	5 hours ago
omnibot_msgs	Init template files	5 hours ago
scripts	Init template files	5 hours ago
CMakeLists.txt	Init template files	5 hours ago
Dockerfile	Init template files	5 hours ago
README.md	Init template files	5 hours ago
docker_build.sh	Init template files	5 hours ago
docker_run.sh	Init template files	5 hours ago

The C++ code for P2 and P3 (this folder is mounted in Docker)



Run cmake and make in this folder



Map files are in here



Compile instructions



Docker files



# Code Structure

robotics102 / **autonomous-navigation** Private

<> Code Issues Pull requests Actions Projects Security Insights Settings

main autonomous-navigation / autonomous\_navigation / src /

janapavlese Add robot potential field control script

..		
graph_search	P3 code	Init template files
potential_field	P2 code (modify here)	Add robot potential field control script
utils	Utilities (some functions must be modified)	Add find neighbors function solution
robot_potential_field.cpp	This code runs on robot	Add robot potential field control script
web_server.cpp	Web server runs in Docker	Init template files

# Code structure

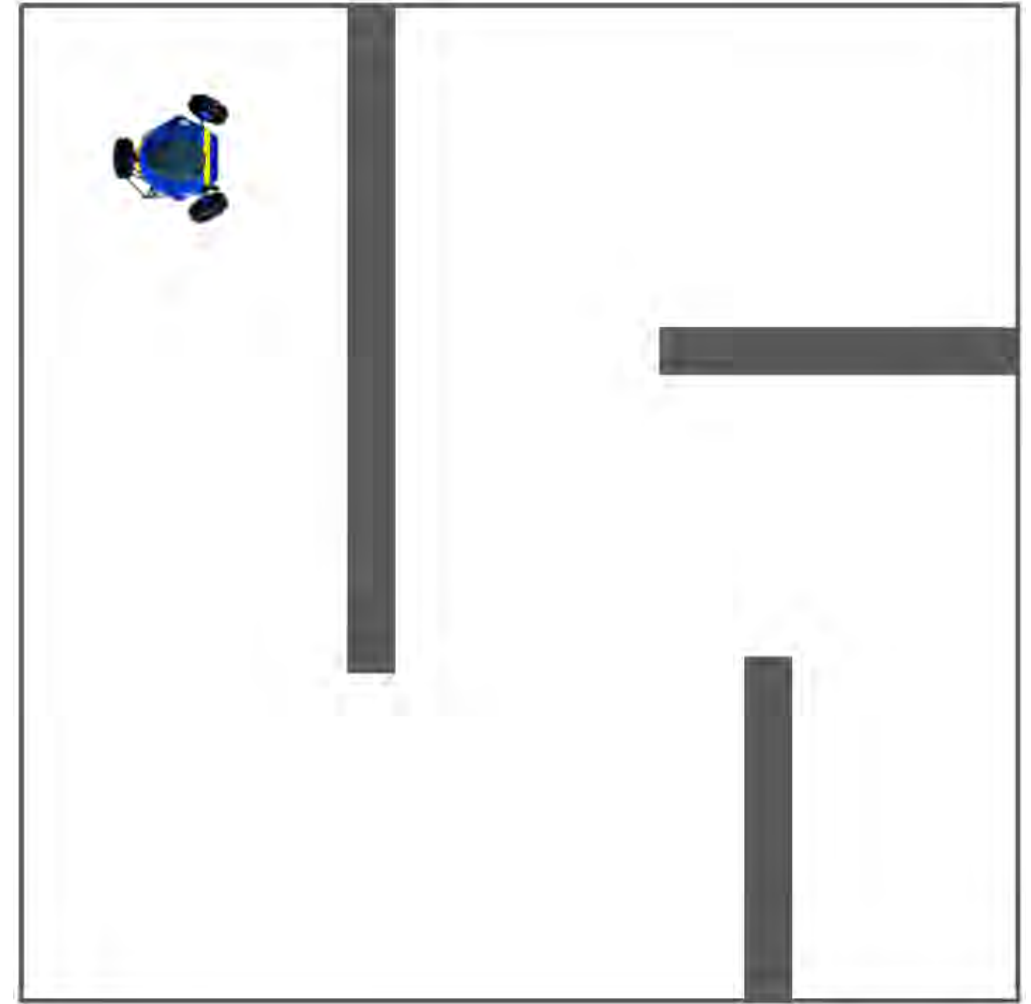
There are three things you will run for this project:

1. **Navigation web server**: When you want to run your algorithms and visualize the field, and simulate the robot.
  - Run in the Docker on your computer.
2. **Test files**: To run tests on components of your algorithm on small scale examples for sanity-checking.
  - Run in the Docker on your computer, or on the robot.
3. **Robot control scripts**: To control the robot. The robot code will use the same functions for 1. and 2.
  - Run on the robot.

# Storing a Map in C++

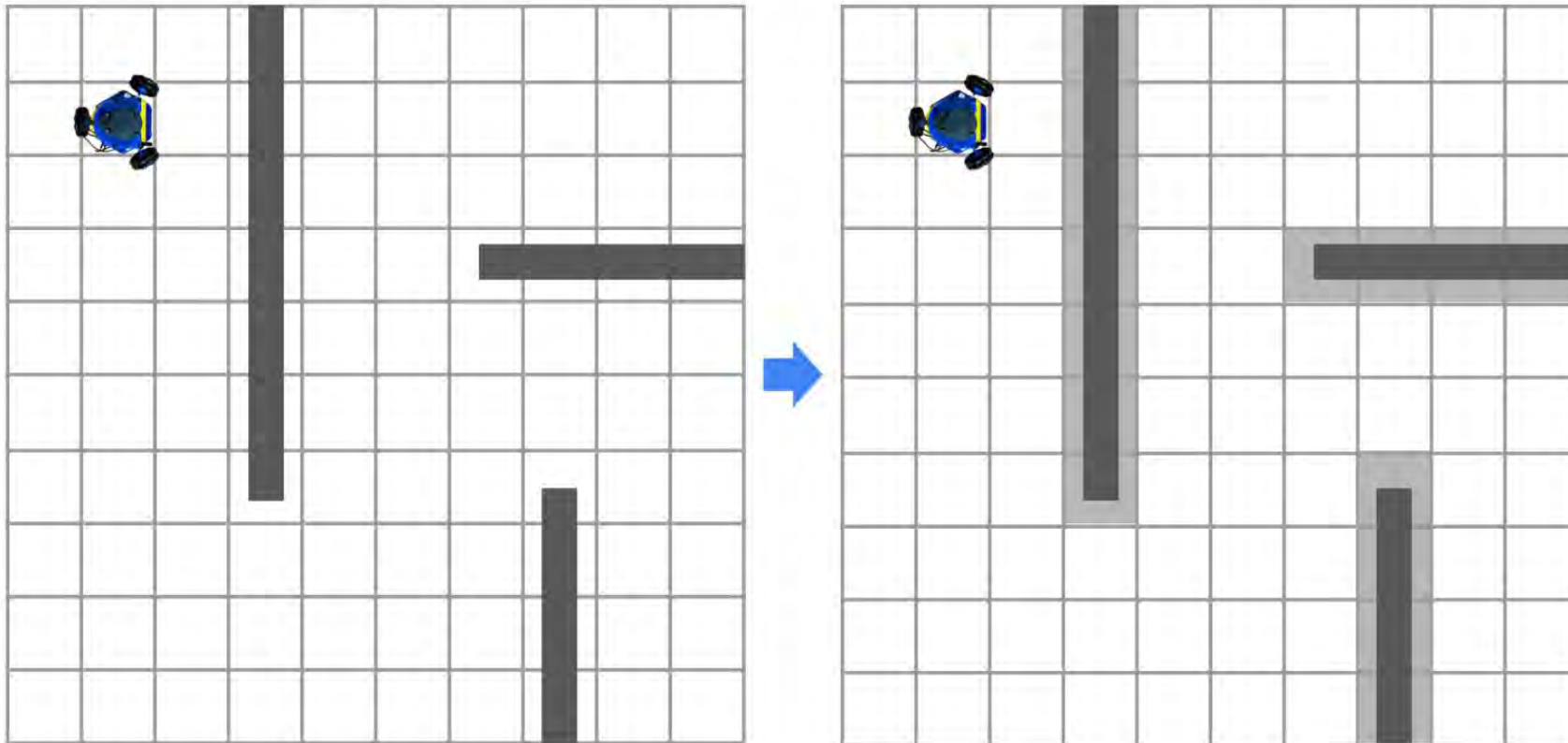
Imagine you have a robot in a map like this one.

How can we represent the map in a computational structure?



# Storing a Map in C++

We can draw a grid in the environment and mark every cell as either free (white) or occupied (grey).

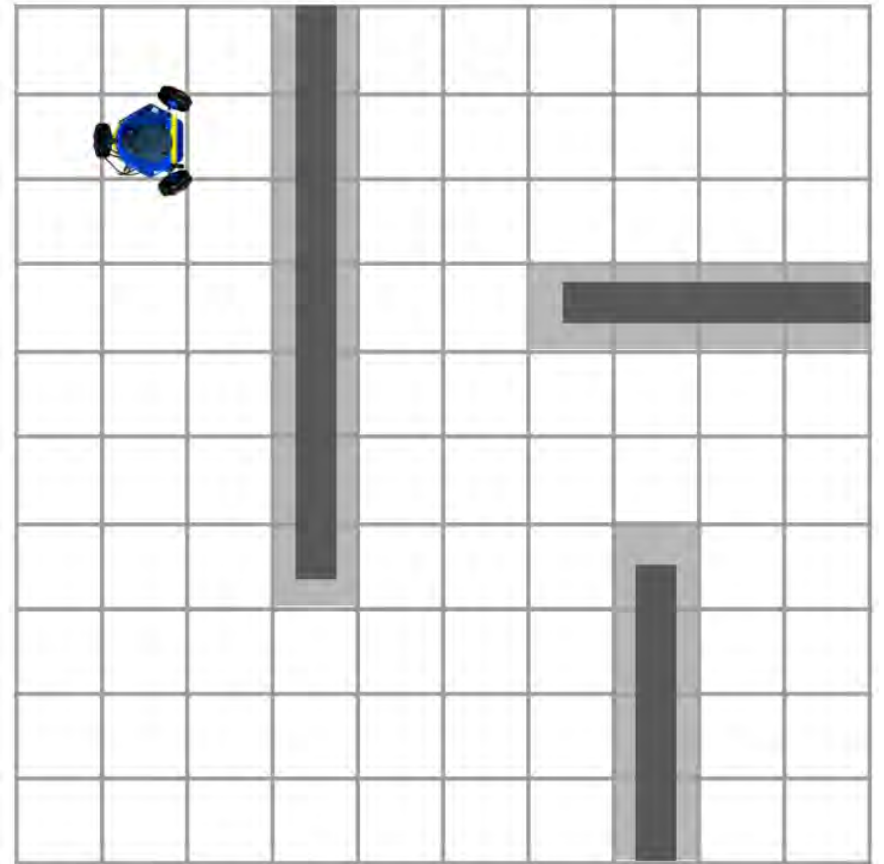


# Storing a Map in C++

GridGraph has a vector called `cell_odds` which has a value in the range  $[-128, 127]$  for each cell.

A **high** value means the cell is likely to be **occupied**.

A **low** value means the cell is likely to be **free**.



# Storing a Map in C++

We represent the cell in terms of a coordinate in the grid.

The coordinate is written  $(i, j)$ , where  $i$  is the index of the row and  $j$  is the index of the column.

9	-128	-128	-128	127	-128	-128	-128	-128	-128	-128
8	-128	-128	-128	127	-128	-128	-128	-128	-128	-128
7	-128	-128	-128	127	-128	-128	-128	-128	-128	-128
6	-128	-128	-128	127	-128	-128	127	127	127	127
5	-128	-128	-128	127	-128	-128	-128	-128	-128	-128
4	-128	-128	-128	127	-128	-128	-128	-128	-128	-128
3	-128	-128	-128	127	-128	-128	-128	127	-128	-128
2	-128	-128	-128	-128	-128	-128	-128	127	-128	-128
1	-128	-128	-128	-128	-128	-128	-128	127	-128	-128
0	-128	-128	-128	-128	-128	-128	-128	127	-128	-128
	0	1	2	3	4	5	6	7	8	9

rows

columns



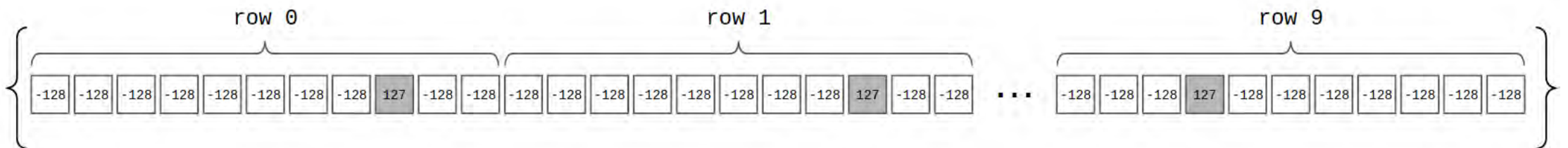
# Storing a Map in C++

We will store the cell odds data in a single vector.

We can line up each row next to each other.

**P2.0:** Write functions to convert from a cell coordinate to an index in the vector (and the inverse).

9	-128	-128	-128	127	-128	-128	-128	-128	-128	-128
8	-128	-128	-128	127	-128	-128	-128	-128	-128	-128
7	-128	-128	-128	127	-128	-128	-128	-128	-128	-128
6	-128	-128	-128	127	-128	-128	127	127	127	127
5	-128	-128	-128	127	-128	-128	-128	-128	-128	-128
4	-128	-128	-128	127	-128	-128	-128	-128	-128	-128
3	-128	-128	-128	127	-128	-128	-128	127	-128	-128
2	-128	-128	-128	-128	-128	-128	-128	127	-128	-128
1	-128	-128	-128	-128	-128	-128	-128	127	-128	-128
0	-128	-128	-128	-128	-128	-128	-128	127	-128	-128
	0	1	2	3	4	5	6	7	8	9



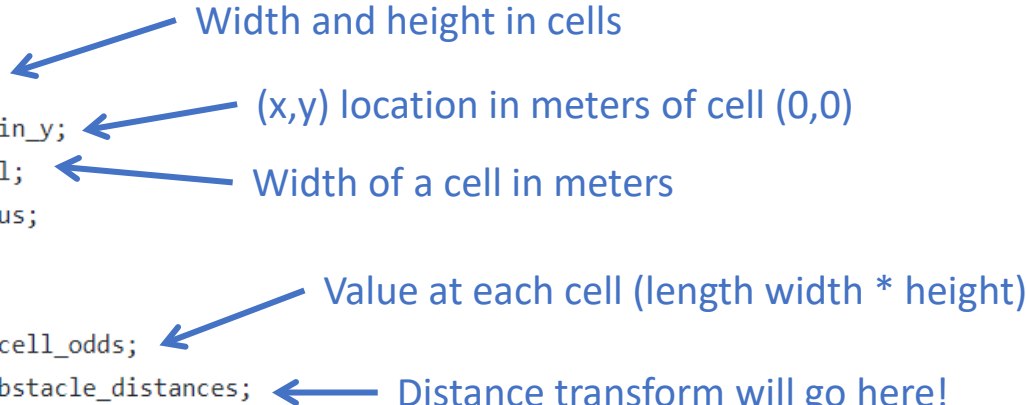
# Storing a Map in C++

```
struct GridGraph
{
    GridGraph() :
        width(-1),
        height(-1),
        origin_x(0),
        origin_y(0),
        meters_per_cell(0),
        collision_radius(0.15),
        threshold(-100)
    {
    };

    int width, height;
    float origin_x, origin_y;
    float meters_per_cell;
    float collision_radius;
    int8_t threshold;

    std::vector<int8_t> cell_odds;
    std::vector<float> obstacle_distances;
};

include/autonomous_navigation/utils/graph_utils.h
```



Width and height in cells

(x,y) location in meters of cell (0,0)

Width of a cell in meters

Value at each cell (length width \* height)

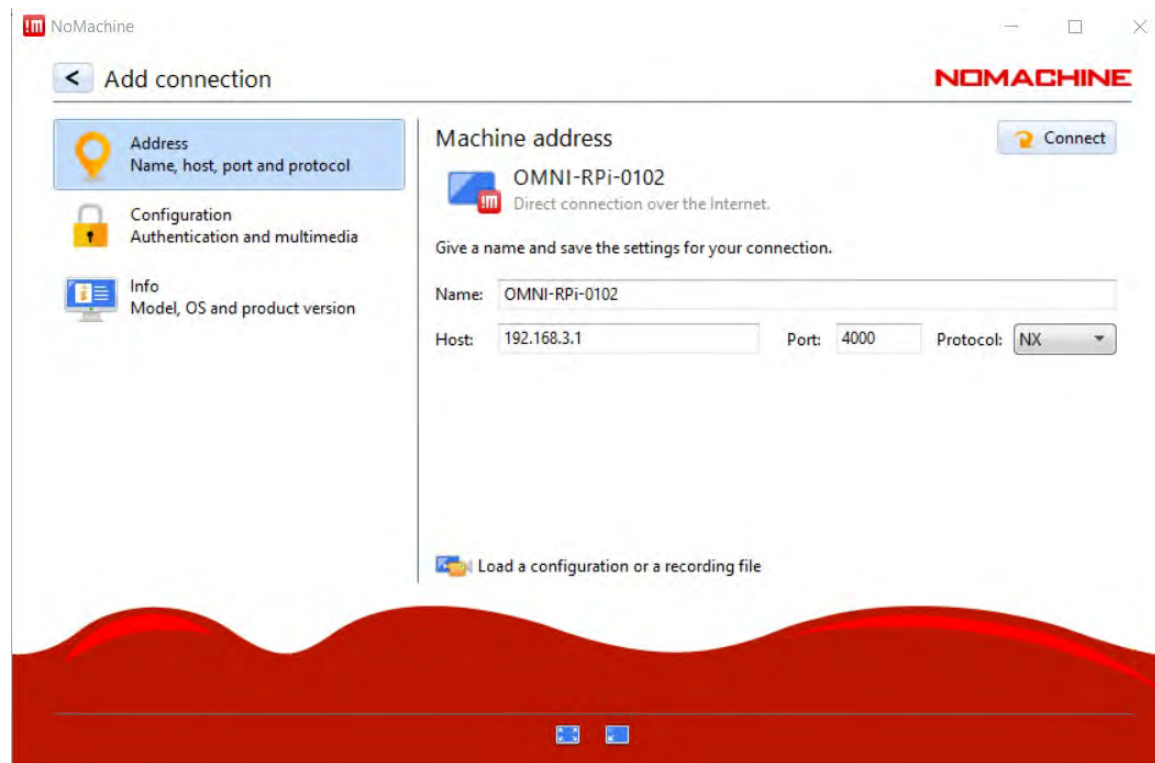
Distance transform will go here!

The GridGraph struct represents the map data.

# Building a Map on the Robot

Download the program NoMachine: <https://www.nomachine.com/>

Create a new connection to the Raspberry Pi.



# Building a Map on the Robot

You will see the robot's desktop!



# Building a Map on the Robot

Open a terminal on the robot.

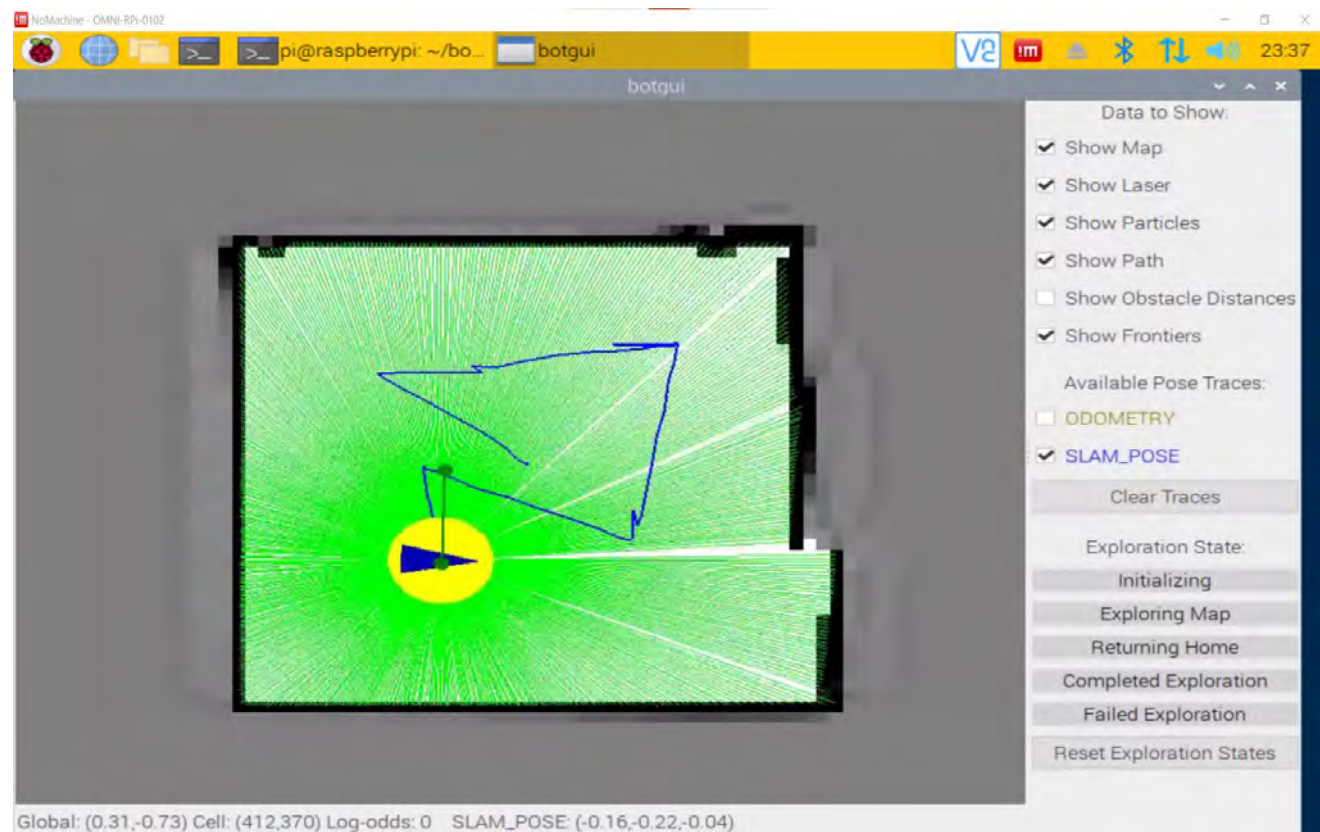
Run:

```
./botlab-bin/launch_botlab.sh
```

Then run:

```
./botlab-bin/bin/botgui
```

Use Ctrl+Click on the GUI to move the robot.



# TODO Today:

1. Clean up robots from Project 1
  - a) Make sure all your code is pushed to GitHub
  - b) Remove your code and any other data
2. Get Docker and the web app running
  - a) Accept the assignment on GitHub (find the link on Slack)
  - b) Clone it onto your computer (all teammates should do this!)
  - c) Build & run the Docker and look at the web app
3. Build a map on the robot