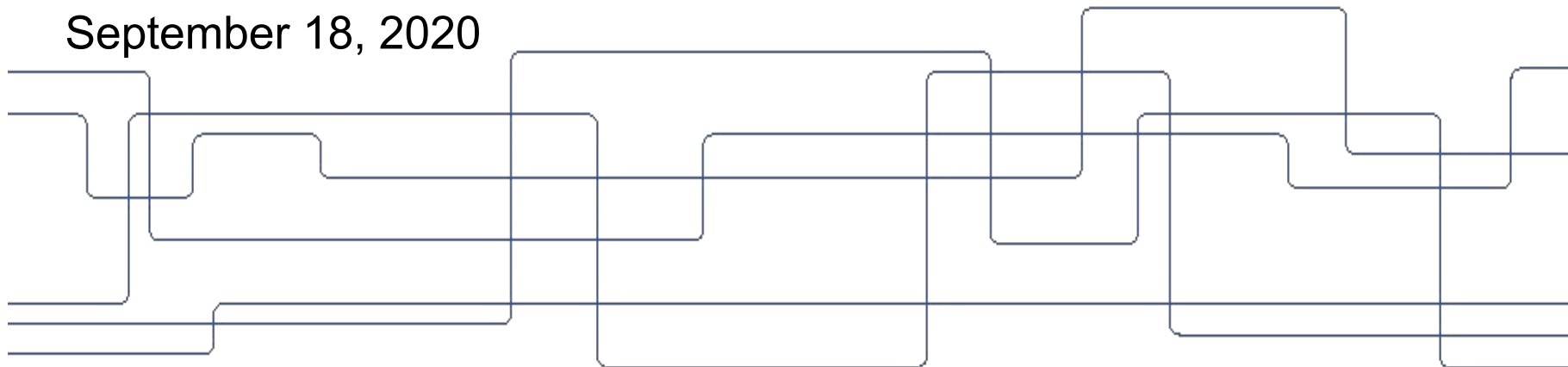


Assignment 4: Mapping

Daniel Duberg (dduberg@kth.se)

DD2410 Introduction to Robotics

September 18, 2020





Ask questions

You are welcome to ask questions at any time during this presentation



Introduction - mapping

- One of the core competencies of truly autonomous robots

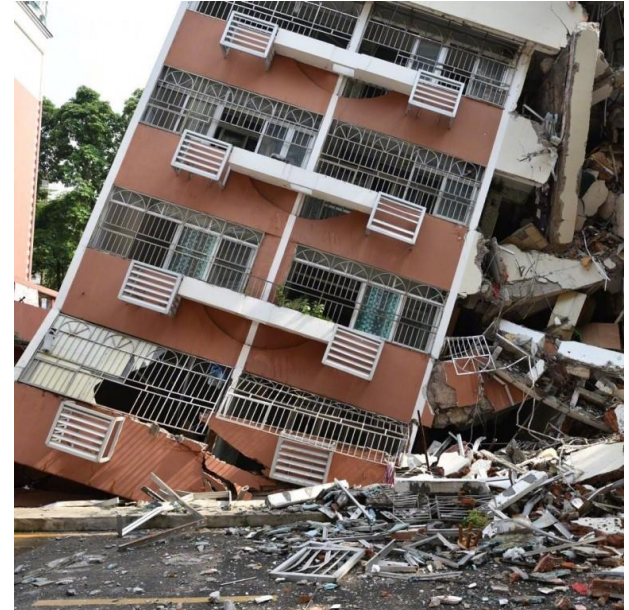


Introduction - mapping

- One of the core competencies of truly autonomous robots
- Can be used by robots in a number of different ways:

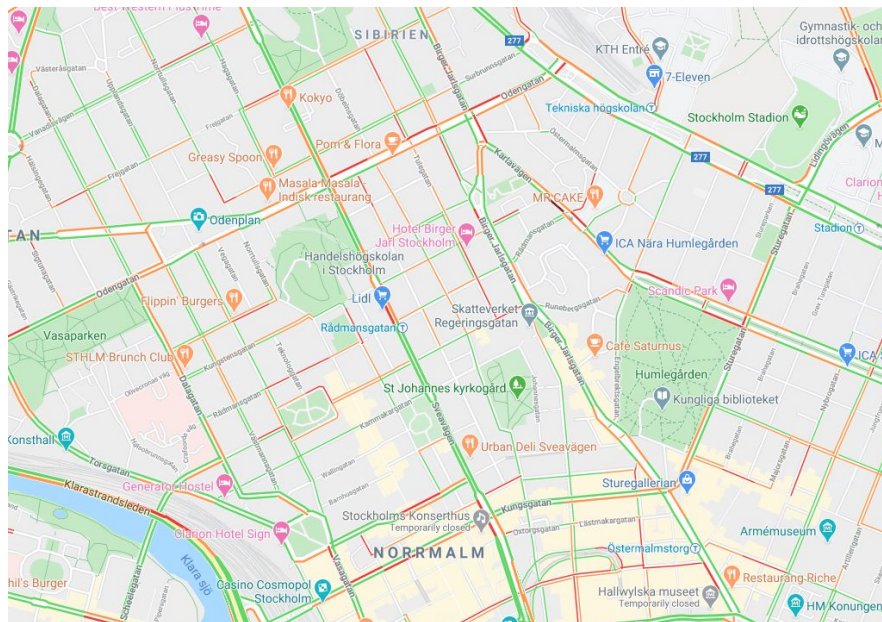
Introduction - mapping

- One of the core competencies of truly autonomous robots
- Can be used by robots in a number of different ways:
 - Search and rescue



Introduction - mapping

- One of the core competencies of truly autonomous robots
- Can be used by robots in a number of different ways:
 - Search and rescue
 - Autonomous cars





Introduction - mapping

- One of the core competencies of truly autonomous robots
 - Can be used by robots in a number of different ways:
 - Search and rescue
 - Autonomous cars
 - To localize themselves
-



Introduction - mapping

- One of the core competencies of truly autonomous robots
 - Can be used by robots in a number of different ways:
 - Search and rescue
 - Autonomous cars
 - To localize themselves
 - In many situations we cannot assume that the robot can be given a map in advance
-



Introduction - mapping

- One of the core competencies of truly autonomous robots
 - Can be used by robots in a number of different ways:
 - Search and rescue
 - Autonomous cars
 - To localize themselves
 - In many situations we cannot assume that the robot can be given a map in advance
 - Even if maps are available, such as blueprints, they are not always useful for the robot and might be incorrect
-

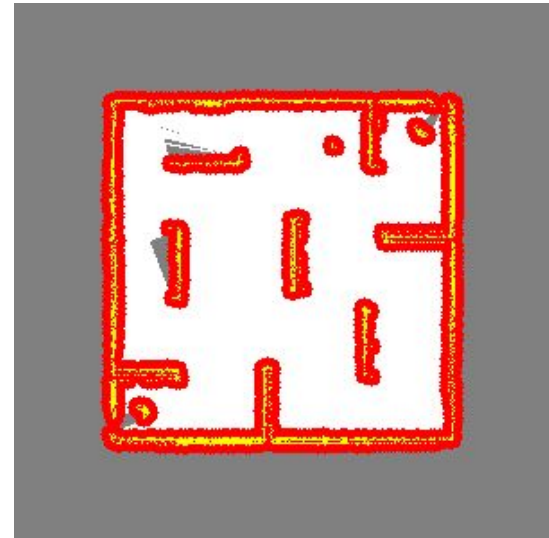
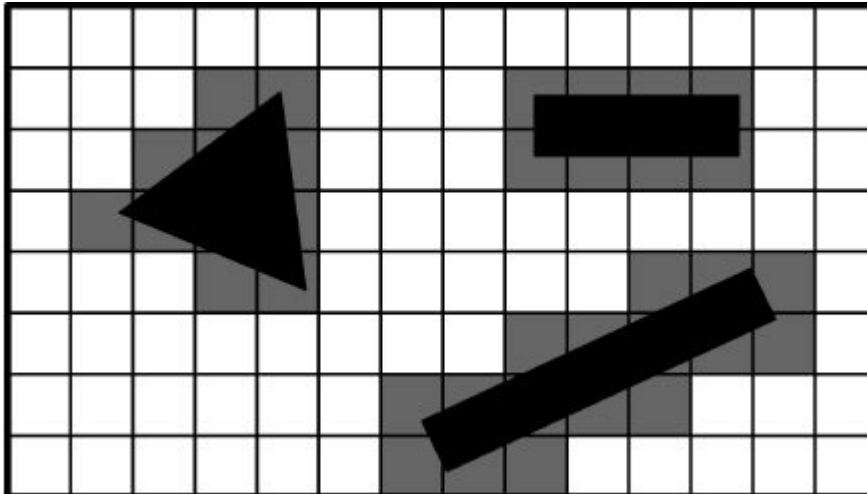


Introduction - mapping

- One of the core competencies of truly autonomous robots
 - Can be used by robots in a number of different ways:
 - Search and rescue
 - Autonomous cars
 - To localize themselves
 - In many situations we cannot assume that the robot can be given a map in advance
 - Even if maps are available, such as blueprints, they are not always useful for the robot and might be incorrect
 - It is therefore of great benefit if the robot can construct a map by itself from scratch
-

Introduction - occupancy grid mapping

- One of many mapping algorithms
- Represents the world as a grid
 - Each cell corresponds to an area in the map
 - The value of the cell tells us if the area is free, occupied, unknown, or something else





The assignment

- A robot is moving around autonomously in a 2D environment
 - It knows its pose
 - It has a laser scanner to gather data about the environment
 - You should use the pose and laser scan data to build a map of the environment
 - You should edit and submit the file:
mapping_assignment/scripts/mapping.py
-



Rosbag

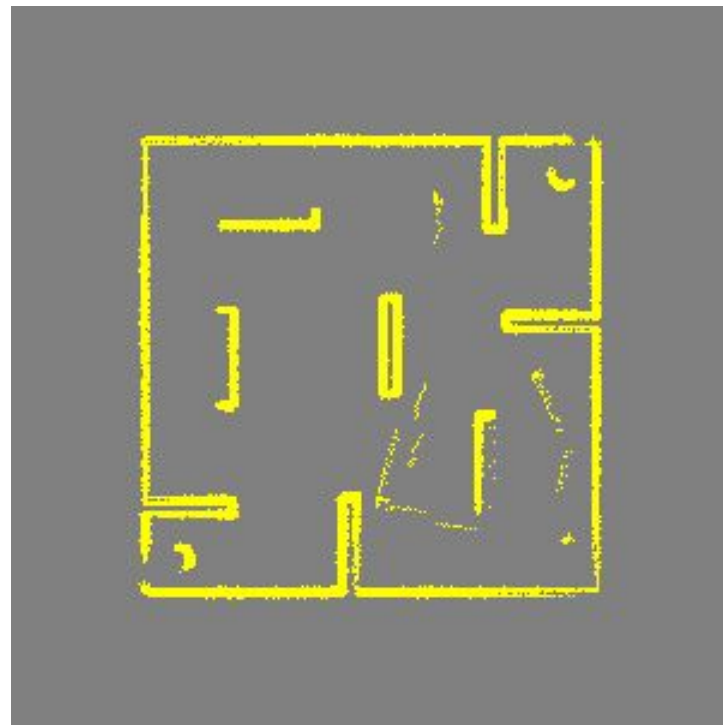
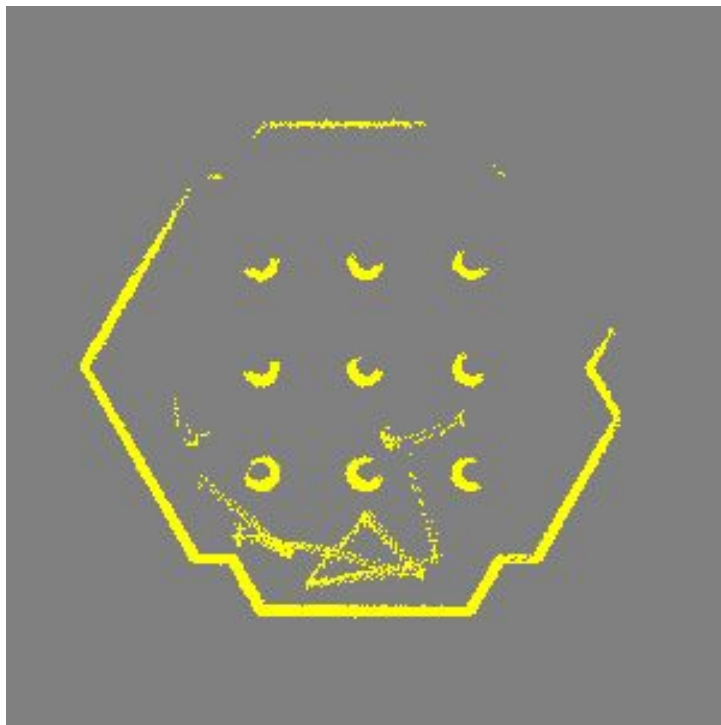
- For this assignment you are given a number of rosbags that we have recorded
 - A rosbag contains messages from different ROS topics that have been recorded at an earlier occasion
 - They are useful during development and for comparing different algorithms, since you get - almost - the exact same data every time you run the rosbag and you do not have to run a simulator or a real robot every time you want to test your code
-



E assignment

1. Convert the laser scan ranges and bearings to coordinates in the laser frame
2. Convert the coordinates to the map frame
3. Convert the coordinates to map indices
4. Fill in the occupied cells
5. Compare your map to the correct map and when they match submit to Kattis

E assignment





C assignment - part 1

- In the previous images we could identify at least two problems:
 1. The maps are noisy, why?
 2. We only know about occupied space (yellow), everything else is unknown (gray)
 - We know that the cells between the robot's position and the scan endpoints are free. So we should raytrace and mark the cells as free (white)
 - It is also wasteful to update the whole map each time. Therefore you should return the updated part of the map
 - Fill in the function:
update_map(self, grid_map, pose, scan)
-



C assignment - part 2

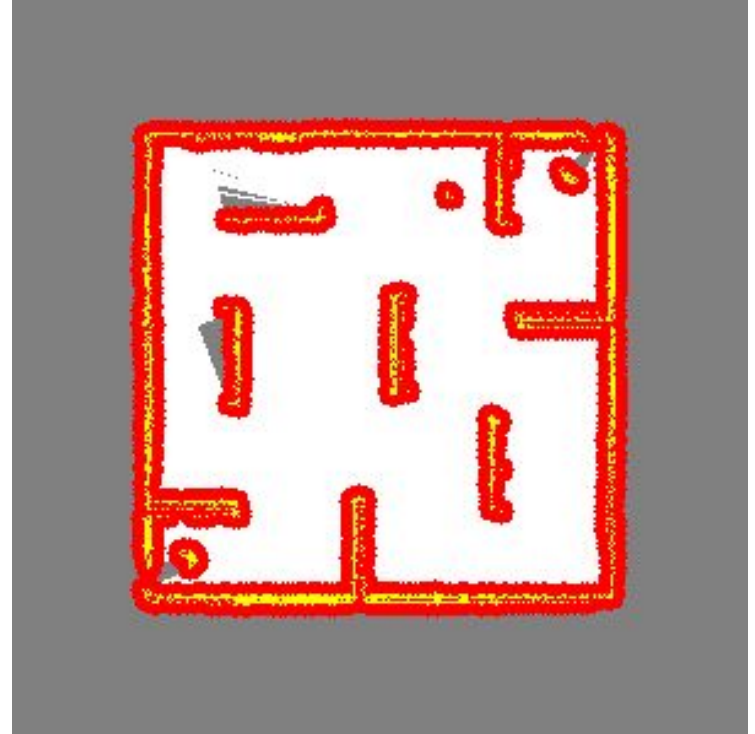
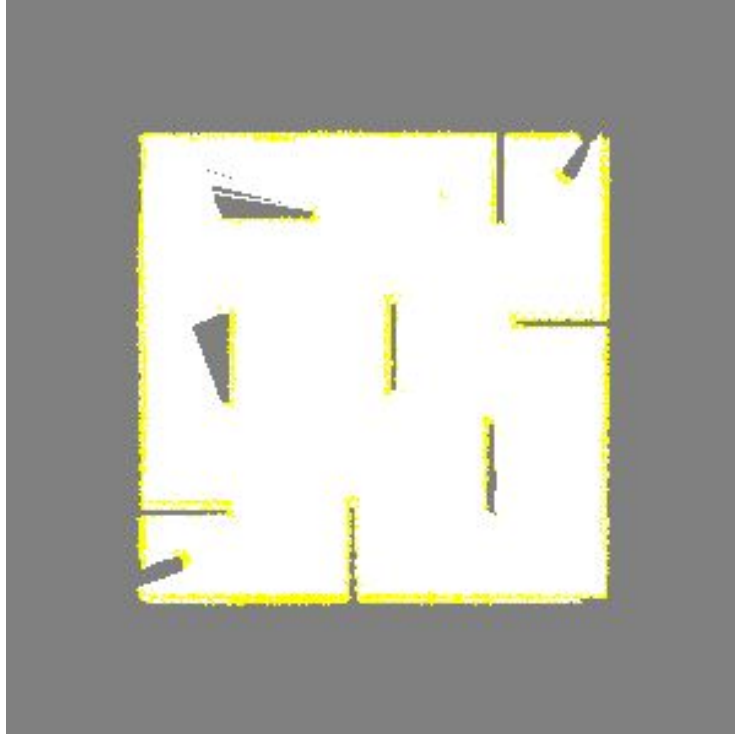
- Before the robot uses the map for planning the grid map will typically be processed to create a so called C-space (configuration-space) map
 - This is done by expanding each obstacle, i.e. each cell, in the map with a circle with the same radius as the robot
 - When this is done we can treat the robot as a point in all further calculations
 - This makes planning a lot faster because all we need to check to see if the robot can be at a certain location is to see if the cell is free or not
 - Fill in the function:
inflate_map(self, grid_map)
-



C assignment

1. Clear free space between the scan endpoints and the robot's position, using the *raytrace(self, start, end)* function
 2. Fill in occupied space as described in the E assignment
 3. Fill in *update = OccupancyGridUpdate()* to only return the updated part of the map
 4. Expand occupied space to create C-space in the *inflate_map(self, grid_map)* function
-

C assignment





How to run

- Two options:
 1. Rosbags (preferred)
 - *This is what you normally use when working with ROS*
 2. Text files (good before you submit to Kattis)
 - *The maps will be saved in a folder called "maps"*
-



Hints

- Use *int(X)* when converting from float to int, do **not** use *round(X)*
 - To check if you have done the update part of the C assignment you can use *rostopic echo*
 - To see what you are publishing on that topic run:
 - `rostopic echo /map_updates`
 - To remove the array from the output add “--noarr” (no array), this makes it easier to read:
 - `rostopic echo /map_updates --noarr`
 - Check if the x, y, width, height, and array length seem reasonable
 - Remember that the data field of the update should only hold a 1D array
 - The order in which you update the map is important for the C assignment. Be sure that you do **not** overwrite something you should not do
-



Hints - grid raytracing

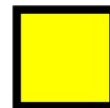
The order in which you update the map is important for the C assignment. Be sure that you do **not** overwrite something you should not do



Unknown space



Free space



Occupied space



Robot



Real world obstacle



Laser beam 1










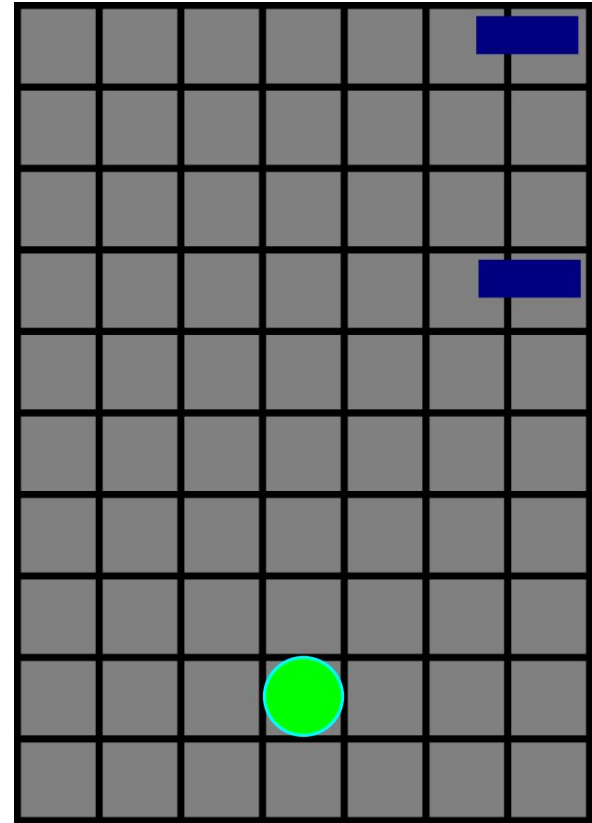
Laser beam 2



Hints - grid raytracing

The order in which you update the map is important for the C assignment. Be sure that you do **not** overwrite something you should not do

-  Unknown space
-  Free space
-  Occupied space
-  Robot
-  Real world obstacle
-  Laser beam 1
-  Laser beam 2



Hints - grid raytracing

- We get two range readings
- We want to mark the cells between the robot and the end of the rays as **free**



Unknown space



Free space



Occupied space



Robot



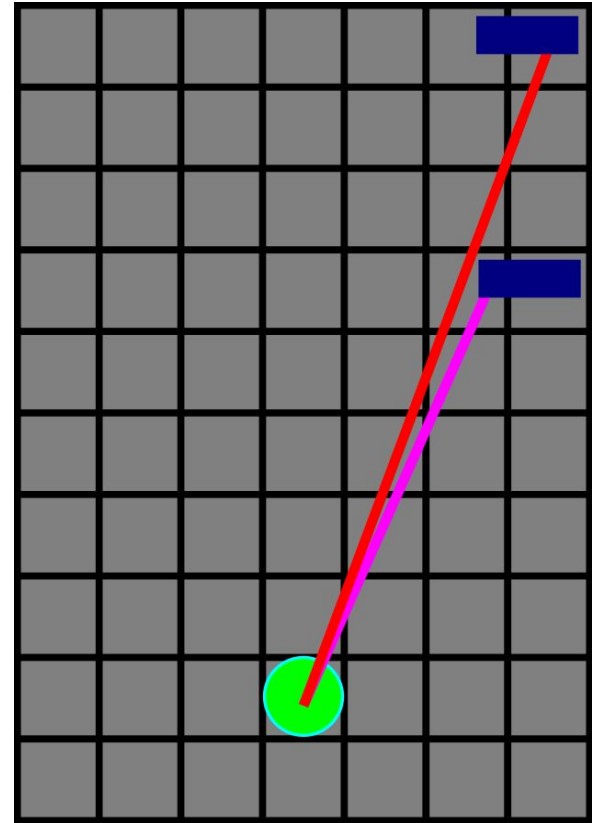
Real world obstacle



Laser beam 1

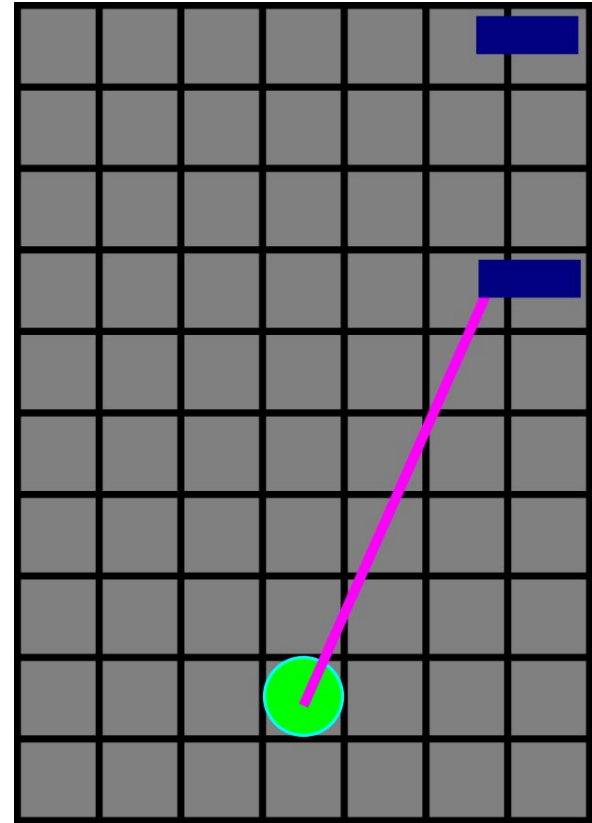
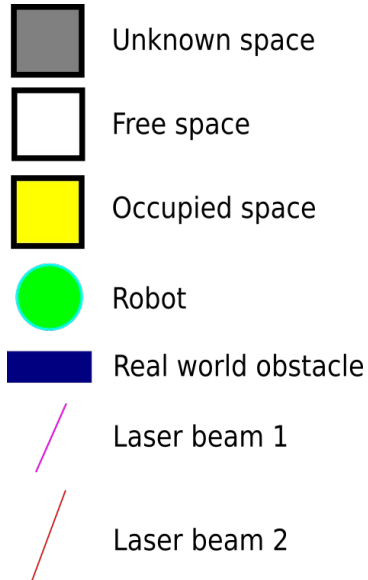


Laser beam 2



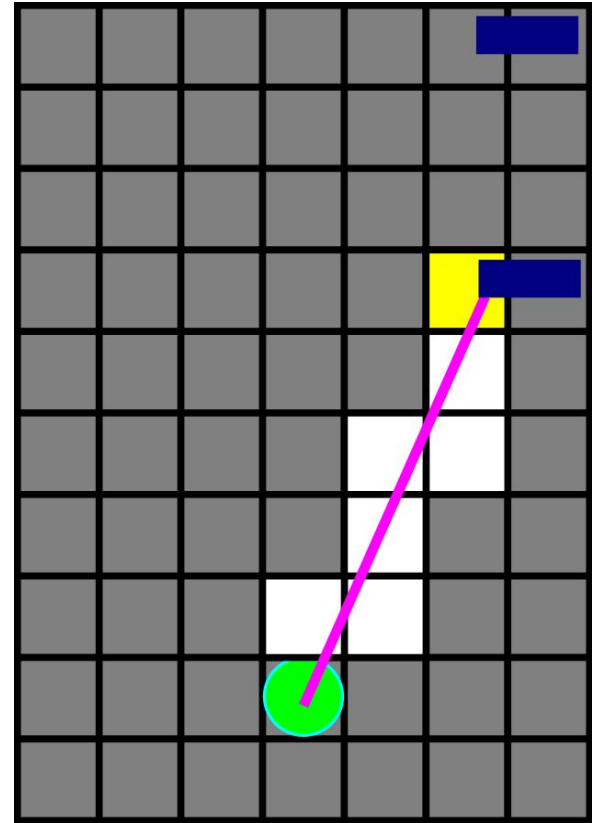
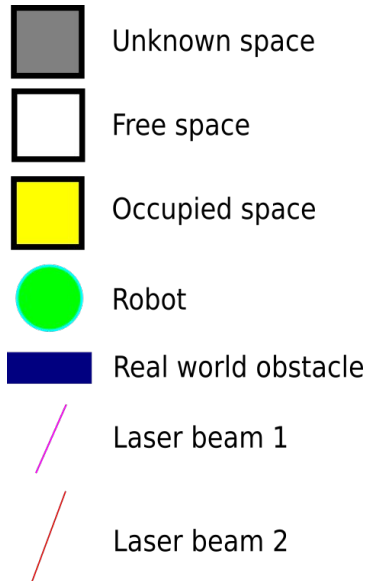
Hints - grid raytracing

- We start of by processing the first range reading



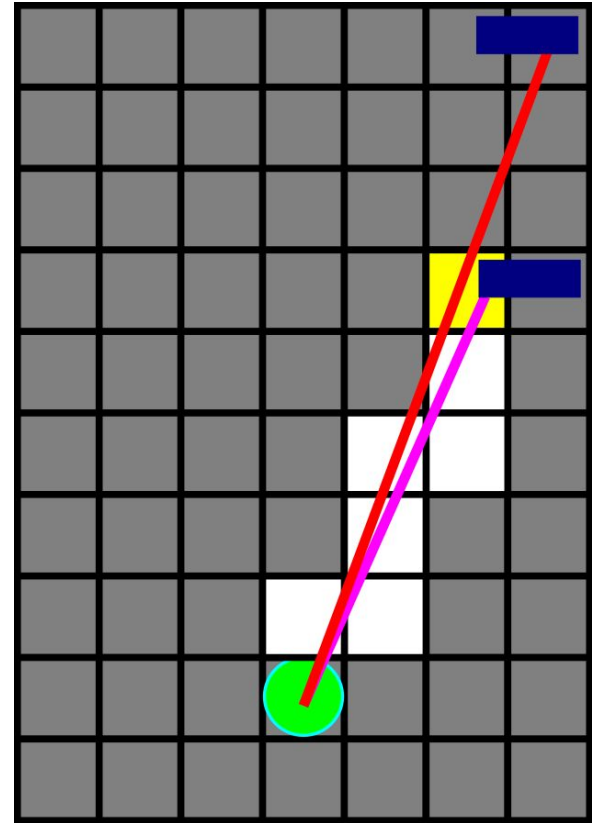
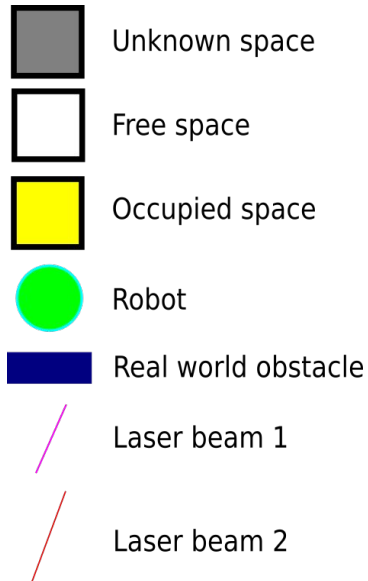
Hints - grid raytracing

- As you can see in the picture we marked some cells as **free** and one as **occupied**



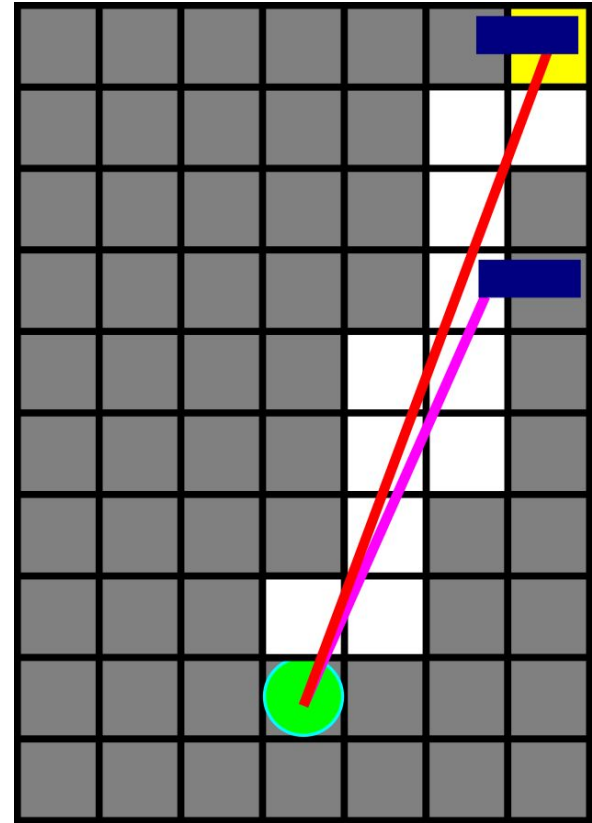
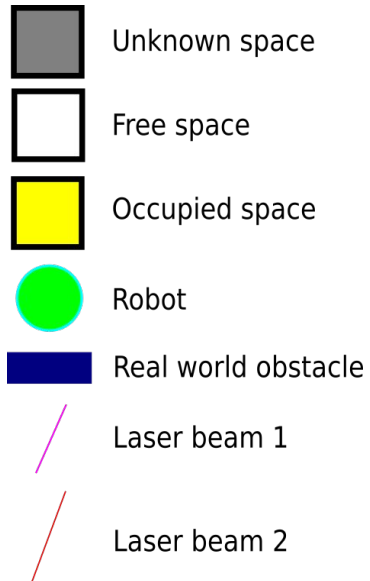
Hints - grid raytracing

- Now we start processing the second range reading



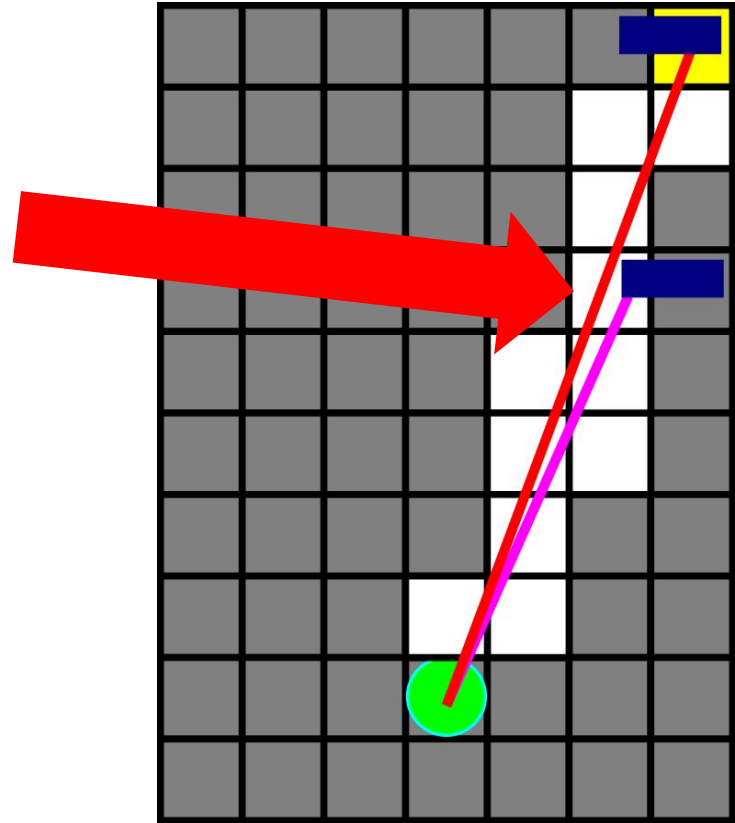
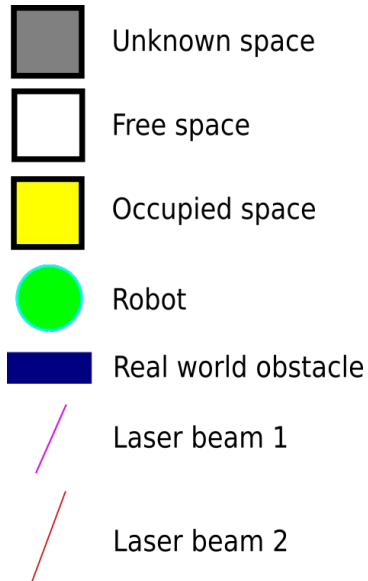
Hints - grid raytracing

- **Ops!** What went wrong?



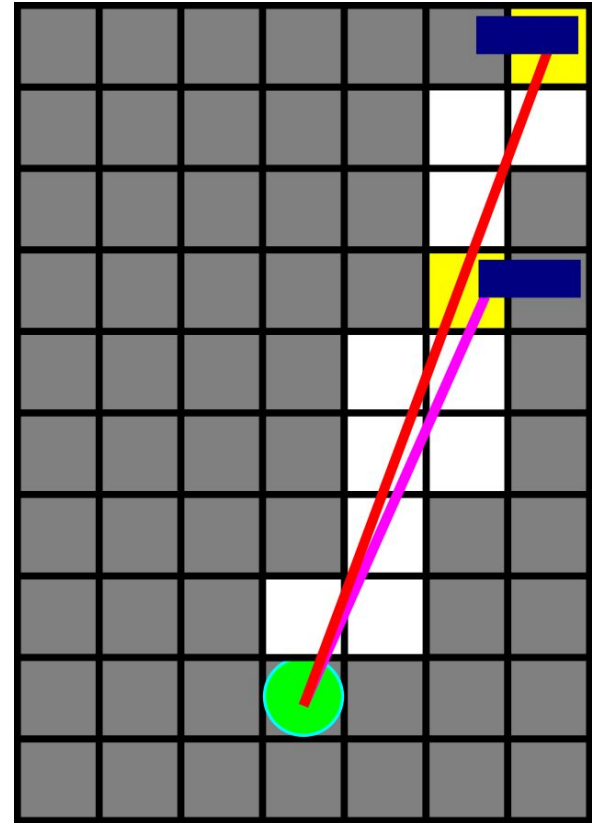
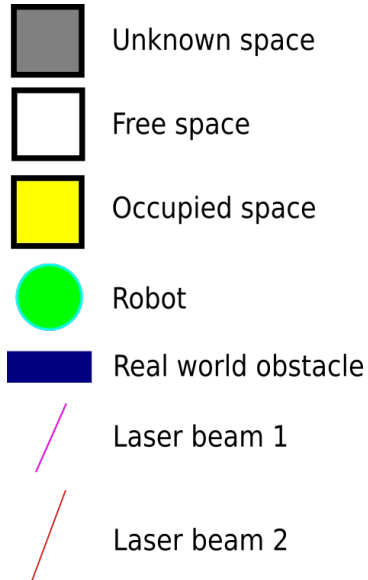
Hints - grid raytracing

- We marked the cell we just marked **occupied** (from the first range reading) as **free**



Hints - grid raytracing

- This is how it should look like
- Fill in **occupied** cells AFTER you have filled in ALL **free** cells





Hints - grid raytracing

The order in which you update the map is important for the C part of the assignment. Be sure that you do **not** overwrite something you should not do



Unknown space



Free space



Occupied space



Robot



Real world obstacle



Laser beam 1



Laser beam 2



The end

Questions?





ROS messages - geometry_msgs/PoseStamped

- Contains a pose and a header
 - The pose consists of the position and the orientation (in quaternion form) of the robot
 - You can ignore all the headers in this assignment. They are however very useful when working with ROS
-



ROS messages - sensor_msgs/LaserScan

- Contains a laser scan
 - angle_min - Start angle of the scan in radians
 - angle_max - End angle of the scan in radians
 - angle_increment - Angular distance between measurements in radians
 - range_min - Minimum range value in meters
 - range_max - Maximum range value in meters
 - ranges - Range data in meters (values $<$ range_min or $>$ range_max should be discarded)
- Again, ignore the header



ROS messages - nav_msgs/OccupancyGrid

- Contains the actual occupancy grid
 - The map data is stored in data, in row-major order, starting with (0,0)
 - Also contains a header
 - Also contains meta data for the map, in form of nav_msgs/MapMetaData
 - You will **not** be working directly with this message, instead you will be working with a class called *GridMap* which is easier to use
-



ROS messages - map_msgs/OccupancyGridUpdate

- Only used in the C assignment
 - Enables us to only send the portion of the map that has been updated
 - It contains a header, which you should ignore
 - x - Starting x coordinate of the rectangle area (should be minimum x)
 - y - Starting y coordinate of the rectangle area (should be minimum y)
 - width - The width of the rectangle area
 - height - The height of the rectangle area
 - data - The actual map data for the rectangle area, in row-major order, starting with (x,y)
-