Yasser Elnemr, ID: 00831018, Yasser.elnemr@student.tugraz.at
Felix Breuß, ID: 11771855, fbreuss@student.tugraz.at

Exercise 4: Mapping

Introduction

In this assignment, the aim is to create a 2D globally consistent grid-map using odometry and a range sensor. A grid-map is a grid structure of random variables that represent the probability that a particular area of the environment is occupied by an object like a wall. Generally, maps are not available in advance and need to be obtained by the robot via a mapping process. Since the robot's sensor readings are uncertain and the map usually cannot be directly obtained, then the mapping process is a state estimation problem with two interlinked estimations. The mapping process is generally named the *Simultaneous Mapping and Localization Problem* (SLAM) because the position or path of the robot and the map needs to be estimated simultaneously.

To ease the SLAM problem an assumption was made that the pose or path of the robot is already known, then the problem is only to estimate the map.

Robot, simulation, and Visualization

The robot used in this assignment is a differential drive robot named TurtleBot3 which is fully described in the user manual [1]. The robot provides odometry information where the pose of the robot is estimated using data from internal sensors.

Gazebo [2] was used as a 3D environment simulator. The motion execution is deterministic and error-free. Thus, the estimation of the robot's pose using odometry is accurate. The robot was controlled via the use of a tele-operation node, using the keyboard.

The position and the sensor data measured by the robot were visualized using RViz [3].

Task 1 Occupancy grid mapping algorithm:

The goal of the task is to implement mapping with known poses using odometry information and distance data from the recorded laser scan. A grid-map represents a 2D array of random variables with the probability *P(mi = occupied)* that a cell m_i is occupied by an object. The map can be estimated using the following expression:

$$m^* = argmax_m\left[P\left(m|u_{1,}z_1, u_{2,}z_2, \dots, u_{t-1,}z_{t-1}, u_{t,}z_t\right)\right] \quad (1)$$

Where u represents the control inputs to the robot and z represents the recorded measurements and t is the time for measurements.
Assuming that the individual cells are independent, the map probability can be reformed as:

$$P\left(m|u_{1,}z_1, \dots, u_{1,}z_t\right) = \prod_i P\left(m_i|u_{1,}z_1, \dots, z_{t-1}, u_{t,}z_t\right) \quad (2)$$

Assuming that the pose is known, then this estimation can be rewritten as:

$$P(m|z_{1..t}, X_{1..t}) \quad (3)$$

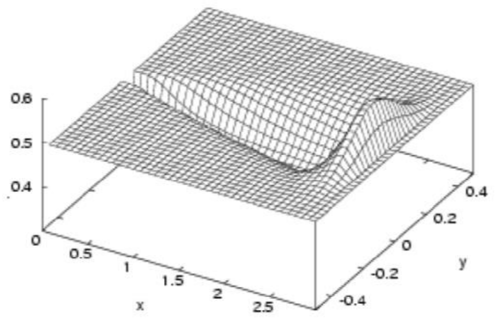
Using Markov assumption and a recursive Bayes Filter for this map estimation problem Given the probability of the map cell at time $t-1$ as well as the measurement z_t and the pose X_t at time t we can estimate the probability of the map cell at time t.
A log-odd representation of this estimation will avoid rounding and saturation problems and allow summation instead of multiplication

$$l(x) = log\left(\frac{P(x)}{1-P(x)}\right) \quad (4)$$

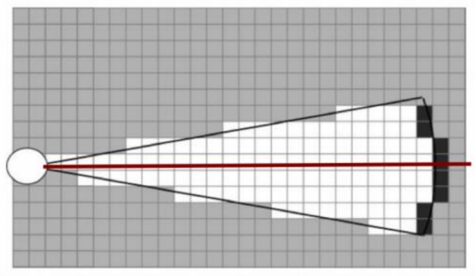
Then the map probability can be calculated from:

$$l(m_i|z_{1..t}, X_{1..t}) = l(m_i|z_{1..t-1}, X_{1..t-1}) + l(m_i|z_t, X_t) - l(m_i) \quad (5)$$

The left-hand side represents the log-odd ratio of the map cell m_i, while the first term on the right-hand side represents the log-odd ratio of the previous time step, the second term represents the inverse sensor model which represents the probability of cell m_i is being occupied see figure 1-a, and the last term represents the prior knowledge about the probability of that map cell, which is usually is set to zero.



(a)



(b)

Figure 1: a) Inverse Sensor Model, b) updated grid map using one range measurement [4]

The above algorithm defines three regions as shown in figure 1-b, occupied (black cells) with a high probability (the high mountain in figure 1-a), free (white cells) with low probability (the low valley in figure 1-a), and unknown (gray cells) based on the sensor measurement cone (defined by the open-angle and max. range). The probability of the unknown region is considered as 0.5. normally a smooth transition between the probabilities is considered due to the uncertainties in the sensor characteristics.

Implementation

To implement the above algorithm we assumed the following:
The high probability of the occupied cell is 0.8, the low probability of the free cell is 0.2 and the probability of the unknown cell is 0.5.

Bresenham algorithm [5] was used to create a tracking ray along the sensor beam, that supports the efficient map update.
The code can be found in the following attached document *(~src/mobile_robots_ public/tug_simple_turtle/src/simple_turtle.cpp*).
Using the function (*SimpleTurtle::odomCallback(const nav_msgs::Odometry::ConstPtr& msg)*) to provide the odometry robot positions and the function (*SimpleTurtle::laserScanCallback (const sensor_msgs::LaserScan::ConstPtr& msg)*) for laser measurement and algorithm computation.
In this implementation, we looped overall laser measurements at time step t and considered the sensor readings within the range [range_min, range_max], and discard any other reading. And checked every single cell within the sensor open-angle to compute its probability, then update its value using the function (*occupancy_grid_map_.updateCell(unsigned int cell_x, unsigned int cell_y, double value)*). This scenario is called incremental map updating.

Task 2: Most likely map

Due to the use of the incremental map updating and the uncertainties of the sensor readings some conflicting situations can happen as in figure 2. These situations can cause the map not to be represented correctly.

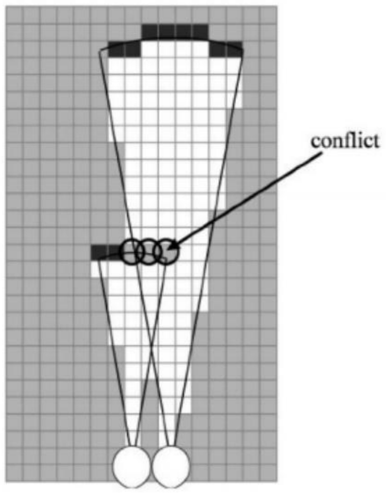


Figure 2: Conflicting grid-map after update with two successive laser scans [4]

To improve this situation, the probabilities of the obtained map given the sensor readings should be maximized by following a global approach where measurements are considered simultaneously. Equation 1 can then be reformulated as:

$$m^* = argmax_m \left[\frac{P(z_{1..t}|\mathrm{m}, X_{1..t})P(\mathrm{m}|X_{1..t})}{P(z_{1..t}|X_{1..t})}\right] \tag{6}$$

Assuming independent cells, and since the prior of the map is independent of the path of the robot and since the denominator is independent of the variable we maximize, then equation 6 can be reformulated using logarithmic representation as:

$$m^* = argmax_m[const + \sum_{t=1}^{T} \log P(z_t|\mathrm{m}, X_{1..t}) + l_o \sum_i m_i] \tag{7}$$

Neglecting the constant part and fixing the last term (the prior probability of m_i) to 0.5, we can then consider only the following hillclimbing approach on the probability function:

$$m^* = argmax_m[\sum_{t=1}^{T} \log P(z_t | X_{1..t}, m\ with\ m_i = k)] \tag{8}$$

Since this approach is quite hard to be solved computationally a closed-form solution was used [6] as follows:

$$m^* = argmax_m\left[\sum_{j=1}^{J} \alpha_j . \ln(m_i) + \beta_j . \ln(1 - m_i)\right] \tag{9}$$

Where

$$\alpha_j = \sum_{t=1}^{T}\sum_{n=1}^{N}\left(I(f(x_t, n, z_{t,n}) = j).(1 - \varsigma_{t,n})\right)$$

Corresponds to the number of times a beam that is not a maximum range beam ended in cell *j* (*hits(j)*) And

$$\beta_j = \sum_{t=1}^{T}\sum_{n=1}^{N}\left[\sum_{k=0}^{z_{t,n}-1} I(f(x_t, n, \mathrm{k}) = j)\right]$$

corresponds to the number of times a beam intercepted cell *j* without ending in it (*misses(j)*).
With x_t represents the pose at time *t*, $z_{t,n}$ represents the beam measurement *n* at time *t* and the value $\varsigma_{t,n} = \begin{cases} 1, & at\ max.range\ readings \\ 0, & when\ beam\ reflected\ by\ an\ object \end{cases}$

By setting

$$\frac{\partial m}{\partial m_j} = \frac{\alpha_j}{m_j} - \frac{\beta_j}{1 - m_j} = 0$$

We obtain:

$$m_j = \frac{\alpha_j}{\alpha_j + \beta_j}$$

Using that approach a cell m_j will be checked for how often has reflected measurement and how often it was intercepted.
Based on that we can then flip individual cells from occupied to free or vice versa if it improves the probability that all measurements are generated by the underlying map with the flipped cell.

Implementation

This approatch was implemented in the function (*SimpleTurtle::odomCallback(const nav_msgs::Odometry::ConstPtr& msg)*). To trigger the optimization part from task 2, the button 0 has to be pressed.

Results and discussion:

Figure 3 shows the environment that was used in this exercise built-in Gazebo.

Figure 4 shows the initial robot readings (no robot movement yet) and updated maps for the occupancy_grid_map approach (Figure 3-a) and the most_likely_map approach (Figure 3-a). Even though the first scan is not positioned correctly (due to a now fixed raise condition), this figure is still good for demonstration that the approach from task 2 improves the results.
While figure 5 shows the initial robot readings (no robot movement yet) and updated maps for the occupancy_grid_map approach (Figure 4-a) and the most_likely_map approach (Figure 4-a).

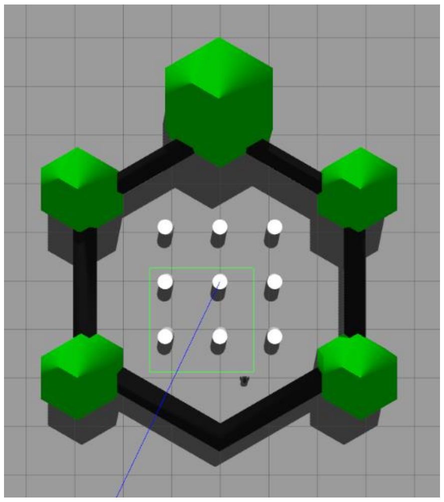


Figure 3: shows the environment that was used in this exercise built-in Gazebo

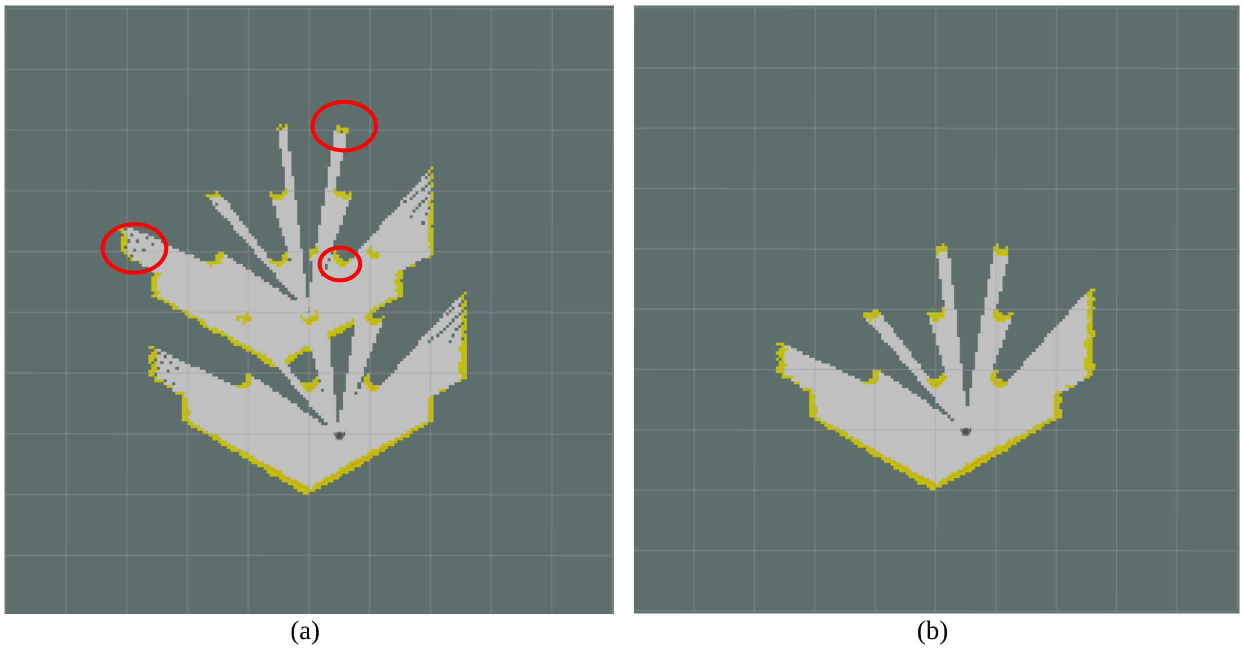


Figure 4: Initial robot position, a) occupancy_grid_map approach, b) the most_likely_map approach

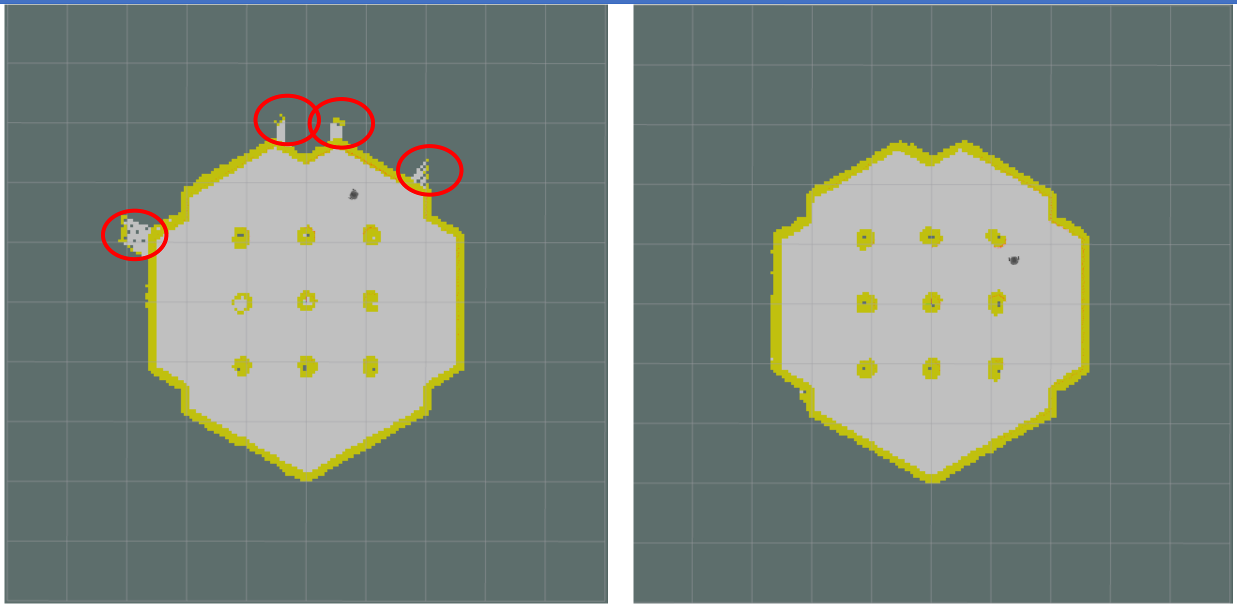


Figure 5: Navigation through the map, a) occupancy_grid_map approach, b) the most_likely_map approach

Comparing the areas depicted by the red circles it is clear that the most_likely_map approach has some improvements in the map estimation. But it’s hard to get compareable results, as there are very few measurement errors in the simulation.

As the probability for each cell in task 1 is calculated incremental, the more measurements there are, the consistent the map becomes in term of moving objects, as those moving objects have to be there for the same time (amount of measurements) as there are already measurements for the underlying cells. This is indeed desired as we want to build a static map.

References:

[1] TurtleBot3 user manual https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
[2] Gazebo Simulator - ROS Integration. http://wiki.ros.org/gazebo_ros_pkgs
[3] RViz. http://wiki.ros.org/rviz
[4] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press, 2005. ISBN: 0262201623.
[5] Bresenham algorithm https://de.wikipedia.org/wiki/Bresenham-Algorithmus
[6] http://ais.informatik.uni-freiburg.de/teaching/ss10/robotics/slides/08-occupancy-mapping.pdf