# 1. Localization

This part of the project involves the implementation of the Markov localization algorithm and successfully performing localization in a given environment with no information regarding the initial position of the robot in the environment.

## a) Environment

The environment used is a grid world with walls surrounding the edges as well as walls within the grid. This is chosen because it can be extended with minimal effort to work with occupancy grid based map representations.
The landmarks that the localization algorithms uses is the state (free or walled) of the neighbouring cells in the lateral and vertical directions. Using these landmarks, the algorithm can form an estimate of the possible locations in the map where the robot is possibly located at. This is done during the perception stage of localization.

## b) Robot & Sensing

The robot has limited sensing capabilities. It is able to detect whether it's first degree neighbors in the lateral and vertical direction are free cells or if they are walled cells. This can be thought of as primitive rangefinder. The sensor is accurate with it's readings with a probability of unity and there are multiple poses on the map where the robot can obtain the same sensor readings. This is handled by the localization algorithm during the perception stage. The posterior probabilities are normalized so that they sum to unity.

## c) Movement & Motion Model

The robot is able to move in four directions —- up, down, left and right on the map. The movement in a chosen direction succeeds with a probability of unity if the cell it wants to move to is free. If the cell is walled, the robot stays at its previous location. Thus, if the robot is at pose $(row_i, col_j)$ at time $t$, then it can be either at $(row_i, col_j)$ or either of the four neighbouring cells ($(row_{i-1}, col_j)$, $(row_{i+1}, col_j)$, $(row_i, col_{j-1})$, $(row_i, col_{j+1})$) based on the intended direction of movement at time $t + 1$.

## d) Localization Module

The localization module [1] is split into two subparts corresponding to the perception and the prediction stages of the localization algorithm. The algorithm used for localization is the Markov localization algorithm. The robot executes a set of actions in the environment. The actions of the robot correspond to the predication stage of the localization algorithm. The perception stages of the localization algorithm are interleaved between the actions. The various components are described in the following pages.

[1] The code for the localization module is available at https://github.com/say4n/mobile-robotics.

### Initial Belief

The initial belief of the localization algorithm is set to an uniform distribution over all the possible states that it can occupy. This is shown in figure 1. The hatched out cells represent walls in the environment. The numbers on the cells represent the belief of the localization algorithm. The red outlined cell is the actual position of the robot, this is unknown to the localization algorithm. Belief initialization is performed as follows.

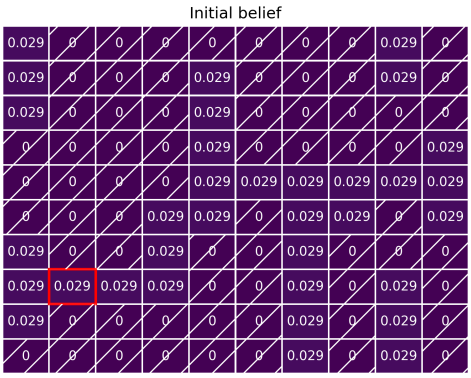


Figure 1: The localization algorithm starts with an uniform initial belief and then updates it's estimate via perception and prediction.

```
n_rows = env.get_n_rows()
n_cols = env.get_n_cols()
free_cells = env.get_free_cells()

belief = np.zeros((n_rows, n_cols))
for cell in free_cells:
    r, c = cell
    belief[r, c] = 1/len(free_cells)
```

### Perception Stage

During this phase, the robot uses its exteroceptive sensors to correct its previous estimate of its pose in the map. The uncertainty in the pose estimate decreases during this stage. This is implemented in the following code block.

```
# See phase.
obs = env.sense()
possible_locs = env.get_possible_locations_given_observation(obs)

# Computer posterior probabilities.
posteriors = np.zeros(len(possible_locs))
for i, (r, c) in enumerate(possible_locs):
    # Assuming perfect sensing.
    posteriors[i] = 1 * belief[r][c]

# Normalize probabilities.
# breakpoint()
posteriors = posteriors / np.sum(posteriors)

# Update beliefs.
belief = np.zeros(belief.shape)
for i, (r, c) in enumerate(possible_locs):
    belief[r][c] = posteriors[i]
```

### Prediction Stage

During this stage, the robot uses its proprioceptive sensors to estimate its pose. The uncertainty in the pose estimate increases during this stage as it the robot also performs a move action. This is implemented in the following code block.

```
# Act phase.
direction = action[1]
env.act(direction)

# Keep a copy of previous beliefs.
```

```
belief_copy = deepcopy(belief)

# Re-initialize beliefs.
belief = np.zeros(belief.shape)

# Update beliefs.
non_zero_probability_cells = np.argwhere(belief_copy > 0)
for r, c in non_zero_probability_cells:
    next_r, next_c = env.get_next_state(direction, (r, c))
    # Assuming single possible movement here.
    # Else, needs to be divided by number of possible next states.
    belief[next_r][next_c] += belief_copy[r][c]
```

### Localization in Action

A sample run of the localization algorithm is illustrated in figure 2. As can be observed from the figure, the localization algorithm is able to locate the robot in the grid world after just 5 steps. After this, the robot's pose predicted by the localization algorithm perfectly matches the actual pose of the robot in the environment.


(a) Perception. (b) Prediction after moving left. (c) Perception.

(d) Prediction after moving up. (e) Perception. (f) Prediction after moving down.

(g) Perception. (h) Prediction after moving right. (i) Perception.

Figure 2: A sampled execution of the markov localization algorithm. The box with the red outline is the actual location of the robot, unknown to the localization algorithm. The hatched out cells are walls. Numbers on the boxes represent beliefs for each location.

## 2. Mapping

This part of the project is implemented in ROS. The robot platform chosen is TurtleBot3. The robot is moved in the environment using teleoperation and a map is built from the data obtained through the LIDAR sensors on the robot. This data is transformed from the robot's frame of reference to the word frame and is visualized in an RViz window. It is demonstrated in the following video: https://youtu.be/EWWotzJBU4A. The pose of the robot is estimated by fusing the sensor data from the inertial measurement unit of the robot as well as it odometry sensors, these are then fed into an Extended Kalman filter to obtain the estimate of the pose.
The code for this part is also available in the github repository: https://github.com/say4n/mobile-robotics.