# Markov localization in the CiberRato simulation environment

João Santos, 76912[1] [0000-0003-0393-6456]

Department of Electronics, Telecommunications and Informatics
University of Aveiro, Portugal

**Abstract.** In this assignment, the author used the Markov localization to iteratively compute the most probable position of a robot in a simulated environment. A brief explanation on how it has accomplish the goal will be given, as well as the major limitations of the implemented methodologies. Finally, a visual representation of the expected result will also be provided.

**Keywords:** Peception · control · Markov · localization · Bayes'.

## 1 Introduction

When using map-based localization, the main goal is to supply the robotic agent with its most likely position within the map, provided that it is known a priori. There are multiple techniques on how to perform this task, one of those being the Markov localization, named after the Russian mathematician Andrey Andreyevich Markov.

When the robotic agent interacts with the surrounding environment, either by sensing or actuating, these actions are susceptible to noise and, therefore, inherent errors. These errors and their accumulation over time makes it almost impossible for the agent to know, with absolute certainty, its location.

What the Markov localization provides is a method to compute the probability (or belief) of the robot being in any given position, at any given time step, while simultaneously decreasing the accumulated error.

For a map-based localization, it is needed:

1. The initial belief. If this is unknown, a uniform distribution over the map should be used;
2. The known map. Is order to compute the most likely position, the agent must have information about the possible location that is can be;
3. Sensor data. The agent needs to continuously gather information about its surroundings (proximity, vision, etc.) to update its belief.
4. Control data. When the robot moves, this information must be used to also update the belief.
5. The sensor model. Basically, this is the probability of obtaining the correct sensor data ($z_t$) if the robot was at a given position $x_t$ on map $M$, i.e. $p(z_t|x_t, M)$.

6. The motion model. This model represents the probability of the robot being on position $x_t$, given the previous position $x_{t-1}$ and the control action requested ($u_t$), i.e $p(x_t|u_t, x_{t-1})$.

The reader must keep in mind that this algorithm relies on the Markov assumption, which claims that the location of a robot is solely affected by current information and the previous position. Given that, in reality, all previous positions and actions affect the robot at the current instant, Markov localization remains an approximation, albeit a very valuable one.

### 1.1 Environment

The developed agent was tested and evaluated using the *CiberRato simulation environment*. Two motors (left and right) are installed in the simulated robot and provide movement capabilities to it, accordingly to its motion model [3].

The simulated robot navigates a rectangular arena, which may be regarded as a bi-dimensional array of fixed-size cells.

Cells can be identified by their location, with the origin cell being placed at the bottom left corner. Within this map, thin walls are placed in order to define the maze.

### 1.2 Assumptions

A set of assumptions have been made in order to simplify the implementation. This set covers not only the specificities of the maps used but also the allowed/expected motions of the robot. They are:

- the maps are 7 rows by 14 columns
- the unit is the diameter of the robot
- cells are squares with side equal to two diameter
- inner walls have a thickness of 0.2 diameter
- walls on the border of the map have no thickness
- execution ends when the robot hits a wall
- motion noise does not exist
- sensor noise is Gaussian with mean zero and variance 0.1
- the robot motion is deterministic
- the robot only moves from left to right
- the Infra Red (IR) sensors detect the closest obstacle
- the IR sensors have a symmetric Field of View (FOV) of 60 degrees, centered on the normal direction

### 1.3 Motion Model

The motion model of the robot allows the agent to compute how much it has displaced from the original pose (position and orientation).

When the agent decides to move, it sends to the simulator the desired speed for each wheel. To simulate the inertia, the simulator applies an Infinite Impulse Response (IRR) filter that will, essentially, outputs the mean velocity from the desired and current ones, for each wheel. Notice that one of the assumptions is that there is no noise on the motion control system and, therefore, the model is deterministic.

Following the linear uniform movement equations, we can compute the position at time $t$ by knowing the position and orientation at the previous instant $(x_{t-1},\ y_{t-1},\ \theta_{t-1})$. $lin$ is the mean of the output velocities of both wheels.

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} + lin \begin{bmatrix} \cos(\theta_{t-1}) \\ \sin(\theta_{t-1}) \end{bmatrix}$$

On the assumption that the motion is always linear and from left to right, the orientation of the robot will always be zero radians. Yet, if that was not the case, the orientation at time $t$ would be given by

$$\theta_t = \theta_{t-1} + \frac{v_r - v_l}{D}$$

where $v_r$ and $v_l$ are, respectively, the velocities of the right and left wheels, and $D$ is the distance between them (or, by other words, the diameter of the robot).

### 1.4 Sensor Model

The sensor model is really a mathematical model that describes the relationships between sensor output and actual observed parameter values.

In the scope of this assignment, the IR sensor measures the distance to the closest obstacle and outputs this value as $\frac{1}{distance}$ to simulate the inverse proportionality expected on real sensors. This measurement is, however, altered by some noise.

This noise is expected to follow a normal (i. e., Gaussian) distribution that, as stated in [4], follows the Probability Density Function (PDF) on Definition 1. In it, $x$ denotes a specific event while $\mu$ and $\sigma^2$ are, respectively, the mean and variance.

**Definition 1.** $\mathcal{N}(x, \mu, \sigma^2) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\}$

### 1.5 Bayes' Filter

Given that the map is know to the agent and that the state space is finite we can, in a very direct way, implement the Bayes' filter to, iteratively, compute the the agent location or, more precisely, to compute the probability of the agent being on any given cell.

Algorithm 1 shows the general implementation of the Bayes' filter, as stated in [4]. In it, $bel(x_t)$ represents the belief at time $t$. $u_t$ and $z_t$ are, respectively, the control actions inputted and the measurements taken also at time $t$ and $\eta$ is a normalization factor.

**Algorithm 1:** Bayes' filter general algorithm.

```
  Input: $bel(x_{t-1}), u_t, z_t$
  Output: $bel(x_t)$
1 foreach $x_t$ do
2     $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx$
3     $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$
```

Note that for updating the belief at any given instant, only the prior estimate is need, which comes from the Markov assumption.

**Motion Update** On the scope of this assignment, since the agent is on a finite state space (it either is stationary or moves to the center of another cell) and can only move right, we can say that

$$\begin{aligned}\overline{bel}(x_t) &= \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx \\ &= \sum_x p(x_t|u_t, x)bel(x) \\ &= p(x_t|u_t, x_{t-1})bel(x_{t-1}) + p(x_t|u_t, x_t)bel(x_t)\end{aligned}$$

where $p(x_t|u_t, x_{t-1})$ and $p(x_t|u_t, x_t)$ denote, respectively, the probabilities of the agent has moved right and has stayed stationary, given the control action $u_t$.

Carried for this assignment and based on the motion model of the robot, it simply becomes that if the agent has stayed stationary ($u_t = [0, 0]$) then $p(x_t|u_t, x_{t-1}) = 0$ and $p(x_t|u_t, x_t) = 1$, and vice versa if the agent has moved one cell right ($u_t = [1, 0]$).

**Measurement Integration** It is in this step were the data gathered from all sensors is fused. Given that any given measure from each of the sensors follows the sensor model described on section 1.4, we can compute $bel(x_t) = \eta \cdot p(z_t|x_t) \cdot \overline{bel}(x_t)$ from algorithm 1 with algorithm 2, where $gt(x)$ represents the ground truth values in cell $x$. A more detailed explanation about the ground truth follows in section 2.2, but for now consider it to be the measurements expected to be gathered in each cell for a given sensor. This ground truth was computed a priori since the map is known to the agent.

**Algorithm 2:** Measurement integration algorithm.

```
  Input: $bel(x_{t-1}), z_t, gt$
  Output: $bel(x_t)$
1 foreach $x_t$ do
2     $product \leftarrow 1$
3     $\eta \leftarrow 0$
4     foreach $sensor$ do
5         $product = product \times \mathcal{N}(gt(x), z_t, 0.1)$
6     $\overline{bel}(x_t) = product \times bel(x_{t-1})$
7     $\eta = \eta + \overline{bel}(x_t)$
8 foreach $x_t$ do
9     $bel(x_t) = \eta^{-1}\overline{bel}(x_t)$
```

## 2 Implementation

As stated on section 1.2, the robot only moves in a straight line, from left to right. This poses several advantages, the first of which is that the walls are always perpendicular or parallel to the motion direction of the robot. To take advantage of this, the author choose to set the four IR distance sensors perpendicular to the border of the cell, i. e., to 0, 90, -90 and 180 degrees measured from the travelling direction.

Also, given the possible APIs, Python3 and the NumPy [1] library were chosen given their powerfulness, ease of use and familiarity. Also, the Matplotlib [2] library was used for visualization purposes.

Markov localization is computationally intensive since it computes the probability across all possible robot positions. This fact caused the issue where the agent would take a long time since it was called up to it being ready for the simulation to start. To solve this, the author decided that it should use an *offline* approach, this is, while the simulation is running, the agent is also computing in another *thread* the ground truth, while storing both sensor and control data in the main process. Only when the termination criteria is reached is that the agent computes the probability map at each time step (i. e., every time it reached the center of a cell) and outputs it to the required *localization.out* file.

### 2.1 Initial Probability Distribution

One of the requirements to allow us the application of the Bayes' filter is that the prior probability distribution is know. At time $t = 0$ the agent has no information about its location on the map. Considering that it can actually be on any of the cells, the distribution must be uniform and equal to $\frac{1}{\#cells}$ in each cell.

### 2.2 Ground Truth

Ground truth is information that is true or verifiable. It is derived from observation and measurement, which are the two main components of factual data.

When available, it allows us to compute $p(z_t|x_t)$, which is a must-have for the direct and discrete implementation of the Bayes' filter.

For this assignment, the main source for the ground truth is the map provided to the agent. By knowing the walls position on the map, the pose of each sensor on each cell and the FOV, the agent can precisely calculate the distance to the closest wall and return the measurement $\frac{1}{min(distances)}$.

In the chosen representation, a wall is defined by the coordinates of the two corners. These coordinates are in distance units (one robot diameter) to the reference frame of the map.

Since the inner walls have thickness, each map wall will produce two walls in this representation. Figs. 1 and 2 show, respectively, a map (pathFinderDefault_lab.xml) loaded into the *CiberRato Robot Simulation Environment* and the correspondent representation on the agent.

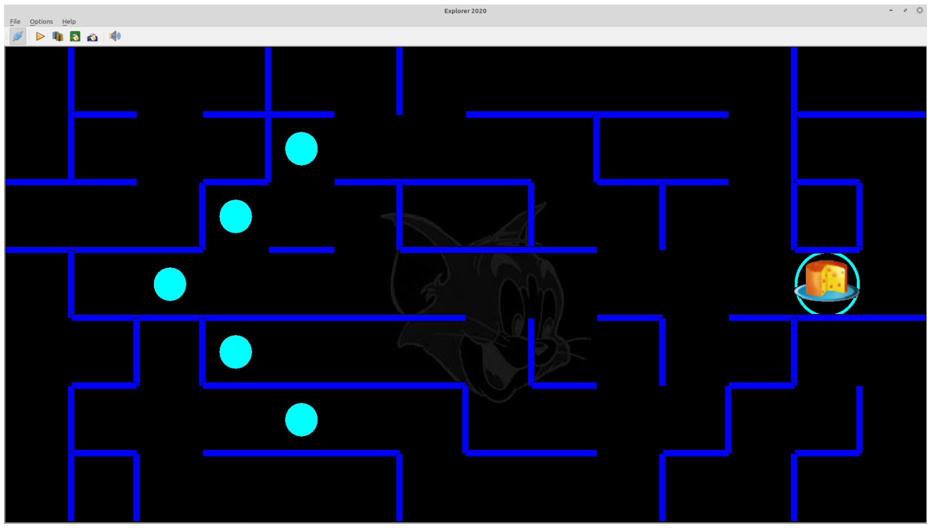


Fig. 1: Walls graphically displayed on the *CiberRato Robot Simulation Environment.*

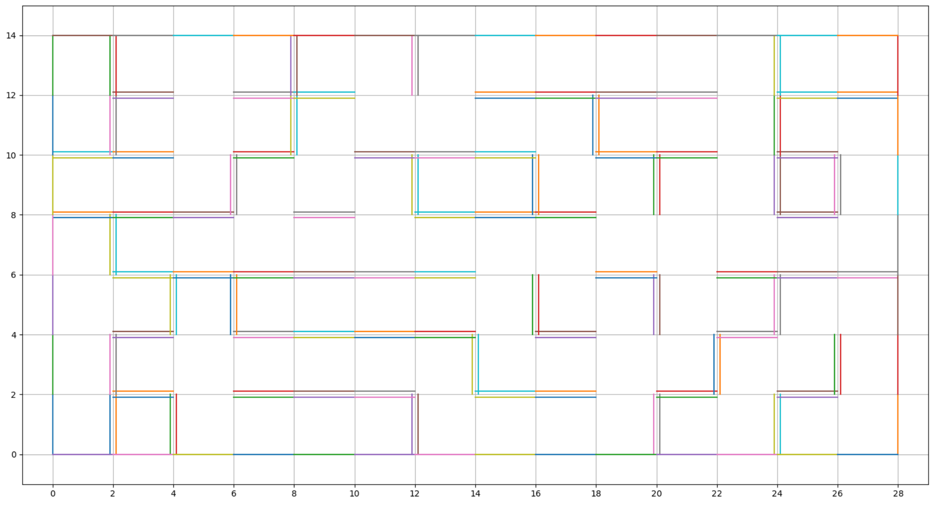
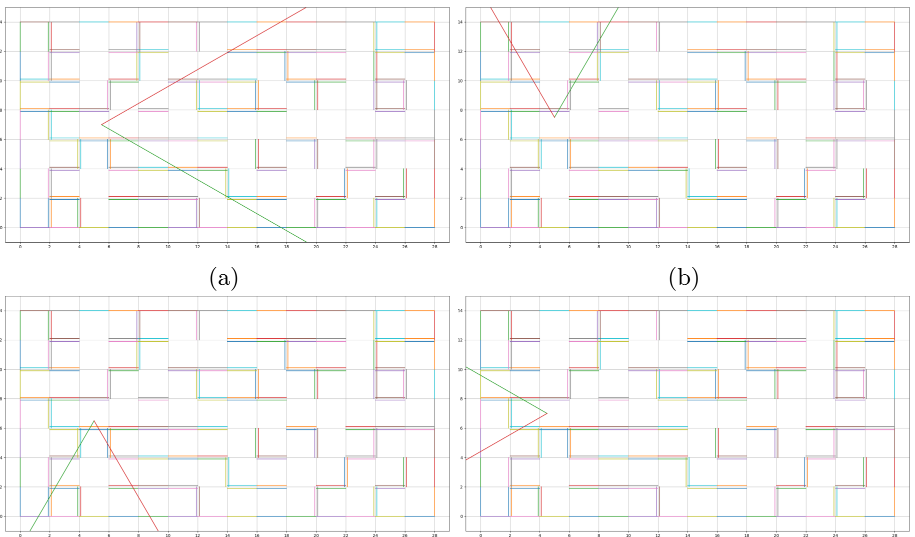


Fig. 2: Walls graphically displayed as they are stored by the agent.

For the implicit outer walls that delimit the map, the coordinates are trivial to compute as they are the combination of of the outer measures.

For all the explicit inner walls, we first obtain the indices of all walls by iterating over the provided *Map* structure and, then, convert the indices to the four corners, in map coordinates.

Having these coordinates, and using some geometry and vector algebra, it is possible to compute, for all sensors in all cells, the closest distance to any of the walls in the map, that lay inside the FOV of a given sensor. Notice that for this ground truth to by reliable, the sensors had to be placed in the correct position (distancing half a diameter from the center of the cell) and orientation inside each cell. On Fig. 3 a more visual explanation on how this was accomplished is provided.



(a) (b) (c) (d)

Fig. 3: Sensor pose and FOV for cell (3, 2). Sensors are at (a) 0º, (b) 90º, (c) -90º and (d) 180º.

### 2.3 Markov Localization

With the agent being in possession of the ground truth it can now start the iterative process where the initial belief is successively updated with new information (see Fig. 4).

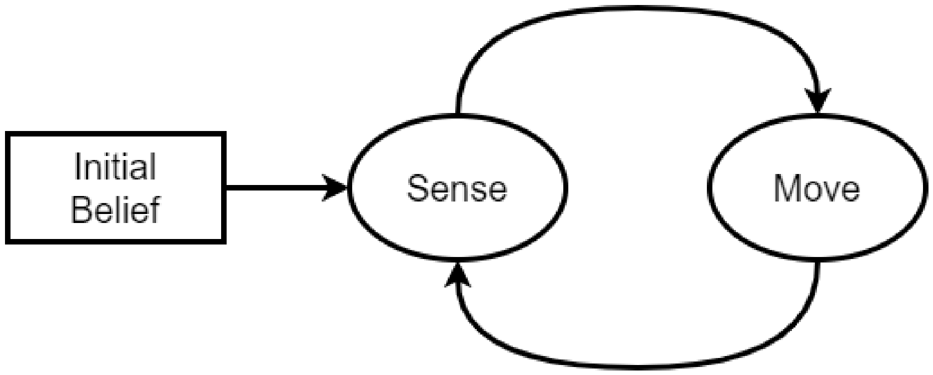


Fig. 4: Flowchart of the iterative localization procedure (Markov localization).

The agent is continually updating its displacement since the staring of the simulation up until the present moment. This is necessary so that the agent knows when it has arrived to a new cell (actually, the center of the cell). The agent does, in fact, know how much it has moved because the noise on the motors is null. If it was not, this would only be a prediction.

At the start of the simulation, the robot is stationary and, so, the displacement is null. The agent will use this control action to update the initial belief but, as demonstrated on section 1.5, when $u_t = [0, 0]$ there are no changes in the probabilities distribution.

Immediately after this, the IR sensors measures are integrated using algorithm 2 and the agent waits until it reaches a new cell to iterate the process again. This time, given that the motion is not null ($u_t = [1, 0]$) the Motion Update will cause a shift on the probability distribution.

### 2.4 Known Limitations

There are some known limitations in the present work. These are stated bellow:

1. The Motion Update procedure is only developed for stationary of one cell right motion control actions. This is on purpose because i) other motions are outside the scope of this assignment and ii) it limits the amount of code needed to compute the updated belief.
2. The computation of the ground truth is relatively slow. Because of this, the algorithm runs on another *thread* and the actual updating of the beliefs, at each time step, is only done after the simulation ended (*offline* approach). At this stage, the agent should have finish gathering the measurements and control actions at each cell of the simulation.
3. The agent uses the collision detection has a termination criteria. This works well is the scope of this assignment given the single Degree of Freedom (DOF) of the motion, but would be counterproductive in a real world scenario.

## 3 Results

On Fig. 5, a demonstration of the localization process is presented. At the starting state (Fig. 5a) the agent is completely unaware of its position on the map (again using the same map as on section 2.2 for an easier understanding). Then, as it senses the surroundings and moves, both for the first time, the pool of possible locations shrinks (see Fig. 5b), becoming just two possibilities with equal chances within the next two iterations (see Figs. 5c and 5d). In posterior evaluations, the agent is completely sure of its position (see trough Figs. 5e to 5m) [1].

[1] Video available on https://youtu.be/gf-PzjpjPEI

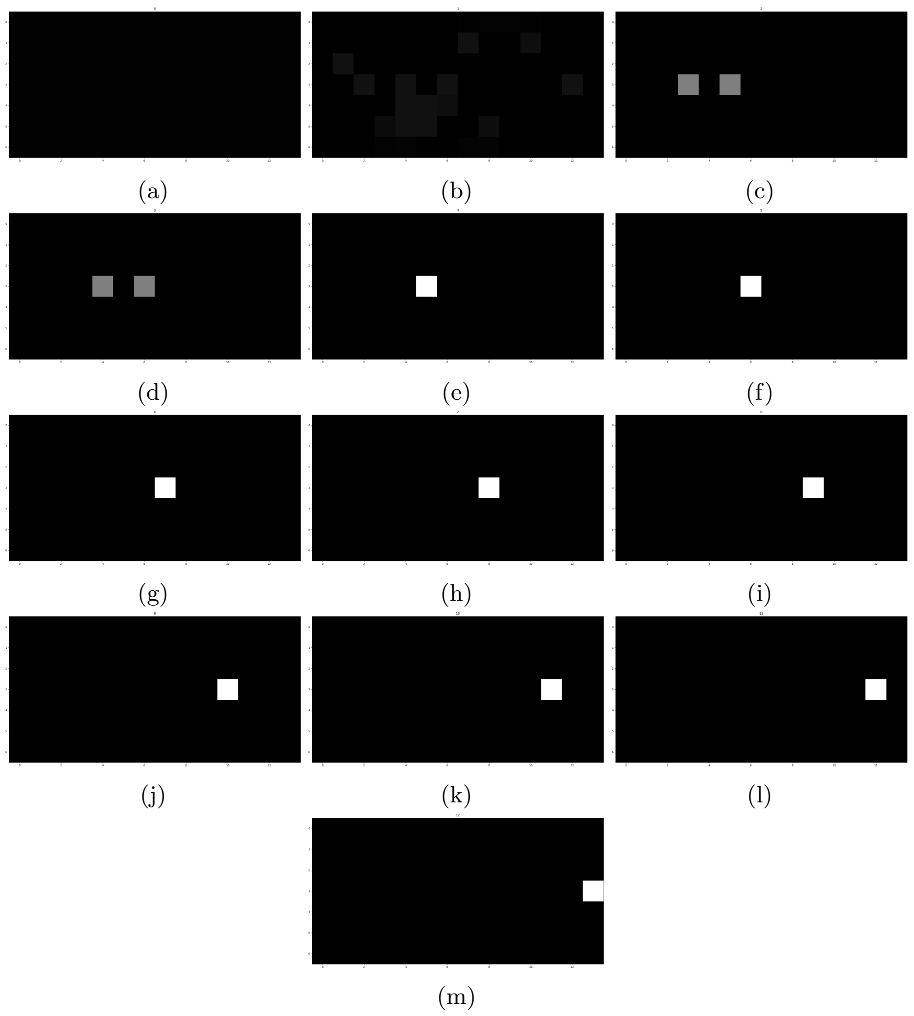


Fig. 5: Visual representation of the agent localization belief updating process.

## References

1. Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature **585**(7825), 357–362 (Sep 2020). https://doi.org/10.1038/s41586-020-2649-2, https://doi.org/10.1038/s41586-020-2649-2
2. Hunter, J.D.: Matplotlib: A 2d graphics environment. Computing in Science & Engineering **9**(3), 90–95 (2007). https://doi.org/10.1109/MCSE.2007.55
3. Lau, N.: Assignment 1: Markov localization in the ciberrato simulation environment (2021)
4. Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics. MIT Press, Cambridge, Mass. (2005)