Universidade de Aveiro

Robótica Móvel

Assignment no. 1 - April 2023

Robot Probabilistic Localization using Redundant Landmarks

Mestrado em Robótica e Sistemas Inteligentes

1 Main objective

Localization of a robot in environments with multiple landmarks along a path that starts at a reference point and passes through all the intermediate landmarks (beacons) until stopping at the last one.

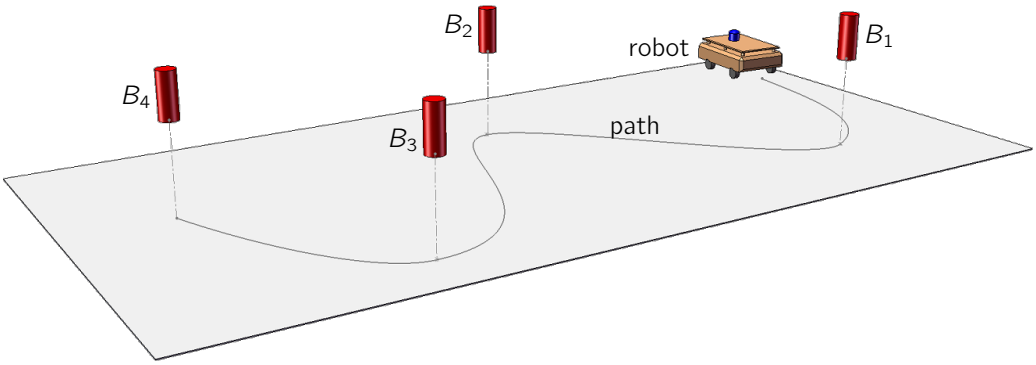

Figure 1: *Illustration of a scenario where a robot follows a trajectory that passes through several landmarks and performs localization with a redundant constellation of landmarks. The 4 landmarks in the figure are merely illustrative: the solution to be developed must handle any number of landmarks greater than 2.*

The landmarks (or beacons) have known positions, and it is assumed that the robot has a sensor that allows it to identify and distinguish the beacons and measure their distance and orientation in relation to its current position, but with an associated uncertainty. However, even if, in general, there is redundancy of beacons (more than strictly necessary), there are situations in which one or more beacons may not be detectable, in which case the necessary adaptations must be made to use only known information. It is intended that the localization is the best possible and therefore it is recommended to use the maximum amount of information available to do so, although it is possible to do localization using only part of the total information at each moment.

While other tools are acceptable, the use of Extended Kalman Filters (EKF) is expected using appropriate motion and observation models. The main result of the simulation should be the successive estimated robot locations along the trajectory.

2 Description

Localization can be done deterministically using the classic approaches of triangulation and/or trilateration, or with probabilistic models that take into account the kinematic model and the predicted movement of the robot, as well as the observation model of the sensors. In either case, obviously, the readings of the distance and/or direction data measured by the sensors on board are necessary. To reduce localization errors due to measurement uncertainty, probabilistic solutions such as Extended Kalman Filters are recommended under certain conditions (such as the requirement of the Gaussian nature of the variables involved).

The aim is to have the best possible localization and, hence, the more beacons used in each observation, the more robust is the solution expected to be, since it takes advantage of the redundancy that may exist at any time. In the exercises done in classes, examples were always made with only two visible beacons (although they were not always the same). In this work, we intend to use as many beacons as possible at each instant to strengthen the confidence of the location, but there is a risk that one or more beacons may fail to be detected! In that situation, the program has to deal well with this information, especially in the observation models and, more specifically, in the innovation calculation, either in the innovation covariance matrix or in the Kalman gain matrix, which both depend on the Jacobian of the

observation function $h()$. This Jacobian of function $h()$ will have variable dimensions according to the number of measurements (beacons) used at each instant.

2.1 Generic equations for the Kalman Filter

In the following expressions, and adopting the ISO 80000-2 directives, vectors are represented by a bold italic lower case letter (like $\boldsymbol{u}_k$) and matrices are represented by a bold italic upper case letter (like $\boldsymbol{P}_k$). Also, in general, predicted quantities are represented by a overlay bar (like $\bar{\boldsymbol{u}}_{k+1}$) and updated estimated quantities are represented with a overlay hat (like $\hat{\boldsymbol{u}}_{k+1}$).

Prediction with the process model:

$$\begin{cases} \bar{\boldsymbol{x}}_{k+1} = f\left(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right) \\ \bar{\boldsymbol{P}}_{k+1} = \boldsymbol{J}_{f_x}\left(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right) \boldsymbol{P}_k \boldsymbol{J}_{f_x}^{\mathsf{T}}\left(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right) + \\ \qquad + \boldsymbol{J}_{f_w}\left(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right) \boldsymbol{Q}_k J_{f_w}^{\mathsf{T}}\left(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right) \end{cases} \tag{1}$$

where

- $f()$ is the function of the process:
  - it allows obtaining the next state depending on the current state ($\hat{\boldsymbol{x}}_k$) and the control variables ($\boldsymbol{u}_k$), which are the linear and angular velocities.
- $\boldsymbol{P}$ is the process state covariance matrix
- $\boldsymbol{Q}$ is the process error covariance matrix
- $\boldsymbol{J}_{f_x}\left(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right)$ is the jacobian of function f in relation to $\boldsymbol{x}$;
- $\boldsymbol{J}_{f_w}\left(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right)$ is the jacobian of function f in relatin to $\boldsymbol{w}$;
  - both evaluated at $\left(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right)$

In all the previous expressions, as the process noise (control variables) is generally assumed to be zero-mean Gaussian, $\boldsymbol{w}_k \sim \mathcal{N}(0, Q)$, we normally use $\boldsymbol{w}_k = 0$. Some literature considers that the control input refers to the next moment and, in this case, $\boldsymbol{u}_{k+1}$ can be used instead of $\boldsymbol{u}_k$, but in any case it is the control value applied to estimate the next state.

Update using observation:

$$\begin{cases} \hat{\boldsymbol{x}}_{k+1} = \bar{\boldsymbol{x}}_{k+1} + \boldsymbol{K}\left(\boldsymbol{z}_{k+1} - h\left(\bar{\boldsymbol{x}}_{k+1}\right)\right) \\ \boldsymbol{P}_{k+1} = \bar{\boldsymbol{P}}_{k+1} - \boldsymbol{K}\boldsymbol{S}\boldsymbol{K}^{\mathsf{T}} \end{cases} \tag{2}$$

where

- $\boldsymbol{z}_{k+1}$ is the vector of all sensorial readings obtained at a given time.
- $\boldsymbol{z}_{k+1} - h\left(\bar{\boldsymbol{x}}_{k+1}\right)$ is the innovation
- $\boldsymbol{S}$ is innovation covariance given by:
  - $\boldsymbol{S} = \boldsymbol{J}_h\left(\bar{\boldsymbol{x}}_{k+1}\right) \bar{\boldsymbol{P}}_{k+1} \boldsymbol{J}_h^{\mathsf{T}}\left(\bar{\boldsymbol{x}}_{k+1}\right) + \boldsymbol{R}$
  - $\boldsymbol{R}$ is the covariance matrix of the observations (sensory measurements)
- $\boldsymbol{K}$ is the Kalman gain given by:
  - $\boldsymbol{K} = \bar{\boldsymbol{P}}_{k+1} \boldsymbol{J}_h^{\mathsf{T}}\left(\bar{\boldsymbol{x}}_{k+1}\right) \boldsymbol{S}^{-1}$
- where $\boldsymbol{J}_h\left(\bar{\boldsymbol{x}}_{k+1}\right)$
  - is the jacobian of function $h()$ in relation to $\boldsymbol{x}$ evaluated at $\bar{\boldsymbol{x}}_{k+1}$.

2.2 Environment and trajectory

The process scenario is a rectangular geometry environment where N passing points are defined that also correspond to the position of landmarks or beacons B_n $(n = 1, 2, \cdots, N)$ that can be detected with a given measurement uncertainty.

The robot must try to execute a trajectory that starts at the initial point $(0, 0, 0)$ and passes through all the beacons, stopping at the last one in the list.

To define the motion model, more particularly the control variables, the trajectory must be specified in such a way that the velocity control parameters (linear and angular) are defined at regular intervals in each section between beacons, thus defining the points where it is expected to update the robot control signals (and if there are new observations from the sensors).

In fact, whenever there are new observations, the localization estimate can be updated. Thus, the velocity control update points (angular and linear) are conditioned by the availability rate of the sensors and the desired velocity.

To simplify this estimation of trajectory points, and since this is a kinematic simulation and not a reactive motion adapted during execution, for this calculation, it is assumed that the robot moves along a linear segment at constant velocity. The actual positions on the desired final trajectory may be slightly different, but this is not decisive and is described below.

As an example, if the sensors have an update interval of 1 s, and the average linear velocity of the robot is intended to be 5 m/s then, in a segment of 20 m between two beacons, $\frac{20}{5\times1} = 4$ intermediate points for updating sensor readings and control inputs (velocities) will be defined. These points are in addition to the points of the beacons themselves and the starting point that make up the total trajectory.

To understand this better, follows another example: between each pair of beacons (and between the starting point and the first beacon), there must be points where new sensor readings will be taken and a new control will be imposed, approximately every $\Delta d = \Delta t \times V$ (in m) in the linear trajectory, where Δt is the interval between successive sensor readings (in s) and V is the linear velocity (m/s). Follows a numerical example: consider two consecutive beacons $B_1 = \begin{bmatrix}10 & 10\end{bmatrix}$ and $B_2 = \begin{bmatrix}40 & 50\end{bmatrix}$, the linear distance between them is 50 meters ($\left\|\begin{bmatrix}40-10 & 50-10\end{bmatrix}\right\| = 50$); with a linear velocity of $V = 5\,\mathrm{m/s}$ and a sensor update interval $\Delta t = 2\,\mathrm{s}$, $\frac{50}{5\times2} = 5$ intervals, that is, 4 intermediate points, as shown in Figure 2.

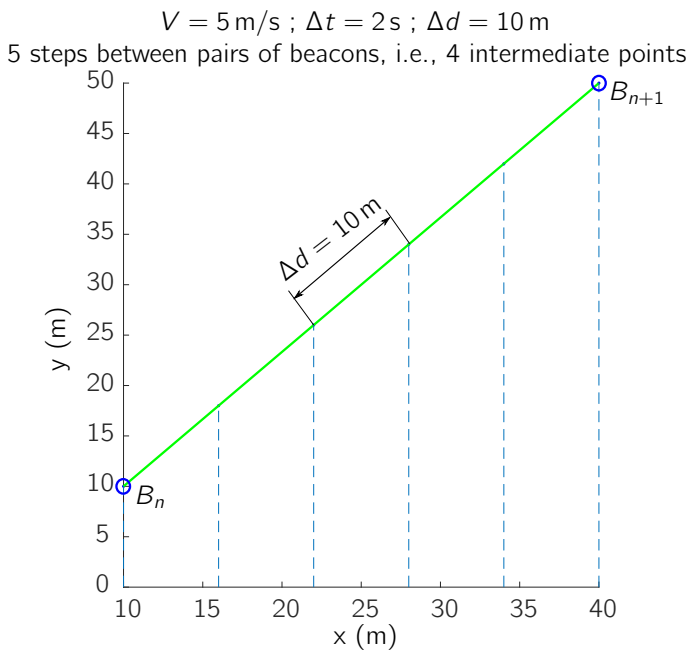

Figure 2: *Illustration of the calculation of the number of points to use in the trajectory segment between successive beacons. The linear section of the trajectory was divided into segments of* 10 m*, which is the value for the desired velocity and the sampling time of the sensors.*

2.3 Sensor measurements

The measurements (observations) of the sensors will be simulated using a provided function called: `BeaconDetection()`. This function returns the distance and direction readings in relation to the robot's posture that is passed as an argument. Measurements are uncertain and under some conditions are not

available for the following possible reasons: too close or too far from the beacon, or even due to sporadic failures. In these situations the beacon measurements come as `NaN` (*not-a-number*). In addition to the measurements, the returned data also has information about the beacons, such as their coordinates in the world. The general description of `BeaconDetection()` is as follows:

```
function [B]=BeaconDetection(N,P,obsNoise)
%
% Function that does two main tasks:
%   Creates a set of up to N beacons in a fixed arrangement.
%   Returns informations of those beacons as described.
%
% INPUTS:
%   N - number of beacons to create/use (N>3) but large values may not be respected
%   P - current estimated position (x,y,a). (0,0,0) if absent.
%   obsNoise - observation noise [range, heading]. If not passed, use a default value
%
% OUTPUT
%   B - array of structures with data from beacons:
%     B.X - real beacon X position (fixed and known)
%     B.Y - real beacon Y position (fixed and known)
%     B.d - measured distance (with uncertainty)
%     B.a - measured angle (with uncertainty)
%     B.dn - sigma in B.d (either the passed in obsNoise or a default)
%     B.an - sigma in B.a (either the passed in obsNoise or a default)
%
% NOTES:
%   Occasionally B.d and B.a are not available (too close or to far from beacon)
%   and in those cases their value is NaN and sensorial data is not available.
```

The robot must start at point $(0, 0, 0)$, that is, at the origin and oriented horizontally to the right. The planned trajectory must be smooth and pass through all points B_n and end at the last one. Of the various possibilities for defining such a trajectory, the Hermite cubic polynomial interpolation should be adopted, which, in MATLAB, is obtained with the `pchip()` function, that is, Piecewise Cubic Hermite Interpolating Polynomial. The control and observation points are defined along this trajectory.

These points must be evenly spaced in x and the respective values in y are obtained by applying exactly the aforementioned `pchip()` function. This does not always result in uniform spacing along the path because of the smooth interpolation adopted, but it is an approximate process and greatly simplifies the definition of these control points! Figure 3 illustrates a case with 6 beacons in a rectangular space of 300×160 where on the right the trajectory with Hermite interpolation is shown and the marking of the different points along the trajectory to try to simulate a regularity throughout the movement.

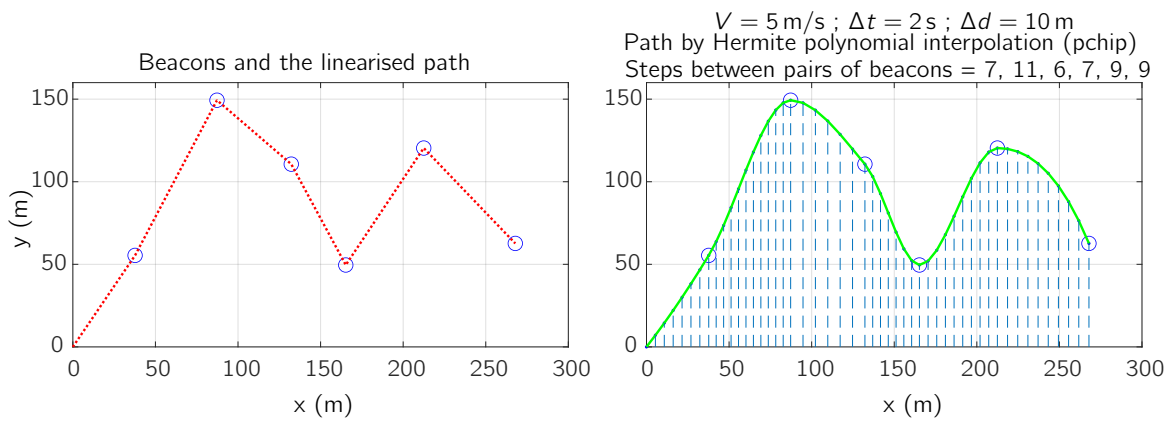

Figure 3: *Trajectory that starts at the origin and passes through 6 beacons. On the right, the illustration of the intervals that define the control and observation points along the interpolated trajectory.*

The interpolated trajectory represented on the right side of Figure 3 is considered the ideal planned trajectory for the case of 6 beacons and, therefore, will be defined as the *ground truth* for all comparisons and analyses that will be carried out. Other trajectories of a similar nature will be defined for other values of the number of beacons.

2.4 Motion models and physical requirements

The planning and localization calculation can be done for a general robot case where the control inputs are linear velocity and angular velocity, and this will be the recommended methodology in the general case.

However, the defined trajectories may for some reason require excessive mechanical requirements. For example, a certain linear or angular velocity of the robot may require excessive angular velocity values for a traction wheel or steering limits for a directional wheel.

Thus, an analysis of the angular velocities of the traction wheels and the values of the steering wheel of a tricycle for two robots with different kinematic models along the path to be executed must be carried out, namely:

- Differential drive
- Tricycle

These kinematic models are different but can be parametrized relatively easily: they all have a parameter L which is the distance that relates the relative position of the wheels, as illustrated in Figure 4, and the tricycle has a second parameter which is the direction of the front wheel, and which is called α.

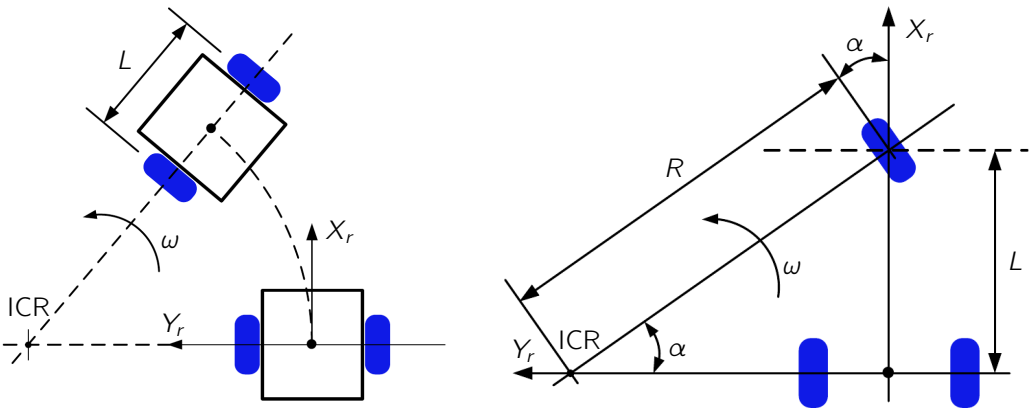

Figure 4: *Kinematic models of robots to use: differential and tricycle.*

Thus, having established the trajectory with the geometric and kinematic requirements calculated with the methodology presented above, the consequences for each of the kinematic models in the velocities of the traction wheels and in the values of the angles of the steering wheel must be verified.

In the case of the tricycle, it is assumed that the traction is done on the back wheel, more specifically on the "virtual" wheel equivalent to the two wheels that will be connected by a mechanical differential. Therefore, when we mention the velocity of the back wheel of the tricycle, we are assuming the "virtual" wheel equivalent to an "average" of the two back wheels of the tricycle.

More particularly, the physical quantities that must be determined for each of the kinematic models are the following:

- Differential drive - ω_R and ω_L (right and left wheels)
- Tricycle - ω_T and α (rear wheel velocity and front wheel angular position)

It is assumed that the robots have the same distance value L and their respective wheels are assumed to have the same radius r. Therefore, for the physical parameters L and r of each robot, and for each point of the calculated trajectory, it is necessary to obtain the values of the angular velocities of all driving wheels and the angle of the steering wheel on the tricycle.

3 Student program

3.1 Summary

Students must create a main program that is a MATLAB function that accepts the following parameters (where default values are indicated if omitted):

- N - number of beacons (4)
- Dt - sensors sampling time interval (1 s)
- r - radius of the robots wheels (0.15 m)
- L - separation/distance of the wheels according to the kinematic model (1 m)
- Vn - uncertainty (sigma) in the linear velocity to be imposed on the robot (*input*) (0.1 m/s)
- Wn - uncertainty (sigma) in the angular velocity to be imposed on the robot (*input*) (0.1 m/s)

The code developed by the student must be in a single main file, including any auxiliary functions. This file must be named `rm1_nnnnnn.m` with `nnnnnn` the student's number. Functions extracted from the Internet, form the classes or other sources must be provided separately and with an indication in the header of each one of the source from which they were used. These functions must be in a separate folder called "lib" which will then be part of the MATLAB path.

3.2 Procedure steps

The main stages of the procedure, which correspond to the tasks to be carried out by the student's program, are the following:

1. Calculate all points on the trajectory with the passed parameters and the coordinates of the beacons obtained with a simple call to the function `B=BeaconDetection(N)`. This trajectory will contain all the points where the robot is expected to pass with a certain pose: (x, y, θ), where θ should be defined as the direction angle to the next midpoint of the trajectory (not necessarily the next beacon).
2. Calculate the linear and angular velocity to apply to the robot at each point of the trajectory. These velocities should be such that, in theory, they take the robot to the next pose during the time interval Δt.
3. For each point (pose) of the trajectory, apply the Kalman filter to estimate the next point (location). For that, it is also necessary to call the `BeaconDetection()` function with the appropriate parameters to obtain the measurements. This step is the most important and the most laborious, and it is the core of probabilistic localization.
4. During the previous process, save the successive poses (estimated locations) and, at the end, write a file with that data with one pose per line and the 3 fields separated by a comma. The file must be called `loc_nnnnnn.txt` where `nnnnnn` is the student number.
5. For each of the kinematic models of robots, calculate the angular velocities of the wheels and/or the direction of the tricycle wheel along the path. There should be distinct files with one entry per line with the fields separated by commas, more specifically according to the following where the name of the file to be created and the format of each line in each robot are indicated, and where `nnnnnn` is the student's number:

Differential drive	`DD_nnnnnnn.txt`	ω_R, ω_L
Tricycle	`TRI_nnnnnnn.txt`	ω_T, α

   Or, in compact form, the student's `rm1_nnnnnn.m` program:

   - must be a MATLAB function
   - needs to use the provided function `BeaconDetection()`
   - generates 3 files during its execution:
     - `loc_nnnnnn.txt`
     - `DD_nnnnnnn.txt`
     - `TRI_nnnnnnn.txt`

4 Material to submit for evaluation

- Program in MATLAB developed by the student in function format as described earlier, and which is the program to execute. The file name should be `rm1_nnnnnn.m` where `nnnnnn` is the student's number.

- All external functions used, not developed by the student, but which are essential to the execution of the main program. It is recommended that these functions are in a separate folder called "lib" which will be integrated into the MATLAB path for executing the main program.
- A report of a maximum of 6 pages (including the cover page) written in LaTeX with the `documentlass` of `report` or similar. This report must include a description of the methodology with the mathematical formalisms used as well as the figures and graphs needed to illustrate the results. It should also include a conclusion and analysis of those same results. A way to enrich this analysis could be an objective statistical comparison of the results that would be obtained by classic triangulation/trilateration techniques and the probabilistic technique used in this work.

The following evaluation elements will be considered:

- Conformity of the trajectories as requested in assignment
- Quality of localization results
- The report: accuracy of language and mathematical formalisms, richness and clarity of figures, typographic quality (LaTeX highly recommended). The conclusions and critical analysis of results will also be considered important.
- Results of the calculation of the physical requirements on the models of the indicated robots.

The code developed by the students will be checked for plagiarism using the MOSS system.