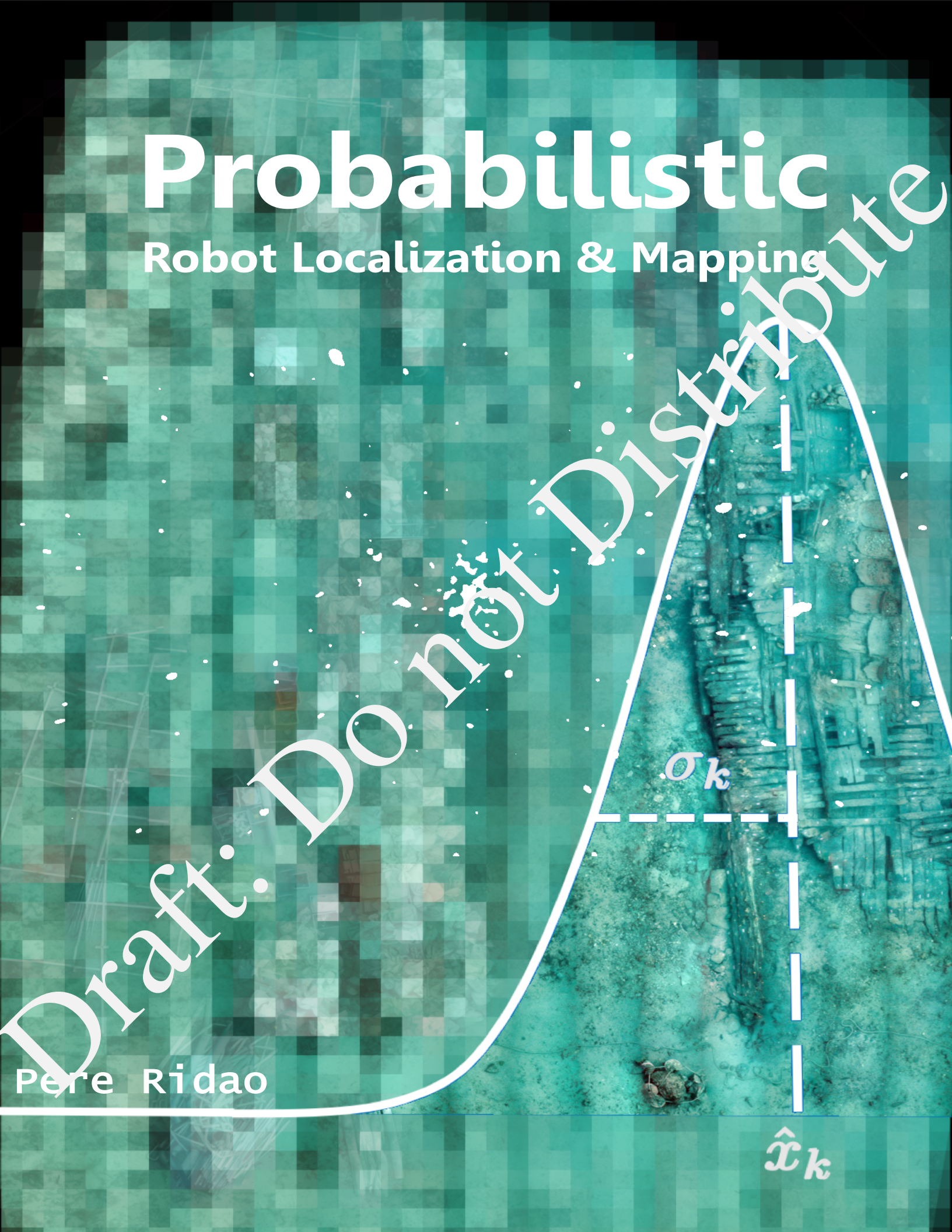
Probabilistic

Robot Localization & Mapping



Pere Ridao

MIR – EMJMD IN MARINE AND MARITIME INTELLIGENT ROBOTICS (MIR) ERASMUS MUNDUS JOINT MASTER'S DEGREE

Draft Lecture notes and proposed exercises of the Marine Localization & Mapping Course. DO NOT DISTRIBUTE

Release, March 2023

List of Algorithms

List of Figures

List of Tables

Contents

1. Differential Drive Mobile Robot Labs

hyperref

1.1 Lab 2: Grid Localization

In this exercises we will implement a Grid Localization method for the Differential Drive Mobile robot using as input the robot velocity obtained form its dead reckoning and using distance measurements with respect to an a priori known map of two dimensional Cartesian features.

1.2 Cloning the PR_LAB2 Repository

In this section, we'll guide you through the process of cloning the "PR_LAB2" repository from GitHub onto your computer. This repository contains materials related to a laboratory exercise or project. Make sure you have Git installed on your computer before proceeding.
Follow these steps to clone the repository:

1. **Open a Terminal or Command Prompt**: Depending on your operating system (Windows, macOS, or Linux), open a terminal or command prompt.
2. **Navigate to the Desired Directory**: Use the 'cd' command to navigate to the directory where you want to store the repository. For example:
```
$ cd Documents
```
3. **Clone the Repository**: Run the following command to clone the "PR_LAB1" repository from GitHub. Replace '<your-username>' with your GitHub username:
```
$ git clone https://github.com/IFROS-MIRS/PR_LAB2.git
```
If you have GitHub authentication set up, you won't need to provide credentials. If not, GitHub may prompt you to enter your username and password.
4. **Verify Cloning**: Once the cloning process is complete, navigate to the cloned repository's directory using the 'cd' command:
```
$ cd PR_LAB2
```

You should now be inside the cloned repository. You can check the contents to verify that everything has been cloned correctly.

That's it! You've successfully cloned the "PR_LAB2" repository onto your computer. You can now work on the contents of the repository or use them for your laboratory exercise or project.

1.2.1 Updating with your solution of LAB1

Once you have cloned the repository, you need to update the python files that you already programmed during lab1. This include the files:

1. DifferentialDriveSimulatedRobot.py
2. DR_3DOFDifferentialDrive.py
3. Pose3D.py

Next you have to update certain variables within your files:

1. self.yaw_reading_frequency=1 (DifferentialDriveSimulatedRobot.py)
2. self.visualizationInterval=1 (SimulatedRobot.py)
3. self.vehicleIcon=VehicleIcon('DifferentialDrive.png', scale=1, rotation=90) (SimulatedRobot.py)

Now we are ready to solve the first part of the lab.

1.2.2 GL: Part I (Update)

In this part of the exercise we are going to implement the *Update* step of the filter. To do it we need to follow the following steps:

1. **Reading the distance towards the landmarks**: A map named *M2D* with 3 Cartesian landmarks is created in the *main.py*. To detect the distances of the robot towards this landmarks, it is necessary to:
   (a) Program the method named *ReadRanges()* within the class *DifferentialDriveSimulatedRobot*.
   (b) Program the method *GetMeasurements()* of the *GLDifferentialDrive* which will call the previous one to get the distance to the nearby features.
2. **Check the *HF* class**: Read the documentation of the *HF* class and check its implementation. This is a base class already programmed.
3. **Check the *GL* class**: Read the documentation of the *GL* class and check its implementation. This class is incomplete and needs to be completed by the student. In particular, you need to complete the following methods:
   (a) ***LocalizationLoop***: Given an initial position histogram p_0 and the control input *usk* used in the *fs* method of the *DifferentialDriveSimulatedRobot* class, this method reads the robot displacement u_k using the method *GetInput* inherited from the *DR_3DOFDifferentialDrive*, reads the distance to the landmarks using the method *GetMeasurements* and calls iteratively the *Localize* passing the previous robot position histogram p_{k_1}, the robot displacment u_k and the measurements z_k. All this process is repreated $kSteps$ steps.
   (b) ***Localize***: This is a single step localization. If performs first a prediction calling the *Prediction* method inherited from the *HF* class, and then performs an update of the filter calling the *Update* method also inherited from the *HF* class. It is worth noting that the *Update* method uses the *MeasurementProbability* method to compute the measurement histogram to be used during the Bayes Rule. You need to provide an implementation of the *MeasurementProbability* in the *GL_3DOFDifferentialDrive* class corresponding to the respective pure virtual method in the *HF* class.

(c) ***MeasurementProbability***: To create the measurement histogram first you need to:

i. Get familiarized with the class Histogram2D. The file *Histogram.py* contain the class implementation and a *main* with code showing how to use the class. You can also check the documentation under the *./doc/build/html/index.html*.

ii. Program the method *MeasurementProbability(·)* which, given the distances to the 3 landmarks creates, and returns, a 2d histogram of probability corresponding to $p(z_k|x_k)$. This histogram should look like the one shown in figure 1.1.

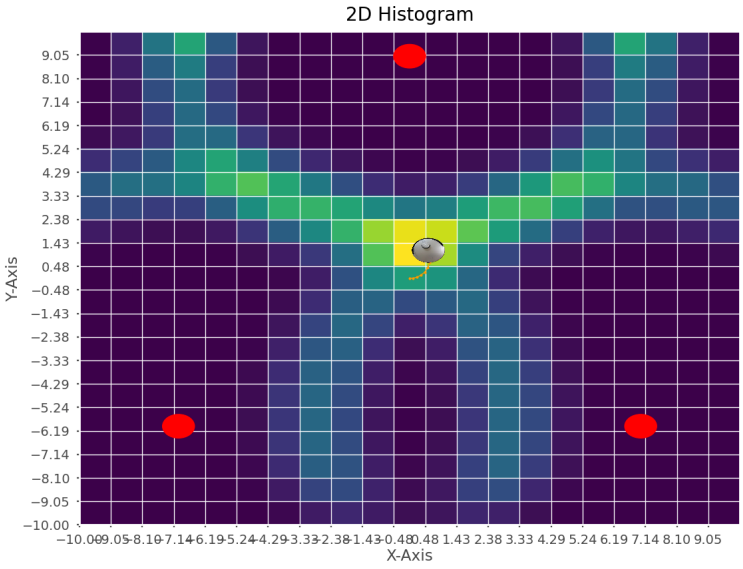


Figure 1.1: Measurement Histogram

4. **Program the main(·) to test the Update Step** alone, without making any prediction:
   (a) Instantiate a p0 Histogram2D initialized as a flat histogram.
   (b) Instantiate a *GLDifferentialDrive* variable and plot its *pk_1* histogram.
   (c) Call the *LocalizationLoop* to see how the measurement histogram evolves over time as the robot moves.

1.2.3 GL: Part II (Prediction)

In this part of the exercise we are going to implement the *Prediction* step of the filter. To do it we need to program the following methods:

1. ***GL_3DOFDiferentialDrive.GetInput*:** This method must return a displacement vector of the robot in number of cells and referenced to the N-Frame. This means that the cells in the x o y directions might be positive or negative. To do it, it will call, the *DR_3DOFDifferentialDrive.GetInput()* method to get the robot displacement during the last sample time. If this displacement is smaller than a cell, it will keep calling he method accumulating the robot displacement until it is at least one cell in any direction.
2. ***GL_3DOFDifferentialDrive.StateTransitionProbability_4_xk_1_uk*** : Computes the state transition probability histogram given the previous robot pose $xk-1$ and the input u_k, returning a histogram of $n \times n$ cells being n the number of cells in the x and the y directions.

3. ***GL_3DOFDifferentialDrive.StateTransitionProbability:*** Computes the complete state transition probability matrix. The matrix is a $n_u \times m_u \times n^2$ matrix, where n_u and m_u are the number of possible displacements in the x and y axis, respectively, and n is the number of cells in the map. For each possible displacement u_k, each previous robot pose x_{k-1} and each current robot pose x_k, the probability $p(x_k|x_{k-1}, u_k)$ is computed. To implement this method it is a good a idea to use the previous one.
4. ***GL_3DOFDifferentialDrive.StateTransitionProbability_4_uk***: This method returns the state transition probability matrix for a given input u_k. The method is used by the *HF.Prediction* method to implement the total probability theorem.
5. ***GL_3DOFDifferentialDrive.uk2cell***: This method is used to convert the robot displacement expressed in cells referenced to the N-Frame (u_k) into an index to be used to access the state transition probability matrix. Note that the indexes used to access a matrix must be postive numbers, while the cell displacement u_k might be negative needing, therefore the conversion provided by this method.
6. **Test the Prediction Step** alone. To do it modigy the *Localize()* method to invoke only the *Prediction()* method and not the *Update()* one. In the *main.py* file, initialize p_0 having a probability 1 of being in cell (0,0) and 0 elsewhere. Check how the probability histogram evolve doing predictions alone (figure 1.2).

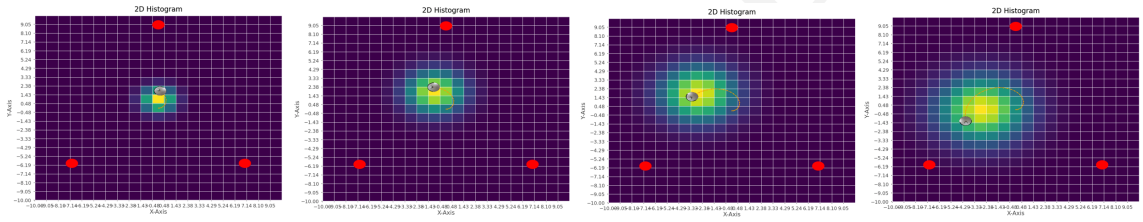


Figure 1.2: Evolution of the position histogram while doing only Prediction

1.3 GL: Part III (Prediction and Update)

Once both parts work perfectly, it is time to couple both together. To do it you need to update the *GL.Localize* method to perform both steps the *Prediction*, and then the *Update*

Bibliography

[1] Pere Ridao and Roger Pi. *prpy: Probabilistic Robot Localization Python Library*. October 2023. Available in pdf in the ./docs/latex folder of the lab repository and in html in the `file: ./docs/html/index.html` file.