Merging of maps obtained with human odometry based on FootSLAM for pedestrian navigation

María Jesús García Puyol

September 9, 2011
Acknowledgements

I would like to express my gratitude to Patrick Robertson for tutoring this project and for his enthusiasm throughout its development. Working day by day with him was a fantastic experience and I feel lucky to have had him as a supervisor.
## Contents

### Acronyms

A General Overview of the FootSLAM Context

#### 2.1 Definitions and Concepts

#### 2.2 State-of-the-art

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.1 Origins of SLAM</td>
<td>19</td>
</tr>
<tr>
<td>2.2.2 FootSLAM Implementation</td>
<td>22</td>
</tr>
<tr>
<td>2.2.3 Addressing Map Merging</td>
<td>23</td>
</tr>
</tbody>
</table>

### 3 Probabilistic Basis of FootSLAM

#### 3.1 Theoretical Basis

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1 Mathematical Model for Human Pedestrian Motion</td>
<td>25</td>
</tr>
<tr>
<td>3.1.2 Nomenclature and Variables Involved</td>
<td>25</td>
</tr>
<tr>
<td>3.1.3 Dynamic Bayesian Network</td>
<td>27</td>
</tr>
</tbody>
</table>

#### 3.2 The Implementation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1 Map Representation</td>
<td>28</td>
</tr>
<tr>
<td>3.2.2 Formal Specification of the Probabilistic Map</td>
<td>28</td>
</tr>
<tr>
<td>3.2.3 Learning the Transition Map</td>
<td>29</td>
</tr>
<tr>
<td>3.2.4 Further Assumptions</td>
<td>30</td>
</tr>
</tbody>
</table>

### 4 Theoretical Basis of FeetSLAM

#### 4.1 FeetSLAM: the Collaborative FootSLAM Problem

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1 Mathematical Model for Human Pedestrian Motion</td>
<td>31</td>
</tr>
</tbody>
</table>

#### 4.2 Dynamic Bayesian Network for a 2-Pedestrian Scenario

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1 Dynamic Bayesian Network for a 2-Pedestrian Scenario</td>
<td>32</td>
</tr>
</tbody>
</table>
5 Towards Map Merging

5.1 Previous Work

5.1.1 2D representation

5.1.2 FootSLAM Maps Parameters

5.1.3 Map Types

5.1.4 The Starting Conditions

5.1.5 Reading the Files

5.1.6 Map Plotting

5.2 Steps Towards Map Merging

5.3 Map Transformation and Projection

5.3.1 Angular Factor

5.3.2 Distance Factor

5.3.3 Transformed Map

5.3.4 Transforming the Starting Conditions

5.3.5 Inverse Transformation

5.4 Map Filtering

5.4.1 Distance Filtering

5.4.2 Angular Filtering

5.5 Examples

5.6 Correlating Maps

5.6.1 Likelihood Function

5.7 Combining Two Maps

5.8 Finding The Best Match Between Two Maps

5.8.1 Running the Loops

5.8.2 Range of Search

5.8.3 Iterating the Loops

5.8.4 Behavior of the Augmented Log Likelihood Function

5.9 The Prior Map

5.10 Important Parameters for FootSLAM

5.10.1 Number of Particles for the Particle Filter

5.10.2 The Dimensions for the Coordinate System

5.11 Comparison with the Ground Truth Floor Plan and Furniture

5.11.1 Plotting the Plan and Furniture

5.11.2 Computation of the Ratio of Crossed Walls and Furniture

5.11.3 Searching Best Fit to Ground Truth

6 "Turbo" FeetSLAM Algorithm

6.1 Goals of the "Turbo" FeetSLAM Algorithm

6.2 The Algorithm

6.3 Data

6.4 FootSLAM Input Parameters Modification

6.5 FootSLAM Map Generation

6.6 Total Map Generation
### CONTENTS

6.6.1 Comparing Maps Pairwise .............................................. 76  
6.6.2 Choosing the Best Combination of All Available Combinations ... 77  
6.6.3 Obtaining a Total Map .................................................. 78  
6.7 Transformation for Individual Maps .................................... 78  
6.8 Prior Map Generation ...................................................... 79  
6.9 Starting Conditions Generation ......................................... 79  
6.10 Zeroth Iteration Properties ............................................. 79  
6.11 Optimization ............................................................... 80  
   6.11.1 Profiling ............................................................... 80  
   6.11.2 Threading ............................................................. 80  
6.12 Properties File ............................................................. 80  
6.13 Logger ....................................................................... 82  
6.14 Naming System ............................................................. 82  
6.15 Complexity Analysis ...................................................... 83  
   6.15.1 Big O Notation ........................................................ 83  
   6.15.2 "Turbo" FeetSLAM Performance ................................. 84  
7 Experimental Verification .................................................. 87  
   7.1 Data Collected at DLR Premises ..................................... 87  
      7.1.1 Starting Conditions and Transformations During Iterative Processing 87  
      7.1.2 Hexagon Transition Error Rate During Iterative Processing 88  
      7.1.3 Comparison With the Ground Truth ........................... 88  
      7.1.4 Discovery: The wall that was misplaced .................... 88  
   7.2 Data Collected at MIT Premises .................................... 88  
      7.2.1 Starting Conditions and Transformations During Iterative Processing 89  
      7.2.2 Comparison with the Ground Truth .......................... 89  
   7.3 Used Resources (CPU, Memory) ...................................... 89  
   7.4 Observations ............................................................ 89  
8 Achievements, Conclusions and Further Work ......................... 105  
   8.1 Achievements ............................................................ 105  
   8.2 Conclusions ............................................................. 106  
   8.3 Further Work ............................................................ 106  
Bibliography ........................................................................ 109  
A Visualization .................................................................... 113  
   A.1 Hexagon Map Drawing Panel Extension ............................ 113  
      A.1.1 Hexagon Maps ....................................................... 113  
      A.1.2 Map Viewer .......................................................... 113  
      A.1.3 Graphical Representation of Subsets of FeetSLAM Maps 114  
      A.1.4 Saving the Maps as Images ..................................... 115  
   A.2 Transformation Viewer .................................................. 115  
      A.2.1 Transformation Viewer for a FootSLAM Map ............ 115  
      A.2.2 Transformation Viewer for the Ground Truth and Furniture 116  
   A.3 Violated Walls and Furniture Viewer ............................... 116  
   A.4 Correlation Viewer ....................................................... 117
CONTENTS

A.5 Chart Generation .......................................................... 119
A.6 Production of Video Sequences of FootSLAM .......................... 120

B Code ............................................................................... 121
B.1 Concepts ...................................................................... 121
B.2 Package and Class Structure .............................................. 121
B.3 UML Diagrams ............................................................... 125
  B.3.1 Inheritance Diagrams ................................................. 125
  B.3.2 Association Diagrams .................................................. 126
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>Two dimensions</td>
</tr>
<tr>
<td>3D</td>
<td>Three dimensions</td>
</tr>
<tr>
<td>DBN</td>
<td>Dynamic Bayesian Network</td>
</tr>
<tr>
<td>DLR</td>
<td>Deutsches Zentrum für Luft-und Raumfahrt, German Aerospace Center</td>
</tr>
<tr>
<td>EBFN</td>
<td>Extended Backus Normal Form</td>
</tr>
<tr>
<td>EKF</td>
<td>Extended Kalman Filter</td>
</tr>
<tr>
<td>GNSS</td>
<td>Global Navigation Satellite System</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IMU</td>
<td>Inertial Measurement Unit</td>
</tr>
<tr>
<td>INS</td>
<td>Inertial Navigation System</td>
</tr>
<tr>
<td>KF</td>
<td>Kalman Filter</td>
</tr>
<tr>
<td>LBS</td>
<td>Location-Based Service</td>
</tr>
<tr>
<td>MAP</td>
<td>Maximum a Posteriori</td>
</tr>
<tr>
<td>MIT</td>
<td>Massachusetts Institute of Technology</td>
</tr>
<tr>
<td>pdf</td>
<td>probability density function</td>
</tr>
<tr>
<td>PF</td>
<td>Particle Filter</td>
</tr>
<tr>
<td>RBPF</td>
<td>Rao-Blackwellized Particle Filter</td>
</tr>
<tr>
<td>SIS</td>
<td>Sequential Importance Sampling</td>
</tr>
<tr>
<td>SLAM</td>
<td>Simultaneous Localization and Mapping</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>ZUPT</td>
<td>Zero Velocity Update</td>
</tr>
</tbody>
</table>
# List of Figures

3.1 Dynamic Bayesian Network for the estimation problem with one pedestrian during three time slices. .............................................. 27
3.2 2D hexagon representation for stochastic pedestrian movement. ....... 29
3.3 Definition of a hexagon transition. ........................................ 29

4.1 Dynamic Bayesian Network for the estimation problem with two pedestrians during three time slices. ........................................ 33
4.2 Two FootSLAM maps, each one with its own coordinate system illustrate the need for transformation. ............................ 34

5.1 Regular hexagon. ................................................................. 37
5.2 Hexagon matrix representation. ............................................. 38
5.3 Example of Aggregated Transition Map and its Best Map. ............. 40
5.4 Schematic representation of how a map is built from a text file. ........ 41
5.5 Examples of shifting and rotation of a point and scaling of a square .... 43
5.6 Illustration of the projection process. ...................................... 44
5.7 Illustration of a transformation of an edge. ................................ 44
5.8 Projection of the vertices of a transformed edge. .......................... 45
5.9 Target hexagons for a transformed edge. ................................... 46
5.10 Illustration of how the traveled edges and the fractional remainder are computed. ................................................................. 47
5.11 Example of the target edges when using the angular factor with two edges. ........................................................................... 48
5.12 Alternative representation for an edge used when computing the angular factor with six edges. .............................................. 48
5.13 Illustration of the overlapping area between a transformed edge and a target edge. ................................................................. 49
5.14 Example of a transformed edge and the edges that also need to receive counts. ................................................................. 50
5.15 Illustration of how the distance factor is computed. ...................... 51
5.16 Illustration of the scale limitation. .......................................... 52
5.17 Illustration of the effect of the distance filter. ............................. 55
5.18 Illustration of the effect of the angular filter. ............................. 55
5.19 Examples of transformations of a FootSLAM map ........................ 56
5.20 Illustration of the combination of two maps. .............................. 62
5.21 Example of how the number of transformations can be reduced using the symmetry of the grid of hexagons. .......................... 64
5.22 Illustration of the sliding of the Accounted Map to benefit from symmetry. 65
5.23 Example of the boundaries for the range of the x shift ............... 66
5.24 Example of the behavior of the correlation function for two maps when
\( \alpha = 0.8 \) and \( \beta = 0.04 \). ........................................ 68
5.25 Illustration of violated walls and furniture. ............................... 70
5.26 Example of the best fit between the building plan and furniture arrange-
ment of building TE01 of DLR premises in Oberpfaffenhofen and a Best
Map. .......................................................... 72

6.1 Schematic illustration of how the algorithm for automatic map merging
works. ................................................................. 74
6.2 Steps for the generation of a total map with 4 different individual maps. 78
6.3 Tree structure for a map merging case in which the number of data sets is
too high. \( N_W = 17, N_D = 3 \). ........................................ 85

7.1 Results of the zeroth iteration for the "DLR data". .......................... 91
7.2 Results of the first iteration for the "DLR data". ............................. 92
7.3 Total combined maps at the end of iterations 2 to 9 for the "DLR data". 93
7.4 Starting Conditions during 10 iterations for the DLR data set number 4. 94
7.5 Transformations applied to the average Starting Conditions (mean x, y,
scale factor and rotation) over the iterations for DLR data set number 4. 95
7.6 Ratio of violated walls and furniture by the total DLR MAP map and the
MAP for data set number 4 over the iterations. ........................... 96
7.7 Best fitting between the total MAP map after 10 iterations and the ground
truth and furniture arrangement at the time of the walks. ................. 97
7.8 Aggregated (on the left) and Best Map (on the right) of data set number 1
of the "DLR data" at iteration 9. The Best Map shows a double corridor
on the top right side. This is not an error, but a result of the stochastic
process that FootSLAM is. ........................................... 97
7.9 Results of the zeroth iteration for the "MIT data". .......................... 98
7.10 Results of the first iteration for the "MIT data". ............................ 99
7.11 Total combined maps at the end of iterations 2 to 9 for the "MIT data". 100
7.12 Starting Conditions during 10 iterations for data set number 4 of "MIT
data". ......................................................... 101
7.13 Transformations applied to the average Starting Conditions (mean x, y,
scale factor and rotation) over the iterations for MIT data set number 4. 102
7.14 Total MIT MAP map at iteration 9 on top of the original plan of the MIT
premises where the walks took place. .................................... 103
7.15 A zoom of Figure 7.14 with the area that was visited by the four walks. 103

8.1 Main tasks undertaken for this thesis. (P) indicates that the task was
developed by Patrick Robertson. ....................................... 105

A.1 Transition map for a given hexagon: the total transition counts for the
hexagon (in the center) and the transition counts for each edge. .......... 114
A.2 The Map Viewer application that allows the user to explore the file system
and choose the map that wants to plot. ................................ 114
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.3</td>
<td>The panel for the &quot;Paint the Results&quot; application with the resulting Aggregated and Best maps obtained with the &quot;DLR data&quot; at iteration 1.</td>
</tr>
<tr>
<td>A.4</td>
<td>Transformation Viewer application to manipulate a map.</td>
</tr>
<tr>
<td>A.5</td>
<td>Transformation Viewer application to transform the ground truth and furniture.</td>
</tr>
<tr>
<td>A.6</td>
<td>Application that allows the transformation of the ground truth and furniture to match a Best Map.</td>
</tr>
<tr>
<td>A.7</td>
<td>Window with different buttons to obtain the correlation between two maps, insert the hexagon correlation factor, run the loops or obtain the combined map.</td>
</tr>
<tr>
<td>A.8</td>
<td>An example of the correlation panel that shows, on a hexagon level, how well two maps fit.</td>
</tr>
<tr>
<td>A.9</td>
<td>Example of a Hexagon Correlation Panel.</td>
</tr>
<tr>
<td>A.10</td>
<td>User interface to run the search of the best transformation for one map to match another.</td>
</tr>
<tr>
<td>B.1</td>
<td>UML inheritance diagram for the transition counts on a hexagon level.</td>
</tr>
<tr>
<td>B.2</td>
<td>UML inheritance diagram for the transition counts on a map level.</td>
</tr>
<tr>
<td>B.3</td>
<td>UML inheritance diagram with for the transformation, correlation and coordinate system.</td>
</tr>
<tr>
<td>B.4</td>
<td>Illustration of the main visualization classes and their most important methods.</td>
</tr>
<tr>
<td>B.5</td>
<td>Illustration of some other main classes and their most important methods.</td>
</tr>
<tr>
<td>B.6</td>
<td>UML association diagram for the log likelihood function.</td>
</tr>
<tr>
<td>B.7</td>
<td>UML association diagram for the Transformation Viewer application.</td>
</tr>
<tr>
<td>B.8</td>
<td>UML association diagram for the Hexagon Matrix class.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Nowadays we live in a world where digital cartography and GNSS (Global Navigation Satellite System) aided automated mapping techniques have become essential tools in our daily life. They provide continuously updated maps of the environment, allowing location-based service (LSB) applications to develop. Gas stations, hotels, monuments, supermarkets, the way back home or people, among others, can be found and located using the geographic coordinates of the GNSS. Nevertheless, we are deprived of these services when we are in underground or indoor locations where satellite signals are strongly disturbed. Moreover, plans of indoor environments are often unavailable, proprietary, outdated or do not reflect furniture or other features that constraint the human motion as much as walls do.

If we could rely on a technology that enabled us to map underground and indoor environments as well as provide us with real-time positioning, many LBS applications such as emergency and security service applications would be possible. People with special needs such as the blind, rescue personnel, visitors to a building, etc. could also benefit from these applications, as they could localize themselves in the areas they visit. How can we overcome these limitations?

Two different approaches can be easily pictured:

1. **Beacon-based solutions:** A series of receivers or emitters are placed at known-locations and allow the estimation of the position of the mobile subject. It is typical to use ultrasound, radio or optical technologies [1].

2. **Beacon-free solutions:** These solutions rely on sensors that the person or mobile object to be located carries, instead of relying on a pre-installed infrastructure.

FootSLAM (Simultaneous Localization And Mapping for pedestrians) is quite a new technology that chooses to address the indoor mapping and positioning challenge with the second approach. This thesis describes the work done during eight months in DLR (Deutsches Zentrum für Luft-und Raumfahrt, German Aerospace Center) in 2010/2011 to extend its performance. More concretely, we have been able to combine different maps that have been obtained with the FootSLAM technique to generate a more precise and complete map of an indoor environment.

This chapter serves as an introduction to the map-aided pedestrian navigation process (Section 1.1), the SLAM problem (Section 1.2), the FootSLAM approach (Section 1.3) and the map merging problem(Section 1.4), the real motivation for this thesis.
1.1 Map-aided Pedestrian Dead Reckoning

In a navigation context, dead reckoning is the process by which the position of a subject can be calculated using a previous determined position and updated with the knowledge or estimation of speeds over elapsed time. Simple integration of the odometry signals of a pedestrian wearing a foot-mounted IMU (Inertial Measurement Unit), which is the case of FootSLAM, can be seen as a form of dead reckoning [14].

The main problem that dead reckoning must face is the unbounded error growth over traveled distance. The error integration can only be stopped at each resting phase of the foot [11], making the time it takes to go from one resting phase of the foot to the next become an important limitation. In an open space, where the human motion is not constrained, this unbounded error growth cannot be avoided. But when the environment constrains the movements of the pedestrian and these constraints (i.e. a map) are known, it has been shown that the error can be limited [14].

Hence the knowledge of the map of the environment where a pedestrian walk improves the performance of the localization problem over the case with no a priori information [14].

1.2 Robotic SLAM

Simultaneous localization and mapping (SLAM) is a technique used by mobile robots that have been placed at an unknown location within an unknown environment to incrementally build a map (without a priori knowledge) and simultaneously locate themselves within the map [2, 3, 4, 6].

Having to locate itself and generate a map of the environment in a simultaneous fashion is a much more difficult problem for the robot than the ones of localization or mapping separately [2]:

- SLAM is more difficult than localization because the map is unknown and has to be estimated gradually while the robot moves.
- SLAM is more difficult than mapping with known poses because the poses are unknown and have to be estimated gradually while the robot moves.

In the next subsections we introduce the localization and mapping problem as separate problems and then the SLAM problem.

1.2.1 Localization Problem

Mobile robot localization is the problem of determining the pose of a robot within a given (known) map of the environment [2]. The pose of the robot refers to its location in 2D and its orientation. In this kind of problems, the map has a global coordinate system and the robot has a local coordinate system. Localizing the robot within the map is reduced to obtaining the transformation between the global and the local coordinate systems so that the robot can express the location of certain points of interest that lie within its own coordinate frame. Knowing the pose of the robot is sufficient to determine this coordinate transformation [2].
1.3 FOOTSLAM

The main challenge for the localization problem comes with the inaccuracies of the actuators of the robot that allow its motion and the sensors that acquire the measurements. Hence the pose has to be inferred from data. Furthermore, a single sensor measurement is not enough and the robot must obtain data over time and integrate it to be able to determine its pose [2, 3].

1.2.2 Mapping Problem

The mapping problem is the problem a robot must face when it does not have any a priori knowledge about the map of its environment. This is the scenario that we have when architects do not provide the blueprints of the building where the robot is located or when these blueprints do not show furniture or other objects, which constraint the motion of the robot as much as the walls and doors do. The robot poses are, nevertheless, well known. Mapping or learning a map from scratch allows the robot to be completely autonomous, in that it can adapt to changes without human supervision.

There are some factors to take into account when mapping [2]:

- **Size of the environment of the robot**: the larger the environment, the more difficult it is to acquire a map.
- **Noise in perception and actuation**: the larger the noise, the more difficult the problem.
- **Perceptual ambiguity**: the more frequently different places look alike, the more difficult it is to establish the connection between different locations traversed at different points in time.

1.2.3 Simultaneous Localization and Mapping Problem

A more realistic case is that in which the robot does not have access to a map of its environment nor to its poses but can acquire measurements using its sensors and can move following a control signal given to its driving wheels. The sensors can be laser rangers or cameras mounted on the robot platform. The given control signal is usually referred to as *odometry* [13].

In SLAM, the robot incrementally builds a map and simultaneously locates itself within this map. Both the trajectory of the platform and the location of all landmarks are estimated on-line without the need for any a priori knowledge of location [5].

Solving the SLAM problem has been one of the major successes in the robotic field in the past decade [3]. Many different approaches have been developed. This thesis’ work is based on the one called FootSLAM and its current implementation and will be presented in the next section.

1.3 FootSLAM

FootSLAM is a pedestrian indoor navigation approach for SLAM in which the role of the robot is played by a pedestrian walking about in a building. Here the word *foot* refers to the use of shoe-mounted inertial sensors. These inertial sensors or IMUs (Inertial
Measurement Unit) are able to measure the pedestrian’s steps while walking [7, 12, 13]. There are two major differentiated applications for FootSLAM [11, 14]:

- We can collect data from people wearing these shoes and then build a map of the building automatically, adapting to possible changes that may occur due to furniture rearrange, construction work or disasters. Later these maps can be used by other people who might need assistance using positioning, such as blind people, visitors, rescue personnel, etc.

- True on-line simultaneous localization and mapping, when the pedestrian does not know where he is and does not have any access to a map of his environment.

The current implementation at DLR is of the first type, where real-time is not needed and the data is processed off-line.

The main difference between FootSLAM and SLAM is that in FootSLAM no visual sensors of any kind are used. The only sensors that are used are the shoe-mounted IMU and optionally a satellite navigation receiver (i.e. GPS) or a magnetometer or both. The pedestrian’s location and the building layout can be jointly estimated using the pedestrian’s odometry (the pedestrian’s own control signals) alone, as measured by the foot mounted IMU [12]. FootSLAM maps can then be used for those cases in which building plans are not available or do not reflect the current state of the building to help the navigation process.

This thesis deepens into the possibility of having multiple data sets coming from different walks around the same premises and how to process them to obtain a combined map.

1.4 Research Statement: the Map Merging Problem

The two simultaneous localization and mapping techniques introduced so far rely on a single robot (SLAM) or pedestrian (FootSLAM). The next step is to address the more realistic situation in which different people walk about in a building during their daily life collecting data that can be used in a combined fashion to generate a map of an indoor environment. This approach is part of Collaborative SLAM, and it is called FeetSLAM.

Suppose each one of those $N_W$ people wears a shoe with IMUs and that the FootSLAM algorithm learns the map of the rooms visited during the day. Three different cases can be envisioned:

- $N_W$ walks all starting at the same starting point or finishing at the same finishing point (or pose) or both and overlapping in explored area to a certain degree.

- $N_W$ walks not necessarily starting or finishing at the same point (or pose) but overlapping in explored area to a certain degree.

- $N_W$ walks not necessarily starting at the same point and not necessarily overlapping in the explored area.

The map merging problem asks how to process and combine these $N_W$ data sets to obtain the best overall map. This resulting map should be more accurate and maybe more
exhaustive (in that it could cover a wider area) than the individual maps separately. Among the benefits of such merger we find that:

- The map can be very easily updated when new parts of the building are opened to the pedestrian (i.e., after months of construction work), or when the furniture has been rearranged.

- The accuracy of the individual maps is improved using the information the other individual maps offer.

Some examples for this FeetSLAM approach are collaborative mapping of airports, museums and other public buildings for use in tourism, travel, and any high-precision LBS.

This thesis has focused mainly on the task of developing, implementing and testing an algorithm to merge data sets and maps in an off-line fashion. The optimal algorithm is prohibitively complex but a suboptimal iterative variant has been proposed to face the restrictions in the resources such as limited memory and has proven to work in two different indoor scenarios.

1.5 Goals

This thesis goal is the implementation of a fully automated algorithm to merge FootSLAM maps in order to:

- Obtain an extensive map of the walkable areas of a building that encompasses the information provided by all the maps involved.

- Increase the accuracy and precision of maps that have been obtained from a single walk.

1.6 Phases of work

The phases that will need to be undertaken in order to develop a fully automated map merging algorithm are:

1. **State-of-the-art Literature Research.** This phase will consist of reading papers and other scientific publications on Particle Filters, SLAM and FootSLAM, to get familiarized with the terms and concepts.

2. **Theoretical Derivation.** At this stage the work will focus on the derivation of a function to transform and project FootSLAM maps and a function to obtain the likelihood between two maps.

3. **Practical Implementation.** This phase will consist of the implementation of the proper methods to transform and project maps efficiently and the development of an automated algorithm that finds the best transformation that makes one map match the another. These functions and algorithms will need to be developed within the current Java® FootSLAM implementation frame at DLR and will rely on GUIs (Graphical User Interface) where the maps will be visualized.
4. **Evaluation of FootSLAM Maps.** This phase will involve the research and implementation of a function that measures how many walls of the ground truth and pieces of furniture are violated by the FootSLAM maps.

5. **Fully-automated Map Merging Algorithm Implementation.** A theoretical derivation will precede the implementation of an algorithm that will take two or more FootSLAM and merge them in an iterative fashion.

6. **Optimization of the Map Merging Algorithm.** At this point the algorithm to merge maps will be optimized to reduce the time to combine the maps. The key idea will rely on concurrent (thread) programming.

7. **Experimental Verification.** At this stage real data from multiple walks in an indoor environment will be used to see the performance of the implemented algorithm.

8. **Thesis Documentation.** This thesis work has been documented on a daily basis. This thesis document has been generated using \LaTeX.

### 1.7 Thesis Outline

This thesis is organized as follows: in chapter 2 a context for FootSLAM and the important ideas involved in the process are presented. In chapters 3 and 4 a more theoretical view for the FootSLAM and FeetSLAM techniques is provided. In chapter 5 the intermediate functions towards a map merging algorithm are introduced. The algorithm itself is presented in chapter 6 and the results obtained in two indoor scenarios, DLR and MIT (Massachusetts Institute of Technology) premises, are shown in chapter 7. Finally, some conclusions and further work ideas are detailed in chapter 8.

Two appendixes have been added to show the main implemented GUIs for the visualization of FootSLAM maps, ground truth plans and curves (Appendix A) and the structure of the Java® code (Appendix B) that has been written to generate a map merging algorithm.
Chapter 2

A General Overview of the FootSLAM Context

This chapter intends to familiarize the reader with the main concepts that support FootSLAM and help understand how it works. Since this thesis job is to explain how FootSLAM map merging is achieved, but not FootSLAM itself, the reader can see the papers by P. Robertson et al. [11], [12] and [13] for further information about FootSLAM.

2.1 Definitions and Concepts

This section describes some very useful concepts and ideas (in alphabetical order) that will help comprehend the mechanism upon which FootSLAM works. All these notions are explained within the context of SLAM and FootSLAM collecting and processing of the data, and will be referred to throughout this thesis.

**Bayes Filter:** the Bayes Filter builds the posterior pdf of the state by taking all the existing information. Since this pdf relies on the available information, it is referred to as the complete solution to the estimation state problem. An estimate will be called optimal if obtained from this pdf. One should take into account that the Bayes Filter is based on the Markov Assumption, which assumes that the past and future states are independent if we know what the present state is [15].

**Bayes Theorem:** if $x$ is a magnitude that wants to be inferred from $y$ (data, i.e. sensor measurement), then the probability $p(x)$ is called prior probability density function and represents the already known information about $x$ before incorporating the data $y$. The probability $p(x|y)$ is called the posterior probability density function over $X$ or conditional probability density function of $x$ given $y$. Bayes rule provides a very useful way to compute this posterior $p(x|y)$ (which is a diagnostic) using the "inverse" conditional probability $p(y|x)$ and the prior probability $p(x)$:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\int p(y|x')p(x')dx'}.$$

(2.1)

This inverse probability $p(y|x)$ describes the probability of data $y$ assuming that $x$ was the case and it is causal. Often causal knowledge is easier to obtain, for example
by counting frequencies. We use Bayes theorem to update the knowledge we have about the target state (here $x$) in the light of extra information from new data (here $y$) [2].

**DBN:** Dynamic Bayesian Network: a Bayesian Network is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph. A DBN is a Bayesian network that represents sequences of variables, such as time-series.

**EKF:** Extended Kalman Filter (see KF): the EKF can face the more realistic case in which the system is not linear by *linearizing* via Taylor Expansion. EKF calculates an approximation to the true posterior using a Gaussian density function, represented by a mean and a covariance, just as the KF does. The difference is that in the KF, the posterior representation is exact, and in the EKF it is only an approximation [2, 15].

**FastSLAM:** it is an algorithm that is based on an exact factorization of the posterior into a product of conditional landmark distributions and a distribution over robot paths [10]. The conditioning on a pose allows the landmarks to be estimated independently, thus leading to lower complexity. FastSLAM recursively estimates the full posterior distribution over robot pose and landmark locations but scaling logarithmically with the number of landmarks in the map.

**First-order Markov process:** it is a process that models the temporal evolution of navigation parameters when the future state given the present state and all its past states depend only on the present state (and not on any past states).

**Gaussian Filter:** it is a recursive state estimator that uses multivariate normal distributions to represent posterior density functions in continuous spaces, that is a technique to implement Bayes filters. A multivariate normal distribution over the variable $x$ is defined by:

$$p(x) = \det(2\pi\sigma)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x - \mu)^T\sigma^{-1}(x - \mu)\}. \quad (2.2)$$

This density function is characterized by two sets of parameters: The mean $\mu$ and the covariance $\sigma$. The mean $\mu$ is a vector with the same dimensionality as the state $x$. The covariance is a quadratic matrix that is symmetric and positive-semidefinite. Its dimension is the dimensionality of the state $x$ squared [2].

**IMU:** Inertial Measurement Unit: inertial sensor that is installed at the pedestrian’s shoe to collect data. An IMU contains three orthogonal rate-gyrosopes and accelerometers, which report angular velocity and acceleration respectively [7]. Nevertheless, errors still accrue over time, especially the heading error which is only weakly observable from these zero velocity updates (ZUPTs). The drifts lead to large divergence from the true path [13].

**(NavShoe with) INS:** Inertial Navigation System: a navigation system that can calculate the displacement of the foot between two resting phases (step) very precisely with the help of ZUPTs, preventing the error from growing excessively.
2.1. DEFINITIONS AND CONCEPTS

KF: Kalman Filter: it is a Gaussian filter, used for filtering and prediction in linear systems. The Kalman filter chooses to represent the posterior density at every time instant $t$ by its moments: its mean $\mu_t$ and its covariance $\Sigma_t$ [2, 15].

Landmark: a place in the space that can be easily located by a robot’s sensors.

Odometry: it refers to the given control signals to a robot or the pedestrian’s own control signals that allow their motion.

pdf: probability density function: it describes the relative likelihood for a random variable to occur at a given point.

PF: Particle Filter, also known as Sequential Importance Sampling (SIS). Particle’s filter core idea is to represent the required pdf by a collection of samples, each one with a corresponding weight. Based on these samples and weights, the particle filter can compute estimates. With this kind of representation, many more distributions can be characterized than with KF and EKF. If the number of samples is very large, the performance of this filter approaches the optimal Bayes estimate. Particle filtering also performs a sampling of the particles to reduce the computational cost of the problem. To do so, the PF combines the particles at a particular position into a single particle with a certain weight (or importance weight) that accounts for the number of particles that were taken into consideration. After the normalization of the weights, the algorithm resamples $N$ particles from the system aided by those weights [2, 15].

Pose of the pedestrian: it encompasses the position and orientation of the pedestrian.

RBPF: Rao-Blackwellized Particle Filter: RBPF identifies variables that do not need to be sampled to lower the cost of the computation. The more a network can be partitioned into subnetworks, the better the performance of the filter. RBPF is the supporting algorithm upon which the FastSLAM technique works: each particle represents a pose and a set of independent EKFs for each landmark [3].

Recursive Filter: it is a kind of filter that allows to update the data we have when extra information becomes available from new data, that is, sequentially, rather than as a batch. There are two main steps in every recursive filter: The prediction step and the update step. The prediction step uses the system model to predict the state pdf forward from one measurement time to the next and the update step uses the latest measurement to modify the prediction pdf. This is achieved using Bayes theorem [15].

State: it is the collection of all the characteristics of the pedestrian and his environment that can have some influence in the future.

State estimation: it focuses on the estimation of quantities that have been inferred from data that has been collected from sensors. Because the measurements obtained from these sensors are distorted by noise, state estimation goal is to recover state variables from them [2].
Step: a step is the movement of the IMU-mounted shoe from one resting phase to the next.

ZUPT: Zero Velocity Updates: the ZUPTs can be applied during the stationary resting phase (i.e. in contact with the ground) during each gait cycle, when the velocity of the sensor array in that moment is zero in all axes. The application of such constraints replaces the cubic-in-time position error growth of the inertial sensor with an error accumulation that is linear in the number of steps [7, 14].

2.2 State-of-the-art

In this section a perspective of the progress made so far in the fields of FootSLAM and map merging is provided. Given that FootSLAM builds on the robotic SLAM technique described in 1.2, we will start with a brief history of the SLAM problem.

2.2.1 Origins of SLAM

The IEEE Robotics and Automation Conference that took place in San Francisco, California, in 1986 is considered to be the beginning of consistent probabilistic mapping, but it was not until the 1995 International Symposium on Robotics Research that the structure of the SLAM problem, the convergence result and the coining of the acronym SLAM were presented. Hence the SLAM problem can be seen as a quite new one [3].

The work by H. Durrant-Whyte and T. Bailey in [3] and [4] (2006) is a two-part tutorial that gives us a first idea of the basics upon which SLAM works. The SLAM problem is addressed in a Bayesian form and the main variables involved are: the state vector with the pose of the robot, the control vector applied to move the robot, the map of landmarks and the observations taken by the robot. The probabilistic form of SLAM refers to a joint posterior density function of the map and the robot state at a certain time instant conditioned to the observations and control inputs up to that time along with the initial state of the robot. Under the assumption of the state being a Markov process, the SLAM algorithm is then implemented in a standard two step recursive prediction-correction form.

In the past twenty-five years many different solutions have been developed. The two most notorious ones are [3]:

- EKF-SLAM: it relies on the use of Extended Kalman Filter (EKF).
- FastSLAM: it is supported by Rao-Blackwellized particle filters (RBPF).

In the second part of the tutorial [4] we can find a description of the computation, convergence and data association issues in SLAM, which was the main topic of research during the last fifteen years. All these studies and developments have made of SLAM a well understood and established part of robotics.

Very many approaches have been proposed to implement SLAM. Some examples are: robotic SLAM [6], visualSLAM [8], RatSLAM [20], WiSLAM [9] and FootSLAM [11, 12, 13], our main subject of interest.
2.2.2 FootSLAM Implementation

FootSLAM is quite a novel approach for SLAM. In 2009 the term FootSLAM was introduced to refer to a dynamic Bayesian estimation formulation for simultaneous localization and mapping for pedestrians based on odometry with foot mounted inertial sensors [12]. The original idea here is that the only sensors that are used are the shoe-mounted IMUs, that is, no visual sensors are used. Thus, the pedestrian’s pose and the building plan can be jointly estimated by merely using the pedestrian’s odometry that the foot-mounted IMU measures. The problem has been resolved using a Rao-Blackwellized particle filter that follows the FastSLAM principle.

Currently, the application domain of FootSLAM differs from that of traditional (robotic) SLAM in that its purpose is to achieve the generation of maps in an automatic fashion so that they can be used later by people wishing to navigate in the building, that is, the data collected by the pedestrian can be processed off-line.

For a practical implementation, a probabilistic map that represents human motion in a 2D grid of adjacent hexagons has been chosen. Every possible future next step of the subject is based only on his current location and the probabilities of each possible next step can be learned through observation [13]. We will explain how this probabilistic map is built in chapter 3.

FootSLAM maps are accurate to about 1-3 meters with better accuracy in the corridors that were frequented more often. Hexagons have to be re-visited once or twice in each main indirected traversal axis for a usable map to emerge, and this governs the required duration of a walk (12 minutes in average). Accuracy in any case is limited by the average physical structure dimension, such as corridors and doors, which is about 1-2m [13].

All results so far have been obtained from just a single track or walk-through and assume no further processing to merge tracks. In the experiments by P. Robertson et al. in [11], GPS was used in the outdoor portion and optionally a magnetometer to reduce ambiguities and obtain more accuracy when building the maps. Some observations were made:

• The estimator converges when the areas are revisited.
• Knowledge of the building help improve the error.
• Knowledge of the real building plan gives best performance.
• When FootSLAM is used, the accuracy cannot be better than the anchor achieved while using GPS before entering the building (3-7m).

Recent work addressed PlaceSLAM [19], that is, the use of recognizable markers that have been manually placed around the building to help the FootSLAM algorithm converge. Every time the pedestrian reaches one of these markers, they are flagged. Furthermore, there are being new developments in sensor technology that will help achieve substantial improvements to performance.

Future work should also integrate more sensors and address 3D implementation.

2.2.3 Addressing Map Merging

If we go back to (robotic) SLAM, some research and work on this topic can be found:
In [16] a distributed approach for multi-robot navigation is proposed. Each robot is capable of mapping the environment on its own and is also able to communicate intermittently with other robots to exchange information.

In [17] an algorithm to perform on-line map merging between robots is described. When a robot detects another, it sends its processed map and the master robot generates a very accurate global map, cutting down the global map building time.

In [18] an algorithm is proposed to be implemented on multiple robot platforms, allowing the generation of a single map of their environment. The generation of the maps relies on the most recent sensor measurement and it is based on the likelihood of the continuation of the previous maps.

Nevertheless, our approach differs from these ones in that we are not in need of real-time map merging. We are focusing on the case in which we have two or more data sets that come from different walks. We can process and combine them in an off-line fashion to generate a more accurate map of the building. Since there is no previous work on this field, we have designed our own and original algorithms to merge maps from scratch. The rest of this thesis explains how we have achieved this.
Chapter 3

Probabilistic Basis of FootSLAM

In sections 3.1 and 3.2, we will show what the theoretical basis for FootSLAM is, going from a general approach for any kind of map representation to the specific representation chosen in this case: a grid of hexagons that represent the 2D space where the pedestrian can walk [13].

3.1 Theoretical Basis

Since the work of this thesis is built at the level of maps, being the FootSLAM map generation almost a black box, we will only present the main ideas that one should understand to be able to comprehend how the FootSLAM algorithm is able to generate a map. The reader can find the whole formal derivation in [12].

3.1.1 Mathematical Model for Human Pedestrian Motion

The first step towards obtaining a FootSLAM model is to find a good but simple representation for human motion. Human motion is a complex stochastic process, but for our purposes a first order Markov process has proven to be sufficient to model it. With this kind of model, every possible future next step of the pedestrian can be represented based only on his current position and the probabilities of each possible next step are learned through observation.

The main difficulty to overcome is the uncertainty of the person’s location. The FastSLAM approach is suited to face this problem: each particle assumes a certain pose history and estimates the motion probabilities conditioned on its particular state. FastSLAM [10] uses Rao-Blackwellized particle filters, where each particle has an associated weight stating how well its motion matches its previous observations of how the pedestrian had walked when in a certain position. This algorithm has proven to converge very quickly when the subject revisits locations once or twice during the walk.

3.1.2 Nomenclature and Variables Involved

In this section a first contact with the variables that will be used to explain how FootSLAM works is provided. When written in bold face, they denote a random variable.
$k$ time instant in discrete-time contexts

$X$ a random variable that can take on a countable number of values in \( \{x_1, x_2, \ldots, x_n\} \).

$X_k$ a random variable $X$ at instant $k$.

$X_{0:k} = \{X_0, X_1, \ldots, X_k\}$ the history of the random variable up to time $k$

$P_k$ the pose of a pedestrian at time instant $k$

$P_{0:k} = \{P_0, P_1, \ldots, P_k\}$ the history of the pedestrian’s pose up to time $k$

$Z_k$ the step measurement at time instant $k$

$Z_{1:k} = \{Z_1, Z_2, \ldots, Z_k\}$ a set of all available measurements up to time $k$

$E_k$ the inertial sensor errors at time instant $k$

$E_{0:k} = \{E_0, E_1, \ldots, E_k\}$ a set of all available errors up to time $k$

$\text{Vis}_k$ the visual indications and signals that the person sees at time instant $k$

$\text{Int}_k$ the intention of the person at time instant $k$

$U_k$ step vector at time $k$, which represents the change from pose at time $k-1$ to pose at time $k$

$U_{1:k} = \{U_1, U_2, \ldots, U_k\}$ a set of all available step vectors up to time $k$

$M$ the map, which is time invariant.

$r$ radius of the hexagons used in the partition of the 2D space for FootSLAM

$h$ the index used to uniquely reference a hexagon’s position

$H = \{H_0, H_1, \ldots, H_h, \ldots, H_{N_H-1}\}$ the set of $N_H$ hexagons that were visited by the particles used in the particle filter.

$M_h = \{M_h^0, M_h^1, \ldots, M_h^5\}$ the set of transition probabilities across the edges of the $h$-th hexagon

$e$ the index used to reference one of the six edges of a hexagon.

$i$ the index used to reference one of the particles used in the particle filter

$\alpha$ the prior knowledge about transition counts
3.1.3 Dynamic Bayesian Network

A dynamic Bayesian network (DBN) is a Bayesian network that represents sequences of variables. To generate a DBN for our scenario, first the human motion needs to be properly represented. A pedestrian uses his vision \( \text{Vis} \) to avoid walls and other obstacles like pieces of furniture and guide his motion. This visual cues along with his own intentions \( \text{Int} \) will determine his next steps.

Since the input to the human visual system cannot be observed and the destination of the person’s walk is unknown, a foot-mounted IMU measures the steps a person takes. In Figure 3.1 the dynamic Bayesian network (DBN) of the estimation problem is represented. All incoming arrows originate in state variables that influence the value of the target. See Section 3.1.2 to check the meaning of the variables represented in the figure.

Note that the step transition vector \( \mathbf{U}_k \) has the following property: given the previous pose \( \mathbf{P}_{k-1} \) and the new pose \( \mathbf{P}_k \) then the step transition \( \mathbf{U}_k \) can be entirely determined; just as knowledge of two of the states \( \mathbf{P}_{k-1}, \mathbf{P}_k \) and \( \mathbf{U}_k \) determines the unknown one [12, 13].

![Figure 3.1: Dynamic Bayesian Network](image)

The main goal is to estimate the states and state histories of the DBN given the series of all observations \( \mathbf{Z}_{1:k} \) from the foot-mounted IMU. The goal in a Bayesian formulation
is to compute the joint posterior:

\[ p(P_{0:k}, U_{0:k}, E_{0:k}, M|Z_{1:k}) = p(\{PUE\}_{0:k}, M|Z_{1:k}), \]  

which can be factorized into:

\[ p(M|\{PUE\}_{0:k}) \cdot p(\{PUE\}_{0:k}|Z_{1:k}) = p(M|P_{0:k}) \cdot p(\{PUE\}_{0:k}|Z_{1:k}). \]  

(3.2)

The left term in (3.2) represents the map building process from the odometry data \( P_{0:k} \) that has been inferred with the right term from the measurements \( Z_{1:k} \).

It is important to point out now that the encoding vision and intention of the pedestrian are never actually used; they only serve as important structural constraints in the DBN (linking \( P_{k-1} \) and \( M \) as "parent" nodes of \( U_k \)).

## 3.2 The Implementation

### 3.2.1 Map Representation

FootSLAM maps are a probabilistic representation of human motion. Each next step of the pedestrian is drawn from a location dependent probability distribution that only depends on his current position.

A grid of regular and adjacent hexagons is used to partition the 2D space where the pedestrian can walk. Every particle in the particle filter stores its own history of transition probabilities across the edges (numbered from 0 to 5) that it visited and updates them using the motion taken by its hypothesis, generating what we call a Transition Map. The particles use a uniform prior when estimating the probability distributions of the transition across the edges. Additionally, the particles also explore deviations of the true pedestrian’s path to account for the IMU errors \( E_{0:k} \). Furthermore, each particle has an associated weight that is updated every time the particle falls onto a new hexagon and based on the previous knowledge the particle had of the state transitions across the edges of the hexagon where it is now located. Consequently, a particle is rewarded when crossing an edge that it had already crossed, that is, when revisiting similar transitions in the grid.

### 3.2.2 Formal Specification of the Probabilistic Map

The starting point for our specification of the probabilistic map is the 2D grid of adjacent hexagons of a given radius \( r \). For each map only the hexagons that were visited by any particle are stored, defining \( H = \{H_0, H_1, \ldots, H_{h}, \ldots, H_{N_h-1}\} \) as the set of \( N_H \) hexagons, where the index \( h \) refers to the hexagon’s position. See Figure 3.2.

Additionally \( M_h = \{M_h^0, M_h^1, \ldots, M_h^5\} \) is defined as the set of six transition probabilities across the edges of the \( h \)-th hexagon and

\[ M_{h}(P_{k}) = p(P_k \in H_j|P_{k-1} \in H_h), \]  

(3.3)

and \( i \neq h \) (we moved to a new hexagon) and where \( 0 \leq e(U_k) \leq 5 \) is the edge of the outgoing hexagon associated with \( U_k \), that is, the edge of the hexagon in which \( P_{k-1} \) lies and which borders the hexagon in which \( P_k \) lies. See Figure 3.3 for a better understanding.
3.2. THE IMPLEMENTATION

Hexagons in which the space was partitioned into

Hexagons that were visited by any particle

N_H hexagons

Figure 3.2: 2D hexagon representation for stochastic pedestrian movement.

Also we can state that \( \sum_{e=0}^{e=5} M_h^e = 1 \). But the random variable \( M_h^e \) denoting the transition probabilities of the hexagon \( h \) across edge \( e \) is unknown to us. We can only estimate \( p(M_h^e | P_{0:k-1}) \) by observing \( P_{0:k-1} \).

The map random variable \( M \) can be decomposed as follows:

\[
M = \{ M_0, M_1, \ldots, M_h, \ldots, M_{N_H-1} \}, \quad (3.4)
\]

where \( M_h \) is a random variable vector of length 6 denoting the transition probabilities of the hexagon with index \( h \). To ease the notation, from now on we will write \( \tilde{h} \) for outgoing hexagon \( h(P_{k-1}) \) and \( \tilde{e} \) for the crossed edge \( e(U_k) \).

3.2.3 Learning the Transition Map

The Transition Map is learned by each particle \( i \) by counting each transition it makes from \( P_{k-1}^i \) to \( P_k^i \) across edge \( \tilde{e} \) in its local map for hexagon \( H_{\tilde{h}} \). Operating in this manner, each particle stores its whole path through the hexagon grid. Learning the map is based on Bayesian learning of multinomial and binomial distributions [13], by which each particle weight is updated as follows:

\[
w_k^i = w_{k-1}^i \cdot \left\{ \frac{N_{\tilde{h}}^\tilde{e} + \alpha_{\tilde{h}}^\tilde{e}}{N_{\tilde{h}} + \alpha_{\tilde{h}}} \right\}^i, \quad (3.5)
\]

where \( N_{\tilde{h}}^\tilde{e} \) are the transition counts for edge \( \tilde{e} \) of hexagon \( \tilde{h} \) and \( N_{\tilde{h}} = \sum_{e=0}^{e=5} N_{\tilde{h}}^e \) always in the map of the particle \( i \) computed up to step \( k - 1 \).
The terms $\alpha_{\tilde{h}}^e$ and $\alpha_{\tilde{h}} = \sum_{e=0}^{e=5} \alpha_{\tilde{h}}^e$ represent the \textit{a priori} knowledge regarding the transition counts across the edges of hexagon $\tilde{h}$ for particle $\tilde{i}$. When no real prior information is available, $\alpha_{\tilde{h}}^e$ has been chosen empirically to be $0.8^{\forall\{e, h, i\}}$, that is, a uniform prior distribution is assumed.

The Aggregated Transition Map of all the particles is then obtained by adding all the particle's Transition Maps properly weighted.

3.2.4 Further Assumptions

Some of the assumptions that are taken when running the FootSLAM algorithm are:

- When computing the counts for each particle, the counts for the outgoing hexagon are incremented as well as the incoming hexagon (on the appropriate edge).

- Weight update is only performed when we have stepped out of the last hexagon.

- Multiple weight update is performed for all edges crossed if the step was larger than one hexagon.
Chapter 4

Theoretical Basis of FeetSLAM

This chapter presents the FeetSLAM technique and the motivation for an iterative approach for its implementation. This chapter introduces the first extensions to the FootSLAM concept that this thesis has accomplished from a theoretical point of view.

4.1 FeetSLAM: the Collaborative FootSLAM Problem

FeetSLAM is a collaborative technique that extends the FootSLAM method to the multi-user case in which \( N_W \) pedestrians walk in an area with the same or different starting and finishing poses. Two different scenarios borrowed from FootSLAM usage can be pictured [14]:

- **Real-time Usage**: a building is mapped by \( N_W \) collaborating pedestrians in order to provide each other or other subjects with instantaneous map and position information. This is the case of emergency situations, i.e., fire-fighters entering a building through different or the same entrances and proceeding to search and rescue people, where it is critical to cover all the areas and avoid revisiting them. In law enforcement scenarios, knowing every team member’s position within a map may potentially reduce the risk of accidentally harming a team member. In these applications a priori map data may not be available.

- **Off-line Mapping**: a building’s map is derived so that it will later serve as basis for localizing pedestrians by map-aided pedestrian dead reckoning. An example of this is collaborative mapping of airports, museums, shopping centers and other public buildings for use in tourism, travel, commerce, and any high-precision location based services.

In this thesis, we will focus on non-real time processing, in the expectation that the techniques can be sped up to real-time capabilities over time.

In our approach to FeetSLAM, pedestrians roam through accessible rooms and areas on all levels of a building and are equipped with a foot-mounted IMU and most likely a GPS receiver for anchoring in an absolute coordinate frame. In this scenario the measurement data needs to be recorded and will then be processed off-line. The resulting map is then stored at a server or distributed to localization devices that use it to perform map-
CHAPTER 4. THEORETICAL BASIS OF FEETSLAM

aided pedestrian dead reckoning. As more data are collected the maps can be refined to incorporate the new walks.

4.2 Dynamic Bayesian Network for a 2-Pedestrian Scenario

The new scenario has two or more pedestrians that collaborate to generate a map. For our explanation we will focus on the case in which only two pedestrians walk in an area. In figure 4.1 the DBN for the estimation problem with two pedestrians who walk within the same area has been represented. The motion of the two pedestrians is constrained by the same time-invariant map \( M \) that we want to obtain. Each one of the pedestrians has their own visual cues \( \text{Vis} \) and intentions \( \text{Int} \), and each one wears a Foot-mounted IMU that observes their steps \( U \), obtaining the measurements \( Z \) that are influenced by correlated error \( E \) to derive the poses \( P \). Furthermore, each pedestrian starts his walk at a different pose and with different orientation, defined by their own Starting Conditions \( \text{SC} \), also time-invariant.

The Starting Conditions \( \text{SC} \) specify the starting pose of the pedestrian with four Gaussian distributions, one for the \( x \) coordinate, one for the \( y \) coordinate, one for the heading angle \( a \) and one for the scale factor \( s \). Each one of them is defined by an average value \( \mu \) and a standard deviation \( \sigma \). A Gaussian distribution for the starting \( x \) coordinate can be specified with:

\[
p(x) = \frac{1}{\sqrt{2\pi\sigma^2_x}} e^{-\frac{(x-\mu_x)^2}{2\sigma^2_x}}. \tag{4.1}
\]

One must note that the poses of each pedestrian \( P \) are relative to his or her own Starting Conditions \( \text{SC} \), which means that after running FootSLAM for the two sets of poses, one of the resulting maps will need to be transformed to fit the other. This is because of the way the particles explore different hypothesis of pose and error histories and the uncertainty of the Starting Conditions (expressed in the form of a standard deviation, \( \sigma_x, \sigma_y, \sigma_r, \sigma_s \)).

4.3 The Optimal Estimator

The optimal FeetSLAM estimator is a trivial extension of the FootSLAM algorithm. A Bayesian estimator needs to process all data sets sequentially or in parallel and the state to include the unknown Starting Conditions of each walk. The common element binding the walks is the map \( M \). This relationship can be seen in the DBN in Figure 4.1 for two walks.

The goal of the Bayesian formulation for a 2-Pedestrian scenario is to compute:

\[
p(P^1_{0:k}, P^2_{0:k}, U^1_{0:k}, U^2_{0:k}, E^1_{0:k}, E^2_{0:k}, \text{SC}^1, \text{SC}^2, M|Z^1_{1:k}, Z^2_{1:k}) = p((\text{PUE}^1)^{1,2}_{0:k}, \text{SC}^{1,2}, M|Z^{1,2}_{1:k}), \tag{4.2}
\]

which can be easily extended to a \( N_W \)-Pedestrian scenario as follows:

\[
p((\text{PUE}^1)^{1:N_W}_{0:k}, \text{SC}^{1:N_W}, M|Z^{1:N_W}_{1:k}). \tag{4.3}
\]
Figure 4.1: Dynamic Bayesian Network for the estimation problem with two pedestrians during three time slices. We have omitted an index that differentiates the two segments for Pedestrian #1 and Pedestrian #2 for clarity.

Note that for this DBN, the time indexes \( k \) are the same for the walks. This is not a requirement for our map merging algorithm, in which the data are processed off-line, and hence can be obtained from walks occurring at different times. Conceptually, what we are doing is pairing the time indexes \( k \) of the walks and using padding (adding samples with value 0) when the walks are not equally long.

In our implementation, where the estimator relies on a particle filter, the particles would have to explore the state space of all odometry error sequences and all Starting Conditions and this approach would suffer from severe depletion, since too many particles are required. A further practical complication for a finite number of particles is that in a
sequential approach FootSLAM will tend to favor early data, which will bias the map to data processed early in the sequential estimation process.

4.4 Motivation for an Iterative Approach

The DBN from Figure 4.1 has similarities to a family of error correction coding schemes from digital communications theory. In 1993 a family of codes that are decoded iteratively at the receiver were developed [22]. The codes are constructed by concatenating two or more component codes and are now used in a wide range of modern mobile communication standards. The optimal detector is prohibitively complex but the suboptimal, iterative variant exhibits very good error correction performance. From a Bayesian perspective this kind of iterative processing can be seen as a form of loopy belief propagation in a Bayesian network [21]. These codes are known as "Turbo" codes. The name "Turbo" was chosen to reflect the nature of the iterative processing. Our algorithm has been named after them and it is called "Turbo" FeetSLAM Algorithm.

For a given walk and its odometry data set, other walks can be incorporated in a given FootSLAM estimation process in the form of prior counts in the FootSLAM maps (see equation 3.5). This is only possible, however, if the geometric transformation that places all data within the same coordinate system is known. As can be seen in Figure 4.2, the resulting maps from different walks are framed within different coordinate systems.

Figure 4.2: Two FootSLAM maps have been obtained by two pedestrians walking in the same building, with some degree of overlapping areas. Each map is framed within its own coordinate system. In order to combine them, a transformation is needed.

So, before a map is applied as an a priori map for another FootSLAM process both data sets must be framed within the same coordinate system. The iterative "Turbo"
4.4. MOTIVATION FOR AN ITERATIVE APPROACH

FeetSLAM Algorithm starts by processing all data in individual FootSLAM runs and then combines the resulting maps in the form of a global map. Using this global map and the individual maps, the algorithm generates a prior map for each data set for the next batch of FootSLAM runs. This is repeated for a number of iterations. It should be noted that when constructing the the prior map for a given odometry data set, the map contribution that arose from that given walk should not be included in the prior map that will be used the next time we process that walk. This is in exact analogy to the prior construction in "Turbo" decoding and loopy belief propagation.

Proceeding like this, the individual FootSLAM maps are benefited from the knowledge of the other maps, and hence increase their accuracy and a global map of the walking areas is generated. Over the iterations the Starting Conditions of each walk become more precise, and hence the resulting individual and global FootSLAM maps are more accurate.

The pseudo-code for the FeetSLAM Algorithm is:

\[
\begin{align*}
\text{FOR} & \text{ iteration FROM 0 TO #iterations-1} \\
& \text{FOR data set FROM 1 TO N_W} \\
& \quad \text{map(data set, iteration)} = \text{FootSLAM(prior(data set, iteration-1), data set)} \\
& \text{END FOR} \\
& \text{total map(iteration)} = \text{combination(maps (iteration))} \\
& \text{FOR data set FROM 1 TO N_W} \\
& \quad \text{prior(data set, iteration)} = \text{obtain prior(maps(iteration) \ map(data set, iteration), total map)} \\
& \text{END FOR} \\
\end{align*}
\]

We will go back to the "Turbo" FeetSLAM Algorithm in chapter 6, where its implementation will be presented. But before, in chapter 5, the intermediate tasks that have been developed towards this iterative map merging algorithm are introduced.
Chapter 5
Towards Map Merging

This chapter is dedicated to explain the current implementation of FootSLAM and how it has been extended to allow the development of a map merging algorithm, which will be introduced in the next chapter.

5.1 Previous Work

This thesis’ work represents an extension to the the FootSLAM method, which has been implemented in a Java® platform using Eclipse®. The first step towards the implementation of new classes and applications is to learn how the current version has been implemented so that the new code is generated coherently.

Mohammed Khider and mostly Patrick Robertson, the supervisor of this thesis, were the authors of the current version of FootSLAM. The most important aspects of their work are summarized in the following subsections.

5.1.1 2D representation

The cornerstone on which the current version of FootSLAM rests is a grid of adjacent hexagons that is used to represent the 2D environment where the pedestrian walks.

A hexagon is a polygon with six edges and six vertices. The hexagons that integrate the grid are regular hexagons, that is, their sides have all the same length and all internal angles are 120 degrees. To define one hexagon, we only need to specify the coordinates of its center and its radius. With this information it is trivial to obtain the coordinates of every vertex. The most important part of the hexagons are the edges, for they are associated to a transition count.

In addition, a hexagon matrix that holds all these hexagons together has been defined. This matrix represents a coordinate system for the FootSLAM maps. It is characterized by the number of columns and rows (the number of

Figure 5.1: Regular hexagon represented by its center (HexX, HexY) and its radius r.
hexagons in direction \(x\) and \(y\) respectively), the coordinates for the base of the coordinate system (the coordinates for the first hexagon), and the radius that the hexagons must have. Figure 5.2 illustrates this representation.

![Hexagon Matrix Diagram](image)

Figure 5.2: Example of a hexagon matrix with 4 columns and 6 rows, where \(r\) is the radius of the hexagons. Note that the \(y\) axis was chosen to grow downwards.

### 5.1.2 FootSLAM Maps Parameters

To generate a FootSLAM map the following key elements are needed:

- **Coordinate System**: the resulting map after walking around a certain area and using FootSLAM relies on the hexagon matrix that specifies the coordinate system. This coordinate system is used to uniquely determine the position of the pedestrian. The hexagon matrix must be large enough to ensure that the whole walk is perfectly mapped.

- **Starting Conditions**: the pedestrian starts his walk in a certain pose within the grid of hexagons. The starting point can be specified giving an estimation of the \(x\) and \(y\) coordinates, the scale factor and the heading of the pedestrian (the angle) at the beginning of the walk. As we will see, the estimation of the Starting Conditions will be crucial when merging maps.

- **Hexagon Transition Probabilities**: a FootSLAM map basically consists of an association between every hexagon of the matrix that was ever visited by a particle.
and its set of six transition counters, one for each edge. Each counter keeps a transition count that represents the number of times that the pedestrian modeled by that particle crossed the edge. These counts were learned during the execution of the FootSLAM algorithm.

5.1.3 Map Types

As was already stated, the FootSLAM technique relies on particle filters. Every particle holds, with a certain weight, its own history of the edge transitions of the hexagons it visited. These transitions allow the generation of what we call Transition Maps (see Section 3.2.3). These Transition Maps are sparse, in the sense that they do not hold all the hexagons in the hexagon grid but only the ones that were visited by the particle. They store, for each hexagon and each edge, the number of times that the particle crossed the edge, that is, they have an integer nature. But FootSLAM has only two output maps:

1. The Aggregated Transition Map: a combination of all the transition maps of all the particles that were involved in the process. This combination is achieved using the weight associated to each particle to add their maps accordingly. Another weight is also generated for the hexagons in the map, stating how probable it is that we can cross it.

2. The "MAP" Transition Map: "Maximum a Posteriori" Transition Map or Transition Map of the particle that had the highest posterior likelihood (the most probable Transition Map), which is the cumulative product of the weight of the particle by a transition likelihood term [19].

Both of these maps are a posteriori maps in the sense that they are built after the history of poses of the pedestrian $P_{0:k}$. From now on, these two types of maps will be referred to as Aggregated Map and Best Map respectively.

When merging maps, we will use the Aggregated Map, which is more complete than the Best Map in that it holds all the possible maps (hypothesis) that could result from the walk for a given number of particles. When comparing the resulting maps with the real floor plan of the building, we will use the Best Map.

There is also another type of map that will be used for map merging. It is called Prior Map and it represents the available a priori information regarding the transitions across the edges of the hexagons. Prior maps are of type Aggregated and help FootSLAM reach convergence. When the prior map is unknown, a uniform distribution is used for the prior transition counts of each one of the edges. Experiments have shown that a good value for the counter of every edge is $\alpha_e = 0.8$.

Figure 5.3 shows an example of an aggregated map and its best map.

5.1.4 The Starting Conditions

The Starting Conditions of a walk indicate the starting pose of the pedestrian at the beginning of the walk. The parameters that need to be specified are:

- **mean Starting X**: the average $x$ coordinate of the pedestrian ($\mu_x$) when starting the walk, relative to a certain coordinate system (Hexagon Matrix).
• **mean Starting Y**: the average $y$ coordinate of the pedestrian ($\mu_y$) when starting the walk, relative to a certain coordinate system (Hexagon Matrix).

• **mean Starting Angle**: the average heading angle of the pedestrian ($\mu_a$) when starting the walk. 0 degrees is North, -90 degrees is East.

• **mean Starting Scale Factor**: the average correction factor ($\mu_s$) for the size of the foot. Always very close to 1.0.

• **standard deviation for Starting X**: variation or dispersion ($\sigma_x$) from the average Starting X.

• **standard deviation for Starting Y**: variation or dispersion ($\sigma_y$) from the average Starting Y.

• **standard deviation for Starting Angle**: variation or dispersion ($\sigma_a$) from the average Starting Angle.

• **standard deviation for Starting Scale Factor**: variation or dispersion ($\sigma_s$) from the average Starting Scale Factor.

Note that the distributions used for the variables are Gaussian distributions. A smaller standard deviation indicates that we are more certain of the Starting Conditions of the pedestrian. The more accurately these starting conditions are known, the more the whole map can be expected to be placed at the right place.

The standard deviation of the variables is used by the FootSLAM algorithm to let the particles explore a wider or a narrower area at the beginning of the process, having a deep impact on how the maps look at the end of the process.

Throughout this thesis, the starting conditions of the particle with the highest weight will be referred to as "winning" Starting Conditions.

### 5.1.5 Reading the Files

The first step to manipulate a map is to generate it. The FootSLAM algorithm writes into two different text files the resulting Aggregated and Best maps. These files contain all
5.1. PREVIOUS WORK

the necessary information to build an association between the hexagons that were visited by the pedestrian and their transition counts:

- The type of map (Aggregated Map or Best Map).
- The hexagon matrix that provides a frame for the coordinate system.
- The "winning" Starting Conditions for the walk.
- A list of all the hexagons that were visited by the pedestrian, their weights and their associated counts for every edge.

Figure 5.4 shows the scheme for the construction of the map.

Figure 5.4: Schematic representation of how reading a file allows the generation of a map. The hexagons that are contained in the map are plotted in blue. The hexagon where the x and y coordinates of the Starting Conditions lie is plotted in red. The transition counts for the edges or the weight for each hexagon are not represented in this figure.

5.1.6 Map Plotting

The tool that draws the maps is called Hexagon Map Drawing Panel. The main features of the representation are:

- Only the hexagons that have some transition counts greater than 0 are plotted.
- For an Aggregated Map, the more weight a hexagon has, the darker it is plotted.
- For a Best Map the hexagons are plotted in light blue.
- The centers of the hexagons are not plotted except for the best map inside the Aggregated Map, in which case they are plotted in yellow.
- The hexagon where the starting point of the walk lies is plotted in red.

Figure 5.3 shows an example.
5.2 Steps Towards Map Merging

In order to do develop a program that automatically merges maps, the following tasks have been accomplished:

- Transform the map with a certain rotation, scale factor and shift (along x and y axes). Transform the Starting Conditions used to run FootSLAM.
- Project the map onto another grid of hexagons.
- Filter the map.
- Research a likelihood function that allows the comparison of two maps.
- Combine two given maps.
- Derive an iterative algorithm that takes two maps and finds the best combined map.

The rest of this chapter is completely dedicated to motivating the need for each one of these steps and explaining how they have been implemented in Java® for this thesis.

5.3 Map Transformation and Projection

Why transform the maps? One important feature of FootSLAM maps is that they are obtained letting the particles explore different hypothesis for the history of errors $E_{0:k}$ for the IMU sensor. As a consequence the maps are rotation invariant: we have no reference whatsoever. Furthermore, the Starting Conditions for the walk give the particles some freedom to explore the area. When FootSLAM is run several times with the same odometry data and the same Starting Conditions, the resulting maps look similar, but they are never the same in terms of transition counts or orientation. Therefore, when merging two maps, a transformation needs to be applied to one of the maps to make it match the other.

Why project the maps? After transforming a map, its hexagons will not be necessarily aligned with the hexagons of the original grid. All the maps involved in the merger need to be framed within the same coordinate system, that is, with all the hexagons aligned. Why? To be able to compare their transition counts on a edge by edge basis and combine them.

Transformation A transformation is a function that maps a set $X$ onto another set or onto itself. In the context of 2D and for our application, a transformation is the combination of a translation (or shifting) along x and y axes, a rotation and a uniform scaling.

- A translation moves every point by a fixed distance in the same direction.
5.3. Map Transformation and Projection

- A **rotation** is a transformation that is performed by "spinning" the point around a so-called center of rotation. Rotations are considered positive when anti-clockwise. The reader must note that, due to y-axis growing downwards for our application, a positive rotation is clockwise.

- **Uniform scaling** is a linear transformation that enlarges or diminishes objects. Uniform means that the scale factor is the same in all directions.

Throughout this document we will refer very often to the 4 parameters involved in the transformation as follows: rotation, scale factor, x shift, y shift.

An illustration of these transformations are shown in Figure 5.3.

![Illustration of transformations](image)

**Figure 5.5**: Illustration of the effect of translation, rotation and uniform scaling: (a) Shifting of the point with coordinates \((x, y)\), (b) Rotation the a point with coordinates \((x, y)\) using \((x_c, y_c)\) as the center for the rotation, and (c) Scaling of the square in red.

The mathematical formula used for our work can be resumed by the following equation:

\[
\begin{align*}
x_t &= ((x - x_c) \cdot s) \cdot \cos(r) - ((y - y_c) \cdot s) \cdot \sin(r) + x_c + \Delta x \\
y_t &= ((x - x_c) \cdot s) \cdot \sin(r) - ((y - y_c) \cdot s) \cdot \cos(r) + y_c + \Delta y
\end{align*}
\]

(5.1)

where \((x, y)\) are the Cartesian coordinates before the transformation, \((x_t, y_t)\) the Cartesian coordinates after the transformation, \((x_c, y_c)\) the Cartesian coordinates of the center for rotation and scaling and \(r, s, \Delta x\) and \(\Delta y\) the rotation, scale factor, x shift and y shift, respectively, that we apply to \((x, y)\).

Hence the transformation order is: scaling, rotation and then shift. It is worth mentioning that the rotation and scaling use the mean x and y coordinates of the starting point of the walk as their center, that is, \(x_c = \mu_x\) and \(y_c = \mu_y\).

**Projection** After transforming every point, the transformed point is projected onto a so-called "target grid of hexagons", which serves as a common coordinate system for all the maps that will be considered. The projection has been illustrated in Figure 5.6.

The **projection** involves two steps: the projection of the vertices of every edge of the map and the projection of the counts of every edge. The projection of the vertices is trivial since it is done from a 2D surface (a plane) to another 2D surface that is parallel to the first, and hence the coordinates for the transformed and projected vertices are the same. The projection of the edges is more complex and will be explained later on.
Figure 5.6: Illustration of the projection of the map in Figure 5.3.

The transformation and projection of a map is performed in a hexagon per hexagon basis, that is, only one hexagon is taken at a time and the following tasks are undertaken:

1. **Scaling:** the hexagon is scaled by multiplying its radius by the scale factor as shown on the left side of Figure 5.7. Then each of the edges of the scaled hexagon is rotated and shifted as shown on the right side of Figure 5.7.

2. **Projection of the edge:** each transformed edge is then projected onto the target grid of hexagons by projecting its two vertices (projection of the vertices A and B) as can be seen in Figure 5.8.

3. **Projection of the counts:** the transition counts (from now on referred to as C) are shared among some of the edges of the target grid. To do that, first the *target hexagons* (which hexagons of the target grid will receive any counts) have
5.3. MAP TRANSFORMATION AND PROJECTION

Figure 5.8: Projected edge 0 of a hexagon that has been transformed. Points A and B are the vertices of edge 0.

been defined: the two hexagons where the two vertices of the transformed edge lie (points A and B) along with all their neighboring hexagons. See Figure 5.9.

Once the target hexagons are defined, C are shared among their edges. To see how much of C each target edge receives, two different factors have been defined: the distance factor and the angular factor. The distance factor takes into account the distance between a transformed hexagon and a target hexagon and the angular factor takes into account the relative orientation between a transformed edge and a target edge. These factors are simply a weight that state how much they have in common with the transformed edge.

5.3.1 Angular Factor

Throughout the work of this thesis, the way FootSLAM operates with respect to how the particles explore the area around them (the angle) changed, to reduce the occurrence of favored angles in the hexagon matrix. Here we present the two approaches that were implemented for the projection. The first one only computes the angular factor for two edges and worked perfectly with the first implementation of FootSLAM. The second one computes the angular factor for the six edges and needed to be implemented to work more accordingly to the second version of FootSLAM.

After explaining how these two angular factors are computed we will illustrate why each one was more convenient for the FootSLAM version at the time.
Figure 5.9: Target hexagons for transformed edge 0 (in red): hexagons A (in purple) and B (in green) where the two vertices of the transformed edge lie and their neighbors (in blue).

**Angular factor with two edges**  The first step is to compute onto which two edges of the target hexagons the transition counts $C$ of each transformed edge will be projected. The criteria to choose the edges relies on how much orientation the transformed edge and the edges of the target hexagons have in common. The two edges with closer orientation will be chosen.

We have to divide the rotation value $r$ applied to the map by $60^\circ$ ($\pi/3$ radians) to calculate how many edges we have "traveled" (integer part of the division):

$$\text{traveled edges} = \text{floor}\left(\frac{r}{\pi/3}\right), \quad (5.2)$$

and obtain the two target edge indexes $e_{tg}^{first}$ and $e_{tg}^{second}$ where the counts of each transformed edge $e_{tr}$ need to be projected onto:

$$\text{target edges} = \begin{cases} 
    e_{tg}^{first} = e_{tr} + \text{traveled edges} \\
    e_{tg}^{second} = \text{mod}_6(e_{tg}^{first} + 1) 
\end{cases} \quad (5.3)$$

The angular factor rewards those target edges that have a closer orientation to that of the transformed edge $e_{tr}$. To do so, the remainder from the division of the rotation by $60^\circ$ can be computed as:

$$\text{fractional remainder} = r - \text{traveled edges} \cdot \pi/3. \quad (5.4)$$
And now a weight $w$ for each edge can be calculated as follows:

$$w^{ang2}(e_{tr}) = \begin{cases} w^{first} = 1 - \text{fractional remainder}/(\pi/3) \\ w^{second} = 1 - w^{first}. \end{cases}$$  \hfill (5.5)

See Figure 5.10 for a graphical illustration of the computation of the traveled edges and fractional remainder.

Figure 5.10: Illustration of how the traveled edges and the fractional remainder are computed. The number of traveled edges is the number of blue triangles. The fractional remainder is the part that needs to be added to complete the rotation. Note that a positive rotation is clockwise due to the y axis growing downwards.

Operating in this manner, the sum of the two weights add to 1, that is, they are normalized. See Figure 5.11 with an example for the transformed edge of Figure 5.9.

**Angular factor with six edges** The previous angular factor does not account for the fact that the pedestrian might have crossed a certain edge with a very wide range of angles with respect to it, as shown in Figure 5.12. The arrows represent possible paths or trajectories the pedestrian might have followed when crossing the edge. This idea is used to draw a semicircle on the *outer* part of the edge, symbolizing the probability of that arrow being the path taken by the pedestrian.

The transition counts $C$ of each transformed edge are now shared among the six edges of the target hexagons. Let’s consider the transformed edge of Figure 5.9 for our explanation. Figure 5.13 illustrates the projection of that transformed edge onto the six edges of one of the target hexagons. The center of the transformed edge has been made coincide with the center of the six target edges. Note that in this figure, both transformed and target edge have been drawn with the same size. As will be shown, the angular factor does not depend on their relative sizes (scale factor).

The alternative representation with the semicircles is very useful to compute the angular factor in a geometric form, considering how much the transformed edge's possible
CHAPTER 5. TOWARDS MAP Merging

Figure 5.11: Example of transformed edge and all the target edges onto which its transitions counts will be projected onto when using the angular factor with two edges.

Figure 5.12: Example of transformed edge and some possible paths that the pedestrian might have taken when crossing it (the arrows) and its alternative representation for the arrows, a semicircle on the outer part of the edge. The semicircle represents the probability with which the pedestrian crossed that edge with the different orientations.

paths overlap with the target edge’s possible paths. Let’s focus on just one target edge, the one shown on the right side of Figure 5.13. We need to compute the overlapping area between the semicircle of the transformed edge $e_{tr}$ (in yellow) and the semicircle of the target edge $e_{tg}$ (in blue). This overlapping area is shown in green.
5.3. MAP TRANSFORMATION AND PROJECTION

The overlapping area can be calculated using the following formula:

\[
\text{Overlapping Area}(e_{tr}, e_{tg}) = \int_{\rho=0}^{\rho=R/2} \int_{\theta=\theta_{e_{tr}, e_{tg}}^{\text{tr}}}^{\theta=\theta_{e_{tr}, e_{tg}}^{\text{tg}}} \rho \, d\rho \, d\theta = (\theta_{e_{tr}, e_{tg}}^{\text{tr}} - \theta_{e_{tr}, e_{tg}}^{\text{tg}}) \cdot \frac{R^2}{8}, \quad (5.6)
\]

where \( R = \min\{R_{tg}, R_{tr}\} \) and \( R_{tg} \) is the radius of the hexagons of the target grid of hexagons where we project onto and \( R_{tr} \) is the radius of the hexagons of the source map after the transformation. Now we can normalize this area by the total overlapping area, that is, three times the area of a semicircle of radius \( R/2 \):

\[
\text{Normalized Area}(e_{tr}, e_{tg}) = \frac{(\theta_{e_{tr}, e_{tg}}^{\text{tr}} - \theta_{e_{tr}, e_{tg}}^{\text{tg}})}{3 \cdot \pi \left(\frac{R}{2}\right)^2 \cdot \frac{1}{2}} = \frac{(\theta_{e_{tr}, e_{tg}}^{\text{tr}} - \theta_{e_{tr}, e_{tg}}^{\text{tg}})}{3 \pi} \cdot \frac{R^2}{8}. \quad (5.7)
\]

Then these factors add up to one.

As can be seen in Figure 5.13, \( \theta_{e_{tr}, e_{tg}}^{\text{tr}} \) and \( \theta_{e_{tr}, e_{tg}}^{\text{tg}} \) depend on which transformed index \( e_{tr} \) is going to be projected onto which target index \( e_{tg} \). To cope with this, a simple formula to compute \((\theta_{e_{tr}, e_{tg}}^{\text{tr}} - \theta_{e_{tr}, e_{tg}}^{\text{tg}})\) has been introduced:

\[
(\theta_{e_{tr}, e_{tg}}^{\text{tr}} - \theta_{e_{tr}, e_{tg}}^{\text{tg}}) = \pi \pm r - \text{steps}_{e_{tr}, e_{tg}} \cdot \frac{\pi}{3}, \quad (5.8)
\]

where \( r \) is the rotation value that was applied to the edge and \( \text{steps}_{e_{tr}, e_{tg}} \) represents the shortest distance, measured in edges, between the transformed edge index and the target edge index. For example: transformed edge 0 and target edge 4 are 2 edges apart, transformed edge 0 and target edge 3 are 3 edges apart. Positive sign + will be used when the target edge index is the source index plus 1, 2 or 3 and - for the other 3 cases (plus 4, 5 or 0), always modulo 6.

### Figure 5.13: Example of transformed edge (in red) that has been made coincide with the six target edges (in black) of a hexagon and how their corresponding semicircles overlap (in green).
The performance when using these angular factors showed that more weight was needed for those target edges that had more orientation in common with the transformed edge. We simply power the normalized area and renormalize as follows:

$$w_{\text{ang}}(e_{\text{tr}}, e_{\text{tg}}) = \frac{\text{Normalized Area}(e_{\text{tr}}, e_{\text{tg}})^{\text{strength}}}{\sum_{e \in h_{\text{tg}}} \text{Normalized Area}(e_{\text{tr}}, e)^{\text{strength}}},$$

(5.9)

where $h_{\text{tg}}$ represents the target hexagon to which the target edge $e_{\text{tg}}$ belongs.

**Why change the angular factor** When the FootSLAM version was changed to account for possible deviations from a straight line, the insufficiency of the previous angular factor was revealed. An example shown in Figure 5.14. Edge 0 of a given hexagon has been scaled and shifted, but not rotated. As a result of using the former angular factor, only edge index 0 of the target hexagons will receive a portion of the counts (with weight 1), but never indexes 1, 2, 4 or 5. It is quite obvious that also those edges need to receive some share of the counts.

![Figure 5.14: Example of a transformed edge and the edges that also need to receive counts.](image)

**5.3.2 Distance Factor**

The distance factor rewards those target hexagons that are closer in distance to the transformed hexagon. All the edges of each target hexagon are assigned the same weight.
The distance factor is computed as follows:

\[ d(h_{tr}, h_{tg}) = \sqrt{(x_{h_{tr}} - x_{h_{tg}})^2 + (y_{h_{tr}} - y_{h_{tg}})^2}, \]

where \((x_{h_{tr}}, y_{h_{tr}})\) represent the x and y coordinates of the center of the transformed hexagon and \((x_{h_{tg}}, y_{h_{tg}})\) the x and y coordinates of the center of the target hexagon. So the distance is computed as the Euclidean distance between the two centers of the hexagons being considered at the time.

With this distance factor a weight can be computed:

\[ w^{\text{dist}}(h_{tr}, h_{tg}) = \frac{1}{d(h_{tr}, h_{tg})^{\text{strength}}} \sum_{h_{tg}^k \in H_{\text{Target}}} \frac{1}{d(h_{tr}, h_{tg}^k)^{\text{strength}}}, \]

where \(h_{tg}^k\) goes through all the target hexagons onto which the transformed edge was projected to normalize the weight. See Figure 5.15.

### 5.3.3 Transformed Map

To obtain a transformed map from a source map, the procedure explained in the previous sections must be applied to every single edge of the source map.

For every source edge, the corresponding weights for its associated target edges are computed. Next, for each target edge the source counts \(C\) are multiplied by its two weights:
\[ C \cdot w_{\text{ang}}(e_{tr}, e_{tg}) \cdot w_{\text{dist}}(h_{tr}, h_{tg}). \]  

(5.12)

This represents the share of \( C \) that each target edge will receive from the source edge. But some target edges will receive counts from more than one source edge. To account for this, the computed new transition counts are added to the current transition counts for that target edge.

In the end an Aggregated Map is obtained and can be manipulated as if it was a fresh FootSLAM map, in that it can be stored in and read from a file and transformed and projected again.

The only drawback for transforming maps is that the maps lose some accuracy, mostly due to the underlying grid of hexagons and the approximations used for the angular and distance factors. To illustrate this, the reader must think of the case in which a map is simply projected onto its own grid of hexagons. Even when the source map hexagons are aligned with the underlying grid, the counts are somehow spread.

**Limitations for transformation**  For our approach there is one limitation when scaling the map. The scale factor cannot be greater than 4 times the radius of the target grid of hexagons. This is because of the way the *target hexagons* have been defined, that is, the neighbors of the hexagons where the two vertices of every edge lie. See Figure 5.16 for more details.

![Figure 5.16: Illustration of the scale limitation](image-url)

Figure 5.16: Illustration of the scale limitation: a source hexagon (transparent) has been scaled with a factor greater than four times the radius of the target grid of hexagons. The target hexagons that the current version of the projection takes into account are hexagons A (in purple) and B (in green) and their neighbors (in blue). The hexagons that should also be taken into account are plotted in orange.
5.3. MAP TRANSFORMATION AND PROJECTION

5.3.4 Transforming the Starting Conditions

Another possibility that can be used to transform a map is to directly transform its Starting Conditions and run FootSLAM with the new transformed Starting Conditions. The transformation of the Starting Conditions is achieved by performing the following computation:

\[
\begin{align*}
\mu_{xt} &= ((\mu_X - x_c) \cdot s) \cdot \cos(r) - ((\mu_y - y_c) \cdot s) \cdot \sin(r) + x_c + \Delta_x \\
\mu_{yt} &= ((\mu_X - x_c) \cdot s) \cdot \sin(r) + ((\mu_y - y_c) \cdot s) \cdot \cos(r) + y_c + \Delta_y \\
\mu_{st} &= \mu_s \cdot s \\
\mu_{at} &= \mu_a + r,
\end{align*}
\] (5.13)

where the under-script \( t \) means transformed and \( \Delta_x, \Delta_y, r \) and \( s \) are the parameters for the transformation and \((x_c, y_c)\) the coordinates of the center for the rotation and scaling.

This transformation makes the FootSLAM algorithm start at a different pose determined by the new average Starting Conditions. The standard deviations \( \sigma \) remain unchanged, meaning that the uncertainty does not grow or decrease when we modify the average Starting Conditions.

The advantage of transforming the Starting Conditions and then running FootSLAM over transforming a map is that, even though both are similarly disposed in the space, the first option does not involve the projection process and hence no accuracy is lost.

5.3.5 Inverse Transformation

The inverse transformation \( inverse(T) = T^I \) of a given transformation \( T \) is a transformation such that, if \( T \) was applied to a map \( M \) to obtain the transformed map \( M^T \), \( T^I \) can be applied to \( M^T \) to obtain \( M \). Then, an inverse transformation \( T^I \) is a transformation that can reverse the effect of a given transformation.

Since a transformation \( T \) is composed, in this order, by a uniform scaling, a rotation and a shift along the \( x \) and \( y \) axes, a series of three transformations that reverse the effect of them in opposite order is:

1. A transformation \( T_1 \) with \( r_1 = 0, \Delta x_1 = -\Delta x, \Delta y_1 = -\Delta y, s_1 = 1 \).
2. A transformation \( T_2 \) with \( r_2 = -r, \Delta x_2 = 0, \Delta y_2 = 0, s_2 = 1 \).
3. A transformation \( T_3 \) with \( r_3 = 0, \Delta x_3 = 0, \Delta y_3 = 0, s_3 = 1/s \).

Then applying equation 5.1 to each point for the transformed map with coordinates \((x^T, y^T)\) successively for the three transformations:

\[
\begin{align*}
x_1 &= ((x^T - x_c) \cdot 1) \cdot \cos(0) - ((y^T - y_c) \cdot 1) \cdot \sin(0) + x_c - \Delta x = x^T - \Delta x \\
y_1 &= ((x^T - x_c) \cdot 1) \cdot \sin(0) - ((y^T - y_c) \cdot 1) \cdot \cos(0) + y_c - \Delta y = y^T - \Delta y
\end{align*}
\]

Now apply the second transformation:

\[
\begin{align*}
x_2 &= ((x_1 - x_c) \cdot 1) \cdot \cos(-r) - ((y_1 - y_c) \cdot 1) \cdot \sin(-r) + x_c + 0 \\
y_2 &= ((x_1 - x_c) \cdot 1) \cdot \sin(-r) - ((y_1 - y_c) \cdot 1) \cdot \cos(-r) + y_c + 0 \\
&= (x^T - \Delta x - x_c) \cdot \cos(-r) - (y^T - \Delta y - y_c) \cdot \sin(-r) + x_c \\
&\quad (x^T - \Delta x - x_c) \cdot \sin(-r) - (y^T - \Delta y - y_c) \cdot \cos(-r) + y_c
\end{align*}
\]
And the third:

\[
\begin{align*}
x_3 &= ((x_2 - x_c) \cdot 1/s) \cdot \cos(0) - ((y_2 - y_c) \cdot 1/s) \cdot \sin(0) + x_c + 0 \\
y_3 &= ((x_2 - x_c) \cdot 1/s) \cdot \sin(0) - ((y_2 - y_c) \cdot 1/s) \cdot \cos(0) + y_c + 0 \\
\end{align*}
\]

\[
\begin{align*}
x_3 &= (((x^T - \Delta x - x_c) \cdot \cos(-r) - (y^T - \Delta y - y_c) \cdot \sin(-r) + x_c - x) \cdot 1/s) + x_c \\
y_3 &= (((x^T - \Delta x - x_c) \cdot \sin(-r) - (y^T - \Delta y - y_c) \cdot \cos(-r) + y_c - y) \cdot 1/s) + y_c \\
\end{align*}
\]

\[
\begin{align*}
x_3 &= ((x^T - x_c) \cdot \cos(-r) - (y^T - y_c) \cdot \sin(-r)) / s + (-\Delta x \cdot \cos(-r) + \Delta y \cdot \sin(-r)) / s + x_c \\
y_3 &= ((x^T - x_c) \cdot \sin(-r) - (y^T - y_c) \cdot \cos(-r)) / s + (-\Delta x \cdot \sin(-r) - \Delta y \cdot \cos(-r)) / s + y_c \\
\end{align*}
\]

But \((x_3, y_3)\) must equal \((x, y)\) of equation 5.1. Then we can extract, by comparison, the parameters for the inverse transformation:

\[
\begin{align*}
\Delta^I_x &= (-\Delta x \cdot \cos(-r) + \Delta y \cdot \sin(-r)) / s \\
\Delta^I_y &= -(-\Delta x \cdot \sin(-r) \Delta y \cdot \cos(-r)) / s \\
r^I &= -r \\
s^I &= \frac{1}{s}
\end{align*}
\]

(5.14)

The important fact here is that the center for the rotation and the scaling must be the same for both transformation, that is: \((x_c, y_c)^I = (x_c, y_c)\).

### 5.4 Map Filtering

The meaning of filtering in the context of maps is that of using a filter factor to spread the counts of the edges. The filter factor is used to indicate the percentage of the count that remains in the same edge. The rest is shared among certain edges in the surroundings.

Map filtering is used for prior maps in the FootSLAM process, to leave more hypothesis open for the exploration of the area by the particles. Two types of filtering have been defined: angular and distance filtering, which are explained later on.

#### 5.4.1 Distance Filtering

The distance filter takes every edge’s associated transition counts \(C\) and shares them among the same edge index for the 6 neighboring hexagons and itself. The filter factor indicates how much of \(C\) stays in the same edge. The rest is divided by 6 and given to the same edge in the neighboring hexagons. See Figure 5.17.

#### 5.4.2 Angular Filtering

The angular filter takes every edge’s associated transition counts \(C\) and shares them among itself and the previous and next edge of the same hexagon. The filter factor defines how much of \(C\) stays in the same edge. The rest is divided by 2 and given to the previous and following edges. See Figure 5.18.
5.5 Examples

In Figure 5.5 real examples of a FootSLAM map that has been transformed and filtered are shown. The strength for the angular factor was chosen to be 5.0, and the strength for the distance factor, 2.0.

The reader might want to account for the lose of accuracy (translated into blurriness of the map) that the map experiments when it is transformed, as was stated before.

5.6 Correlating Maps

In statistics, the correlation between two random variables refers to their dependence or statistical relationship. In the context of FootSLAM maps a measure of how much one map looks like another map is needed. To this purpose, an appropriate correlation
CHAPTER 5. TOWARDS MAP MERGING

(a) Original FootSLAM map.

(b) Rotated 30 degrees.

(c) Scaled with a factor of 1.15.

(d) Distance filtered with a factor of 0.8.

Figure 5.19: Examples of transformations of a FootSLAM map: (a) The original aggregated map resulting from running FootSLAM, (b) The map after being rotated 30 degrees, (c) The map after being scaled with a scale factor of 1.15 and (d) The map after being (distance) filtered with a factor of 0.8.

function that depends on how well two maps fit each other has been derived. Two types of maps have been defined in terms of which one is transformed to fit the other:

- **"Underneath" Map:** This is the map that will stay fixed through the comparison, as a reference for the other map. It will be also referred to as $M^U$.

- **"Accounted for" Map:** This is the map that will be transformed to fit the underneath map and then projected onto the same target grid of hexagons. From now on it will be referred to as "Accounted Map" or alternatively as $M^A$. We will use the term "transformed map" to refer to this map when it has been transformed.

This distinction is important when computing the asymmetric correlation, as will be explained next. The Underneath Map is chosen to be the most certain in terms of heading of the pedestrian at the starting point (smallest standard deviation for the angle, $\sigma_a$). When both maps are equally accurate, their certainty in the X and Y variables is checked, applying the same criteria. When both maps have the same standard deviation for all variables, the Underneath Map is chosen randomly.
5.6. CORRELATING MAPS

5.6.1 Likelihood Function

Derivation Our derivation relies on the DBN shown in Figure 4.1. Basically we have two walks, and hence two histories of two pedestrian’s poses, \( P_{0:k}^1 \) and \( P_{0:k}^2 \). We are interested in finding the transformation that will align the two histories of poses. To do so, we will compute the posterior density function of a transformation \( T \) conditioned on the poses: \( p(T|P_{0:k}^1, P_{0:k}^2) \). The transformation \( T \) transforms the poses \( P_{0:k}^i \) onto the map \( M \). We are assuming that \( P_{0:k}^2 \) is aligned with the map \( M \).

The first step is to include the map in our derivation:

\[
p(T|P_{0:k}^1, P_{0:k}^2) = \int_M p(T|P_{0:k}^1, P_{0:k}^2, M) \cdot p(M|P_{0:k}^1, P_{0:k}^2) dM = \mathcal{I}, \tag{5.15}
\]

but using Bayes rule:

\[
p(T|P_{0:k}^1, P_{0:k}^2, M) = \frac{p(P_{0:k}^1|T, P_{0:k}^2, M) \cdot p(T|P_{0:k}^1, P_{0:k}^2, M)}{p(P_{0:k}^1|P_{0:k}^2, M)}
\]

and

\[
p(M|P_{0:k}^1, P_{0:k}^2) = \frac{p(P_{0:k}^1|P_{0:k}^2, M) \cdot p(P_{0:k}^2)}{p(P_{0:k}^1)}
\]

so (5.15) equals:

\[
I = \int_M p(P_{0:k}^1|T, P_{0:k}^2, M) \cdot p(T|P_{0:k}^2, M) \cdot \frac{p(M|P_{0:k}^2)}{p(P_{0:k}^1|P_{0:k}^2)} dM \tag{5.16}
\]

Now if we assume that \( M \) is in the frame of the poses of the second walk, \( P_{0:k}^2 \), then \( p(T|P_{0:k}^2, M) = p(T) \) and \( p(P_{0:k}^1|T, P_{0:k}^2, M) = p(P_{0:k}^1|T, M) \).

So we are left with:

\[
I = \frac{p(T)}{p(P_{0:k}^1|P_{0:k}^2)} \cdot \int_M p(P_{0:k}^1|T, M) \cdot p(M|P_{0:k}^2) dM. \tag{5.17}
\]

Now we can use the chain rule to obtain \( p(P_{0:k}^1|T, M) = \prod_{k=0}^{k} p(p_{1:k}|p_{1:k-1}, T, M) \) and we assume that we can factor the map into local and conditionally independent components:

\[
p(M|P_{0:k}^2) = \prod_{h \in P^2} p(M_{h}^2|P_{0:k}^2),
\]

and then

\[
p(T|P_{0:k}^1, P_{0:k}^2) = \frac{p(T)}{p(P_{0:k}^1|P_{0:k}^2)} \prod_{k=0}^{k} \int_M p(P_{1:k}^1|P_{1:k-1}, T, M_{h}^1) \cdot \prod_{h \in P^2} p(M_{h}^2|P_{0:k}^2) dM \tag{5.18}
\]

so

\[
I \propto \prod_{k=0}^{k} \int_{M_0} \cdots \int_{M_{N-1}} p(P_{1:k}^1|P_{1:k-1}, T, M_{h}^1) \cdot \prod_{h \in P^2} p(M_{h}^2|P_{0:k}^2) dM
\]

\[
= \prod_{k=0}^{k} \left[ \prod_{h \in P^2} \int_{M_{h} \in P^1} \int_{M_{h} \in P^2} p(M_{h}^2|P_{0:k}^2) d(M_{h} \in P^2) \right] \cdot \int_{M_{h} \in P^1} p(P_{1:k}^1|P_{1:k-1}, T, M_{h}^1 + M_{h}^2) \cdot p(M_{h}^2|P_{0:k}^2) d(M_{h} \in P^1). \tag{5.19}
\]
The integral for the hexagons in $\mathbb{P}^2$ all equate to unity. We are left with the integral over $M_{h\in \mathbb{P}^1}$.

We can now introduce the fact that
\[ p(\mathbf{P}^1_{kl} | \mathbf{P}^1_{kl-1}, \mathbf{T}, M^2_h + M^2_{\tilde{h}}) = p(\mathbf{P}^1_{kl} | \mathbf{P}^1_{kl-1}, \mathbf{T}, M^2_h + M^2_{\tilde{h}}) \]
since only the edge $\tilde{e} = e(U_k)$ determines the transition probability and
\[ p(\mathbf{P}^1_{kl} | \mathbf{P}^1_{kl-1}, \mathbf{T}, M^2_h + M^2_{\tilde{h}}) \alpha M^2_T = T(M^2_h) + M^2_{\tilde{h}} \]
with $T = \tilde{T}$ unknown.

Now we can expand $M_{\tilde{h}}$ into the individual transitions $e$:
\[
I \propto \prod_{k=0}^K \int_{M_{h\in \mathbb{P}^1}^0} \cdots \int_{M_{h\in \mathbb{P}^1}^K} (T(M^2_h) + M^2_{\tilde{h}}) p(M^2_h | \mathbf{P}^2_{0:k}) d(M_{h\in \mathbb{P}^1}^1)
\]
\[
= \prod_{k=0}^K \int_{M_{h\in \mathbb{P}^1}^0} \cdots \int_{M_{h\in \mathbb{P}^1}^K} (T(M^2_h) + M^2_{\tilde{h}}) \int_{M_{h\in \mathbb{P}^1}^0} \cdots \int_{M_{h\in \mathbb{P}^1}^K} p(M^2_h, \cdots, M^2_{\tilde{h}} | \mathbf{P}^2_{0:k}) d(M_{h\in \mathbb{P}^1}^1).
\]
\[ (5.20) \]

Since the right hand integral marginalizes over all other edges of our outgoing hexagon we obtain:
\[
I \propto \prod_{k=0}^K \int_{M_{h\in \mathbb{P}^1}^0} \cdots \int_{M_{h\in \mathbb{P}^1}^K} (T(M^2_h) + M^2_{\tilde{h}}) p(M^2_h | \mathbf{P}^2_{0:k-1}) d(M_{h\in \mathbb{P}^1}^1)
\]
\[
= \prod_{k=0}^K \int_{M_{h\in \mathbb{P}^1}^0} \cdots \int_{M_{h\in \mathbb{P}^1}^K} T(M^2_h) p(M^2_h | \mathbf{P}^2_{0:k-1}) d(M_{h\in \mathbb{P}^1}^1) + \prod_{k=0}^K \int_{M_{h\in \mathbb{P}^1}^0} M^2_{\tilde{h}} p(M^2_{\tilde{h}} | \mathbf{P}^2_{0:k-1}) d(M_{h\in \mathbb{P}^1}^1)
\]
\[
= \prod_{k=0}^K \int_{M_{h\in \mathbb{P}^1}^0} \cdots \int_{M_{h\in \mathbb{P}^1}^K} T(M^2_h) p(M^2_h | \mathbf{P}^2_{0:k-1}) d(M_{h\in \mathbb{P}^1}^1) + K
\]
\[
= \prod_{k=0}^K \int_{M_{h\in \mathbb{P}^1}^0} \cdots \int_{M_{h\in \mathbb{P}^1}^K} \left( \frac{C_{\tilde{h}}^2 + \alpha_{\tilde{h}}^2}{C_{h}^2 + \alpha_{h}^2} \right) T(C_{\tilde{h}}^1) + K,
\]
\[ (5.20) \]

where $K$ is a constant, $C_{\tilde{h}}^2$ represents the counts for edge $\tilde{e}$ in hexagon $\tilde{h}$ in map 2, $C_{h}^2$ the total counts for hexagon $h$, $\alpha_{\tilde{h}}^2$ the prior knowledge we have regarding the counts across edge $\tilde{e}$ of hexagon $\tilde{h}$, usually chosen 0.8 $\forall \tilde{e}$ and $\alpha_h = \sum_{\tilde{e}=0}^{\tilde{e}=5} \alpha_{\tilde{h}}$.

$T(C_{\tilde{h}}^1)$ represents the counts for edge $\tilde{e}$ of hexagon $\tilde{h}$ after transforming the poses $\mathbf{P}^1_{0:k}$ with transformation $T$. So we can use
\[
\prod_{h\in \mathbb{P}^1} \prod_{\tilde{e}=0}^{\tilde{e}=5} \left( \frac{C_{\tilde{h}}^2 + \alpha_{\tilde{h}}^2}{C_{h}^2 + \alpha_{h}^2} \right) T(C_{\tilde{h}}^1)
\]
\[ (5.22) \]

to compute how well $\mathbf{P}^1_{0:k}$ matches $\mathbf{P}^2_{0:k}$. What we will do is try different possibilities for the transformation $T$ and compute the correlation value for each one of them.
The logarithmic form for the likelihood function  
So far we have been able to obtain a formula that states how well one map matches another. We can extend (5.22) to its symmetric form, that is, also taking into account how well map 2 matches 1. But for our purposes we will call the maps $M_A$ and $M_U$, which stand for Accounted Map and Underneath Map. The likelihood value ($LV$) between two maps can be computed as:

$$LV(M^A, M^U) = \prod_{h \in A} \prod_{e=0}^{e=5} \left( \frac{C^e_h}{C^e_h + \alpha_h} \right) \cdot \prod_{h \in U} \prod_{e=0}^{e=5} \left( \frac{C^e_h + \alpha_e}{C^A_h + \alpha_h} \cdot \frac{C^U_h}{C^A_h + \alpha_h} \right). \quad (5.23)$$

We have chosen to implement this function in its logarithmic form, since it is less expensive in terms of computational requirements:

$$logLV(M^A, M^U) = \log \left[ \prod_{h \in A} \prod_{e=0}^{e=5} \left( \frac{C^e_h + \alpha_e}{C^e_h + \alpha_h} \right) \cdot \prod_{h \in U} \prod_{e=0}^{e=5} \left( \frac{C^e_h + \alpha_e}{C^A_h + \alpha_h} \cdot \frac{C^U_h}{C^A_h + \alpha_h} \right) \right]$$

$$= \sum_{h \in A} \sum_{e=0}^{e=5} C^A_h \cdot \log \left( \frac{C^e_h + \alpha_e}{C^e_h + \alpha_h} \right) + \sum_{h \in U} \sum_{e=0}^{e=5} C^U_h \cdot \log \left( \frac{C^e_h + \alpha_e}{C^A_h + \alpha_h} \right). \quad (5.24)$$

We want to introduce now the different versions for the probabilistic and logarithmic forms of the correlation function in (5.23) and (5.24), that is, the asymmetric and normalized formulas.

Asymmetric likelihood function  
When we want to compute how well the Accounted Map matches the Underneath Map, we use:

- Probabilistic form
  $$LV_A(M^A, M^U) = \prod_{h \in A} \prod_{e=0}^{e=5} \left( \frac{C^e_h + \alpha_e}{C^e_h + \alpha_h} \right). \quad (5.25)$$

- Logarithmic form
  $$logLV_A(M^A, M^U) = \sum_{h \in A} \sum_{e=0}^{e=5} C^A_h \cdot \log \left( \frac{C^e_h + \alpha_e}{C^e_h + \alpha_h} \right). \quad (5.26)$$

And when we want to compute how well the Underneath Map matches the Accounted Map, we use:

- Probabilistic form
  $$LV_U(M^A, M^U) = \prod_{h \in U} \prod_{e=0}^{e=5} \left( \frac{C^e_h + \alpha_e}{C^A_h + \alpha_h} \cdot \frac{C^U_h}{C^A_h + \alpha_h} \right). \quad (5.27)$$

- Logarithmic form
  $$logLV_U(M^A, M^U) = \sum_{h \in U} \sum_{e=0}^{e=5} C^U_h \cdot \log \left( \frac{C^e_h + \alpha_e}{C^A_h + \alpha_h} \right). \quad (5.28)$$
Normalized likelihood value  We can now normalize (5.25) and (5.27) by the total counts in the map:

\[
LV_{A}^{\text{norm}}(M^A, M^U) = \prod_{h \in A} \prod_{e=0}^{e=5} \left( \frac{C_{eh}^A}{C_{eh} + \alpha_h} \right)^{C_{eh}^A} \sum_{h \in A} \sum_{e=0}^{e=5} C_{eh}^A 
\]  (5.29)

and

\[
LV_{U}^{\text{norm}}(M^A, M^U) = \prod_{h \in U} \prod_{e=0}^{e=5} \left( \frac{C_{eh}^U}{C_{eh} + \alpha_T} \right)^{C_{eh}^U} \sum_{h \in U} \sum_{e=0}^{e=5} C_{eh}^U 
\]  (5.30)

And their logarithmic forms:

\[
\log LV_{A}^{\text{norm}}(M^A, M^U) = \frac{\sum_{h \in A} \sum_{e=0}^{e=5} C_{eh}^A \cdot \log \left( \frac{C_{eh}^A}{C_{eh} + \alpha_h} \right)}{\sum_{h \in A} \sum_{e=0}^{e=5} C_{eh}^A} 
\]  (5.31)

and

\[
\log LV_{U}^{\text{norm}}(M^A, M^U) = \frac{\sum_{h \in U} \sum_{e=0}^{e=5} C_{eh}^U \cdot \log \left( \frac{C_{eh}^U}{C_{eh} + \alpha_T} \right)}{\sum_{h \in U} \sum_{e=0}^{e=5} C_{eh}^U} 
\]  (5.32)

Properties  Some of the properties that apply to the likelihood formulas in both probabilistic and logarithmic form are now presented:

- **Symmetry**  \(LV(M^1, M^2) = LV(M^2, M^1)\) and \(\log LV(M^1, M^2) = \log LV(M^1, M^2)\) only when we use the symmetric version (equations 5.23 and 5.24).

- **Linearity**  \(\log LV_M(k \cdot M^1, M^2) = k \cdot \log LV_M(M^1, M^2)\) where \(k \cdot M^1\) means that all counts of map \(M^1\) are multiplied by \(k\). Note that we need to use the asymmetric unnormalized version.

- **Decomposability**  \(\log LV_M(M^1, M^2) = \sum_{h \in M^1} \log LV_M(M^1_h, M^2_h)\) where \(M_h\) represents the transition map for hexagon \(h\). Likewise \(\log LV(M^1, M^2) = \sum_{e=0}^{e=5} \log LV(M^1_e, M^2_e)\) is the correlation per hexagon and \(\log LV(M^1, M^2) = \log \left( \frac{C_{eh}^2}{C_{eh} + \alpha_e} \right)\) is the correlation per edge, representing \(M_e\) the edge transition map for edge \(e\).

- **Proportionality to the total counts in the maps**  Only the normalized formulas are inversely proportional to the total counts in the map.
• **Contribution when both maps are empty** \( \log LV(\emptyset, \emptyset) = 0 \) in the logarithmic domain and \( LV(\emptyset, \emptyset) = 1 \) in the probability domain, that is, their contribution is null.

• **Contribution when the Underneath Map is empty** \( \log LV_A^{\text{norm}}(M^A, \emptyset) = \log \left( \frac{a_n}{a_h} \right) = \log \left( \frac{1}{6} \right) = -0.778 \). This value is very important to set a threshold between good correlation values and bad correlation values.

**Hexagon Correlation Factor** The likelihood value formula is completed by adding a heuristic term that takes into account the correlation per hexagon and that increases the robustness of the likelihood function:

\[
HCT = \beta \cdot \sum_{h \in A} C_h^A \cdot C_h^U,
\]

where \( HCT \) means Hexagon Correlation Term.

And normalized for both maps:

\[
HCT_A = \frac{\beta \cdot \sum_{h \in A} C_h^A \cdot C_h^U}{\sum_{e=5}^{e=0} \sum_{h \in A} C_h^A U}.
\]

and

\[
HCT_U = \frac{\beta \cdot \sum_{h \in U} C_h^A \cdot C_h^U}{\sum_{e=5}^{e=0} \sum_{h \in U} C_h^U}.
\]

where \( \beta \) is known as Hexagon Correlation Factor and in our experiments, it was chosen to be 0.04.

Finally we obtain the symmetric augmented likelihood value by putting everything together:

\[
\text{Aug-log}LV^{\text{norm}}(M^A, M^U) = \frac{\sum_{h \in A} \sum_{e=0}^{e=5} C_{h,e}^A \cdot \log \left( \frac{C_{h,e}^U + \alpha_e}{C_{h,e}^A + \alpha_T} \right) + \beta \cdot \sum_{h \in A} C_h^A \cdot C_h^U}{\sum_{e=0}^{e=5} \sum_{h \in A} C_{h,e}^A U} + \frac{\sum_{h \in U} \sum_{e=0}^{e=5} C_{h,e}^U \cdot \log \left( \frac{C_{h,e}^A + \alpha_e}{C_{h,e}^U + \alpha_T} \right) + \beta \cdot \sum_{h \in U} C_h^A \cdot C_h^U}{\sum_{h \in U} \sum_{e=0}^{e=5} C_{h,e}^U}.
\]

That can easily be decomposed into two parts:

\[
\text{Aug-log} - LV_A^{\text{norm}}(M^A, M^U) = \frac{\sum_{h \in A} \sum_{e=0}^{e=5} C_{h,e}^A \cdot \log \left( \frac{C_{h,e}^U + \alpha_e}{C_{h,e}^A + \alpha_T} \right) + \beta \cdot \sum_{h \in A} C_h^A \cdot C_h^U}{\sum_{h \in A} \sum_{e=0}^{e=5} C_{h,e}^A}.
\]

(5.36)
and

\[
\text{Aug-log} - \text{LV}_U^{\text{norm}}(M^A, M^U) = \frac{\sum_{h \in U} \sum_{e=0}^{5} C_{h,e}^{U} \log \left( \frac{C_{h,e}^{A} + \alpha_e}{C_{h,e}^{A}} \right) + \beta \cdot \sum_{h \in A} C_{h}^{A} \cdot C_{h}^{U}}{\sum_{h \in U} \sum_{e=0}^{5} C_{h,e}^{U}}.
\]

(5.38)

We will use (5.37) when finding the best transformation for a map to match another since the Accounted Map is the map we will be transforming to match the Underneath Map. For the winning transformation we will recompute the symmetric augmented log likelihood value (5.36).

### 5.7 Combining Two Maps

Combining two maps is very easy. Now that both maps are in the same coordinate system and with their hexagons aligned, the combination is performed by simply adding the counts of their edges, that is: \(M^C = M^A + M^U\). Figure 5.20 illustrates this.

![Illustration of the combination of two maps](image)

Figure 5.20: Illustration of the combination of two maps \((M^A \text{ and } M^U)\) when one of them has been transformed and are framed within the same coordinate system.

The resulting combined map inherits the Starting Conditions (represented on the plane by the red hexagon) of the Underneath Map.

### 5.8 Finding The Best Match Between Two Maps

The key idea under map merging is to find which transformation makes one map match another best. The problem here lies on finding a way to restrict the area where we look for that maximum, that is, boundaries for the rotation, scale factor and shift along the x and y axes that is applied to one of the maps, which obviously determine how much time and resources it takes to find the best match. We rely on concepts of simulated annealing whereby the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution [27].

An example that illustrates how finding the best match works would be printing the two maps on transparent paper and put one on top of the other. Then we could use our fingers to move (transform) one of them and our eyes to determine when they match best.
5.8. FINDING THE BEST MATCH BETWEEN TWO MAPS

5.8.1 Running the Loops

The search of the best transformation is done by transforming the Accounted Map and then by correlating it with the Underneath Map. Two key questions where solved:

- How to restrict the area of search.
- How to explore the area of search.

As a first approach, we will consider a situation in which the boundaries for the area of search can be typed in. This is the case we can apply in the Correlation Viewer (see Section A.4 in the Appendix A). In further sections we will explain how to use the Starting Conditions of the two given maps to restrict the area of search.

Regarding the exploration of the area of search, we just need to define some *deltas* for the x and y shifts, scale factor and rotation. Then the loops are run by taking every possible combination of rotation, scale factor and x and y shift values, transforming the Accounted Map and comparing it with the Underneath Map.

The number of transformations that is applied to the Accounted Map is:

$$\text{Number Of Transformations} = N_r \cdot N_s \cdot N_x \cdot N_y,$$

where

$$\begin{align*}
N_r &= \frac{U_r - L_r}{\Delta_r}, \\
N_s &= \frac{U_s - L_s}{\Delta_s}, \\
N_x &= \frac{U_x - L_x}{\Delta_x}, \\
N_y &= \frac{U_y - L_y}{\Delta_y},
\end{align*}$$

(5.39)

where $N$ represents the number of possible values, $U$ and $L$ the upper (higher) and the lower boundaries for the area of search and $\Delta$ the delta used for the search. $r$ means rotation, $s$ scale factor, $x$ x shift and $y$ y shift.

An improvement to this approach was made by realizing the symmetry of the grid of hexagons with respect to the x and y axes. The x and y shifts are divided into two parts, the integer part and the fractional part:

- The fractional part for the x shift always goes from 1.5 times the radius of the hexagons to 1.5 times the radius. The fractional part for the y shift always goes from $-\frac{\sqrt{3}}{2}$ times the radius to $\frac{\sqrt{3}}{2}$ times the radius.
- The integer part refers to the number of hexagons that the shift involves.

Figure 5.21 illustrates this concept with a map that is composed by single hexagon (in dark orange) that needs to be shifted with different values along the x axis.
In this example some possible x shift values that need to be explored are reduced to only one shift. It is easy to see that the projection of all those shifted hexagons will involve the same hexagons in terms of relative position with respect to the shifted hexagon. On the right we see the projection of the hexagon, that is, the resulting map after transforming the hexagon.

Once the map has been projected, the resulting map needs to be shifted again by the integer number of hexagons. The integer x shift is computed by multiplying the integer number (-1, 0, 1, 2, 3 or 4 in the picture) by 3 times the radius. This second shift is \textit{costless} because it is not a real shift in the sense that projection is not needed. What is done instead is \textit{slide} the Transformed Map on top of the Underneath Map. Figure 5.22 shows how this is achieved. In gray the Underneath Map has been depicted and stays fixed through the process. In green is shown, for a hexagon of the Underneath Map, the hexagon with which it will be compared in the Accounted Map. In those cases in which the green hexagon of the Underneath Map does not exist in the Accounted Map, an empty hexagon is used (plotted in light green), that is, with all transition counts 0.

Thus, in this example one single projection will be used for six comparisons.

The same reasoning can be applied to the y axis. Just note that the integer number must by now multiplied by $\sqrt{3}$ times the radius.

The improvement in the performance can be seen if the number of transformations needed for this new case is computed:

Number Of Transformations $= N_r \cdot N_s \cdot N_{frx} \cdot N_{fry},$

where

\[
N_{frx} = \frac{U_{frx} - L_{frx}}{\Delta_x},
\]
\[
N_{fry} = \frac{U_{fry} - L_{fry}}{\Delta_y},
\]

(5.40)
5.8. FINDING THE BEST MATCH BETWEEN TWO MAPS

Figure 5.22: Illustration of the sliding of the resulting map of Figure 5.21 to compare it to the Underneath Map in gray. In those cases in which, after sliding, a given hexagon of the Underneath Map (in green) does not exist in the slided Accounted Map, the hexagon in light green is used as an empty hexagon (all transition counts are 0).

and $U_{frx} = 1.5r$, $L_{frx} = -1.5r$, $U_{fry} = \frac{\sqrt{3}}{2}$ and $L_{fry} = -\frac{\sqrt{3}}{2}$ are the upper $U$ and lower $L$ boundaries for the fractional part of the $x$ and $y$ shift respectively.

So now the number of transformations for the $x$ and $y$ shift do not depend on the dimensions of the area of search anymore, reducing the cost of the comparison of two maps in great degree.
Storing the best augmented log likelihood value  Every time a transformation for the Accounted Map makes its combination with the Underneath Map have a higher asymmetric augmented log likelihood value (as stated by equation (5.37)) than the current one, the transformation and the value are stored.

5.8.2 Range of Search

This section aims to explain how the Starting Conditions of two maps can be used to establish the area of search for the transformation of one of the maps to match the other.

Let’s use the X starting condition for our explanation. Say the Underneath Map has a given mean Starting X or $\mu^U_x$ and a given standard deviation $\sigma^U_x$. Likewise, the Accounted Map has a Starting Condition for the X variable defined by the mean $\mu^A_x$ and a given standard deviation $\sigma^A_x$.

To compute a range of search we have to take the worst case scenario: when Underneath and Accounted Map are supposed to be separated by the largest possible distance to match each other, that is, they are located at opposite end of their respective Gaussian functions: $3 \cdot \sigma^A_x + 3 \cdot \sigma^U_x$, taken positive and negative. In figures 5.23(a) and 5.23(b) we have depicted the two extreme scenarios for the comparison.

![Graphs showing range of search](image)

(a) Lower boundary for the x shift for the comparison of two maps.

(b) Upper boundary for the x shift for the comparison of two maps.

Figure 5.23: Example of the boundaries for the range of the x shift.

So the range can easily be computed as:

$$U_w = 3 \cdot \sigma^A_w + 3 \cdot \sigma^U_w,$$
$$L_w = -(3 \cdot \sigma^A_w + 3 \cdot \sigma^U_w),$$

(5.41)

where $U$ and $L$ are the upper and lower boundary for the range and $w$ is any of the variables of the Starting Conditions: $x$, $y$, $a$ or $s$. 
We have chosen 3 times the standard deviation to define the extreme of each Gaussian distribution since the standard deviations from the mean account for about 99.7% of the set.

Nevertheless, for the scale factor variable, the range always explore values between 0.97 and 1.03, because of its multiplying effect.

**Why do we need to restrict the range?**

- **Save resources:** We want to avoid the comparison of two maps that do not overlap. We trust that the Starting Conditions are accurate enough to avoid exploring too large areas where the maps do not have a single hexagon in common.

- **Delimit the area automatically:** We need an automatic method to search for the best transformation without any human intervention.

**The manually written file** When a walk around some premises is performed, we can derive, within the coordinate frame provided by the hexagon matrix, where the walk started and with which orientation. It is a human work to write the Starting Conditions file, so that the FootSLAM algorithm can be launched. We will choose a greater standard deviation when we are less sure of where the walk exactly started, and smaller when we are more certain about it.

In the future a GPS anchor will be used to set the Starting Conditions for the walk when it starts outside the building, or even provide this GPS anchor in the middle or at the end of the walk if the pedestrian goes outside.

The Starting Conditions are especially important when using a prior map for FootSLAM. If the proper averages values for the variables of the Starting Conditions are not provided, the prior map might not even be used. If it is not given enough room for the particles to explore the area around them (greater standard deviation), FootSLAM might not be consistent with the prior map.

Consequently the Starting Conditions play a crucial role when running FootSLAM. As will be soon shown, the Starting Conditions also play a vital role when merging maps.

### 5.8.3 Iterating the Loops

Using some of the concepts of the Greedy Hill Climbing algorithm [23], once a maximum has been found by one single run of the loops, the area of search is reduced around the winning transformation that was found in that run of the loops and makes the deltas smaller. That is, a iteration over the loops is performed. The stopping criteria for this iterative search relies on the use of a correlation increment threshold (the difference between the maximum value found in this iteration and the maximum value found in the previous iteration). If this threshold is not reached after 3 iterations, the loops are stopped.

### 5.8.4 Behavior of the Augmented Log Likelihood Function

We want to present in Figure 5.8.4 the behavior of the augmented log likelihood function when the variables x and y shift, rotation and scale factor are varied one at a time, around
the best set of values (the values that transform one map to fit the other). The maps that are being considered are shown in Figure 5.20.

(a) Augmented log likelihood value as a function of the x shift. (b) Augmented log likelihood value as a function of the y shift.

(c) Augmented log likelihood value as a function of the rotation. (d) Augmented log likelihood value as a function of the scale factor.

Figure 5.24: Example of the behavior of the correlation function for two maps when we vary one of the transformation parameters at a time, and with $\alpha = 0.8$ and $\beta = 0.04$.

The two peaks that can seen at both sides of the absolute maximum for the y shift correspond to those values of y shift that make the upper horizontal corridor of one map match the lower horizontal corridor of the other map and vice versa.

5.9 The Prior Map

When FootSLAM is run with the raw odometry obtained by walking in a certain area for the first time, no prior knowledge about the hexagon transitions within the coordinate system defined by the hexagon matrix is available.

Nevertheless, when FootSLAM is run with the odometry from a second walk, the information provided by the previous walk can be used as a prior to help the new walk converge better. So the prior map can be seen as the transition counts of another map of the same premises.

But to address issues like, for example, the new walk discovering new areas of the building or too thick corridors, the prior map could need a weakening processing, so that it does not bias the path followed by the particles of the second walk.

To do so, a Prior Weakening Factor $\geq 1$ has been defined. This factor divides the transition counts of the prior map, making them smaller, that is, less strong.
Another possibility that can be used for the prior maps is to filter (distance filter) them. This is to account for the fact that, even if the new walk visits the same areas, spaces such as corridors might be wider than one hexagon diameter. Filtering the prior avoids the new walk to be too biased by it. Filtering the map with a distance filter makes the map "fatter".

5.10 Important Parameters for FootSLAM

5.10.1 Number of Particles for the Particle Filter

The particles of the particle filter for FootSLAM are used to explore the area and each one of them holds its own history of the transitions though the edges of the hexagons they visited. If there are too few particles then the space cannot be explored properly, that is, not all the possible hypothesis for the map are found and FootSLAM might not reach convergence. If there are too many particles, the memory consumption grows too much, making the algorithm too slow.

For our experiments in DLR and MIT premises we have chosen 90000 particles.

5.10.2 The Dimensions for the Coordinate System

The dimensions for the coordinate system are chosen before running FootSLAM. The number of hexagons in the x direction and in the y direction and their radius as well as the coordinates of the first hexagon must be provided. See Figure 5.2.

It is important to make sure that the coordinate frame is large enough for the resulting map from FootSLAM to fit into it. In this sense, for a given coordinate system, the Starting Conditions of the walk can also be modified so that the walk fits perfectly.

5.11 Comparison with the Ground Truth Floor Plan and Furniture

Up until now the goodness of a map could only be measured using one’s eyes. Now a more objective method to measure the quality of a map and see how well this map fits into the real plan of the building is proposed.

Because the Aggregated Maps are an aggregated version of all the integer maps that each of the particles of the filter obtained, the Best Map will be used to compare it with the plan. It is very easy to obtain a Total Best Map out of the individual Best Maps by addition of all the individual Best Maps. It is obvious that at the first iterations these are in need of further transformations, but as the iterations move forward, the best maps are placed where they are supposed to and the resulting best map is very accurate.

In this case the Best Map will not be transformed to fit the building plan. Instead the plan will be transformed to fit the Best Map, avoiding the lose of imprecision inherent to the transformation of FootSLAM maps.
5.11.1 Plotting the Plan and Furniture

Two .xml files [26] have been used to store the coordinates of the walls of the building and the coordinates of the pieces of furniture at the time of the walk.

The classes Wall Manager and Polygon Manager Double implemented by M. Khider have been used to build the structure of a building, and some new functions have been added to transform both walls and furniture. Some other functions have also been developed to allow drawing, on the Hexagon Drawing Panel, a Best Map along with the walls and the furniture arrangement.

The transformation for the walls and the furniture is performed by taking every 2D point and applying formula 5.1. The center for the rotation and the scaling is now one of the corners of the building.

5.11.2 Computation of the Ratio of Crossed Walls and Furniture

A ratio of crossed walls and furniture has been defined as follows:

\[
\text{ratio} = \frac{\text{transition count that cross a wall or a piece of furniture}}{\text{total transition counts in the map}}. \tag{5.42}
\]

Since walls are "lines" and furniture are "polygons" two different criteria can be differentiated to determine whether a transition count crosses a wall or a piece of furniture or not.

![Diagram](image)

Figure 5.25: Illustration of a transition counts that crosses a wall (on the left) and a piece of furniture (on the right)

A transition across a given edge from a source hexagon \( h_s \) to a target hexagon \( h_t \) represents somehow the probability of a pedestrian crossing it when walking from \( h_s \) to \( h_t \). The center of the hexagons can be used as starting and finishing points for the transition across the edge as an approximation.

**Criteria for the Walls:** A wall between two hexagon centers indicates a crossed wall. The left side of Figure 5.25 illustrates an example of a transition that crosses a wall.
5.11. Comparison with the Ground Truth Floor Plan and Furniture

Criteria for the Furniture: The center of the target hexagon $h_t$ lying inside a piece of furniture indicates a bad transition. The right side of Figure 5.25 illustrates an example of a transition count that would make the pedestrian step onto a piece of furniture.

Note that in the case we had a transition count crossing a wall and a piece of furniture at the same time, it would only be counted once.

Our approximation for crossing a wall or a furniture has proven to be valid for a radius compared to an average step (0.5 m in our case).

5.11.3 Searching Best Fit to Ground Truth

The idea now is very similar to that of looking for the best correlation between two maps when one of them is being transformed, except for now the plan of the building plays the role of the Accounted Map. The searching algorithm must look for the lowest ratio of crossed walls and furniture.

The loops are run around a point that is inserted manually (visually trying to obtain a good match). When the winning transformation is obtained, an iteration of the loops is performed around that winning point two more times in hopes to obtain a better crossed walls ratio. On the left side of Figure 5.11.3 an example of the plan and a Total Best Map of TE01 after 5 iterations is shown.

A problem was faced when computing this search: the ratio of crossed walls and furniture is 0 when the map and the plan are not on top of each other, or very small when they overlap just a little. To account for this effect, we decided to wrap the outer walls of the building with more walls, so that a solution like the ones mentioned where never found. This also has an important drawback: that we could potentially increase the ratio for those comparisons in which they do overlap. See the right side of Figure 5.11.3.

What we have resolved to do is: use the wrapped plan to find the minimum ratio and then recalculate the ratio for that transformation of the plan.
(a) The Best Map and the best fit with the ground truth and furniture arrangement.

(b) The Best Map and the best fit with the ground truth "wrapped" and furniture arrangement.

Figure 5.26: Example of the best fit between the building plan and furniture arrangement of building TE01 of DLR premises in Oberpfaffenhofen and a Best Map.
Chapter 6

"Turbo" FeetSLAM Algorithm

FeetSLAM is the process by which different maps obtained from multiple walks around the same area are combined to obtain a global map of the walkable areas.

This chapter is completely dedicated to introduce the reader to the algorithm that takes two or more data sets generated by walking in an area with a foot-mounted sensor, obtains an individual map out of each one of them using the FootSLAM algorithm and merges them. The FeetSLAM technique builds on iterative processing of odometry data, using maps originating from other data sets as a prior map for a given data set. With this iterative processing the total maps that are obtained are not dominated by one data set but are a result of balancing the characteristics of each data set with the effect of averaging out errors. Over iterations, the Starting Conditions and FootSLAM maps are gradually combined to obtain a high-accuracy global map. The name for the algorithm is "Turbo" FeetSLAM because it relies on concepts of Turbo Coding, as it was explained in chapter 4.

6.1 Goals of the "Turbo" FeetSLAM Algorithm

The "Turbo" FeetSLAM Algorithm has two main goals:

1. Obtaining a Total Aggregated Posterior Map, in short Total Map, of the walkable areas.

2. Obtaining more accurate Individual Maps.

Obtaining a Total Map is the first evident goal of a map merging algorithm. What is more of a surprising goal is the increase of accuracy in the Individual maps. We will show how the knowledge that the other data sets provide about the environment can be used to make one map more precise.

We have borrowed the heuristic approach from Turbo Coding [22] in which:

- The problem is decomposed into smaller segments.

- The data is processed in an iterative fashion: the maps can be pre-processed during iterations (weakening, filtering).

- Each stage feeds the other with "prior" information.
For FootSLAM, the prior can be shown to be the counts from all other data sets. This approach has also the advantage of being more suited to practical computation limitations such as limited memory.

### 6.2 The Algorithm

In Figure 6.1 the basic structure of the algorithm is presented:

![Algorithm Diagram](image)

Figure 6.1: Schematic illustration of how the algorithm for automatic map merging works.

The algorithm operates as follows: at the zeroth iteration FootSLAM is run for each one of the data sets with the help of some input parameters and the Starting Conditions of each walk. Once the Individual maps \( \{M_1, \ldots, M_n\} \) are obtained, with \( n \) being the number of data sets that is being considered at each iteration, they are properly combined to obtain the Total Map, \( M^C \). Then the Individual maps are transformed one by one to fit the Total map to generate the Individual Transformed maps \( \{M_1^T, \ldots, M_n^T\} \).

The Individual Transformed Maps are then used to generate the prior maps \( \{M_1^P, \ldots, M_n^P\} \) for the next iteration: for each data set, the combination of the other Individual Transformed maps is used to generate its prior map. The transformations \( \{T_1, \ldots, T_n\} \) found for the Individual maps to fit the Total Map along with the "winning" Starting Conditions \( \{SC_1, \ldots, SC_n\} \) at the end of each FootSLAM process are used for the next iteration to
obtain the new Starting Conditions for each data set. The prior maps are also used in the next iteration, properly weakened and filtered.

Our goal now is to explain the algorithm in greater detail, together with the rest of the parameters or input files involved. The index $i$ will refer to the iteration number.

### 6.3 Data

To run the "Turbo" FeetSLAM Algorithm with $N_W$ walks at iteration $i$ following files are needed:

1. Data Sets for the walks $\mathbf{D} = \{D_1, D_2, \cdots, D_{N_W}\}$. These data sets are just the result of converting the raw data from the walks into odometry. This odometry data sets do not change over the iterations.

2. Starting Conditions $\mathbf{SC}^i = \{SC^i_1, SC^i_2, \cdots, SC^i_{N_W}\}$.

3. Transformations for the Starting Conditions $\mathbf{T}^i = \{T^i_1, T^i_2, \cdots, T^i_{N_W}\}$.

4. Prior Maps $\mathbf{MP}^i = \{MP^i_1, MP^i_2, \cdots, MP^i_{N_W}\}$.

To start the "Turbo" FeetSLAM Algorithm at the zeroth iteration the manually written Starting Conditions need to be generated as explained in 5.1.4. As for the transformations, the following parameters are used:

\[
\begin{align*}
\Delta_x &= 0m \\
\Delta_y &= 0m \\
r &= 0rad \\
s &= 1.
\end{align*}
\]

As for the prior maps, a blank map is used, that is, a map with no hexagons.

### 6.4 FootSLAM Input Parameters Modification

This block is in charge of the modification of the input parameters to run FootSLAM for each iteration. The parameters that are modified from one iteration to the next are:

- **Prior Map Weakening Factor**: The Prior Map Weakening Factor is set up at the zeroth iteration at a certain value greater than 1 and is smoothly decreased to 1 over the iterations to make the prior map stronger. In our experiments 1.9 was empirically chosen.

- **Prior Map Filter Factor**: The Prior Map Filter Factor is smaller than 1 and it is smoothly increased up to 1 over the iterations. This is because the prior map will be more certain over the iterations, and can be given more importance during the FeetSLAM process. For our tests it was chosen to be 0.8.
- **Deltas for the search**: These are the deltas or increments used in the search of the best transformation for one FootSLAM map to match another. The deltas used for the 4 main variables are made smaller for each iteration, given that the range is made smaller as well, as we will show. This is achieved by defining a set of initial delta values and multiplying them by a *delta decay factor*, which is a value between 0 and 1.

### 6.5 FootSLAM Map Generation

FootSLAM is run sequentially for each one of the $N_W$ data sets. As a result $2 \cdot N_W$ maps ($N_W$ aggregated maps and $N_W$ best maps) are obtained.

FootSLAM is run *sequentially* because of the requirements of memory of such task, which do not allow, in an average computer, simultaneous FootSLAM processing.

### 6.6 Total Map Generation

This is the most important block of the algorithm. This is the stage of the algorithm in which the $N_W$ aggregated maps are processed to generate a global aggregated map.

#### 6.6.1 Comparing Maps Pairwise

Let’s think of a pool of maps that contains all the $N_W$ individual maps that were obtained with the FootSLAM technique. In $N_W - 1$ stages a Total Map that encompasses the information provided by all the maps can be generated as follows:

- At each stage, the maps are taken two at a time and compared. The comparison is performed by looping over the loops as explained in 5.8.3 for every possible combination of maps.

- At the end of each stage two of the maps are removed from the pool and its combined map is added. This means that after every stage there is one map less in the pool.

- The comparisons that were already run in previous iterations are not computed again.

The number of combinations that need to be tried is:
6.6. TOTAL MAP GENERATION

Number Of Combinations  =  \binom{N_W}{2} + \sum_{k=1}^{k=N_W-2} k

= \frac{N_W!}{2!(N_W-2)!} + \frac{(N_W-2)(1 + (N_W - 2))}{2}

= \frac{N_W(N_W-1)(N_W-2)!}{2(N_W-2)!} + \frac{(N_W-2)(N_W-1)}{2}

= \frac{N_W(N_W-1)}{2} + \frac{(N_W-2)(N_W-1)}{2}

= \frac{(N_W-1)}{2}(N_W + (N_W - 2))

= \frac{(N_W-1)}{2}(2N_W - 2)

= (N_W - 1)^2.

(6.1)

The first part of equation 6.1, \( \binom{N_W}{2} \), is the combinatorial number of the \( N_W \) individual maps taken 2 at a time, like the number of handshakes between \( N_W \) people. The second part of the equation, \( \sum_{k=1}^{k=N_W-2} k \), accounts for the possible combinations that still need to be computed every time a new map is added to the pool, that is, the comparisons between the new map and the other available maps in the pool. Note that there is no need to recompute the comparison between the maps that were already in the pool, since they were computed at the previous stage.

In figure 6.2 an example of how a total map for four individual maps is generated has been depicted. Stage 1 is conformed by a, b and c. Stage 2 by d, e and f and Stage 3 by g, h and i.

6.6.2 Choosing the Best Combination of All Available Combinations

Which pair is combined at the end of each stage? The augmented log likelihood value for each pair decides:

The best transformation for the Accounted map to fit the Underneath Map is found iterating over the loops as explained 5.8.3, that is, using the asymmetric augmented log likelihood function of equation (5.37). Once the best transformation is found, the symmetric augmented log likelihood value is computed only for that transformation and stored. Then another pair of maps is taken and the same procedure takes place. When all the possible combinations for the maps in the pool have been explored, the one that has the best (highest) symmetric augmented log likelihood value is chosen and the combined map is generated and stored.

When iterating over the loops, the asymmetric form of the augmented log likelihood function is used instead of the symmetric form to speed the process of finding the best transformation up. With this approximation we are assuming that the transformation
that was obtained using the asymmetric version will also make the symmetric version reach its maximum. This is not strictly true, but it has shown to work remarkably well in real scenarios.

6.6.3 Obtaining a Total Map

Obtaining a Total Map is then the result of merging $N_W$ maps, some of them transformed to fit each other.

The Total Map presents some drawbacks: as a result of combining maps that have been transformed, it is less precise than the "fresh from the oven" Individual maps that were obtained with FootSLAM. This problem is solved running another iteration that will use the information that was learned at the current iteration, and repeat this process again and again.

We have resolved to stop transforming the maps after 4 iterations, since the transformation is merely a projection (the values for the x and y shift and rotation tend to 0 m and 0 radians respectively and the scale factor tends to be 1). This way the spread of the counts that comes inherently with the projection process is avoided. The Total Map is then generated by simple addition of the Individual maps.

6.7 Transformation for Individual Maps

This block has an impact on:
6.8 Prior Map Generation

- Prior Map for the next iteration Generation.
- Starting Conditions for the next iteration.

The transformation for the individual maps is obtained by running the loops with the Total Aggregated Posterior Map as the Underneath Map and each one of the $N_W$ individual maps that were obtained with FootSLAM as the Accounted maps. The transformation that makes the transformed individual map and the total map have the best correlation value is the winner and is stored in a file along with the Individual Transformed Map.

We use the transformation between the individual map and the total map and not the transformation that was already computed to generate the Total Map for each one of the individual maps because this transformation will be used to transform the Starting Conditions in the next iteration for a FootSLAM process that will run on a prior map based on this same Total Map.

6.8 Prior Map Generation

The prior maps for the next iteration for each one of the $N_W$ data sets are very easily generated: for each Data Set, the other $(N_W - 1)$ Individual Transformed Maps obtained in Transformation for Individual Maps (Section 6.7) are combined. This is done so that when FootSLAM is run for a certain Data Set $d$ its own map is not explicitly included, but the information provided by the rest of data sets.

So the prior map can be seen as the transition counts of the other maps, properly combined.

6.9 Starting Conditions Generation

The Starting Conditions for the next iteration for each one of the $N_W$ data sets are very easily generated by taking the winning transformation computed in the Transformation of the Individual Maps block and applying it to the "winning" Starting Conditions for the FootSLAM process.

Furthermore, for every iteration we reduce the standard deviations by a certain factor, because we are more certain of the starting average conditions. This also makes the range for the search of the best transformation between maps smaller, and hence finding it takes less time over the iterations.

6.10 Zeroth Iteration Properties

Some characteristics of the zeroth iteration are:

- **No use of prior**: no previous knowledge of the transition counts is available.

- **Manually written Starting Conditions file**: as explained in Section 5.8.2, a human written file to state which the Starting Conditions for the walk were is needed.
• **Standard deviations 0 for FootSLAM but greater than 0 for the computation of the range of search:** The standard deviations are chosen to be 0 so that the resulting map from FootSLAM, given that it has no prior to be based on, does not incorporate a wide range of hypothesis for the map, that is, the particles have less freedom at the beginning of the exploration of the space state. But the standard deviations for the variables need to be greater than 0 so that a range can be computed. These standard deviations are read from the manually written Starting Conditions file.

• **Might not include all the \( N_W \) data sets:** data sets that do not converge without a prior might postpone its inclusion in the algorithm, in the expectation that it will converge when a prior map is available after processing the other data sets.

### 6.11 Optimization

The tools that have been used to make the algorithm faster are now presented.

#### 6.11.1 Profiling

A very good resource to check for unnecessary or repeated calls to methods is the Profiler that can be run in a Java Virtual Machine. This profiler proportionates a list of the methods that are being called and the percentage of time they need. With this tool we were able to reduce the duration of the running of the loops to twice its original time.

#### 6.11.2 Threading

A thread is a thread of execution in a program. It is the smallest unit of processing that an operating system can schedule. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently. The threads of a process share the process’ assigned memory, its code and its context \[25\].

We have implemented threading to speed up the \((N_W - 1)^2\) comparisons between maps that we have to perform in order to obtain a Total Map. To do so we have defined a pool of \(N_{Threads} = 4\) and an Executor Service that manages them.

### 6.12 Properties File

A Properties file is a text file with .properties extension. It is used in Java to store the configurable parameters of an applications. Each parameter is stored as a pair of strings, one storing the name of the parameter (called the key), and the other storing the value. The following have been chosen to be configurable parameters:

• The name for the folder where we will store all the files: a path with the folder where all the .txt files or any other information will be stored.

• Starting and finishing iteration number: at which iteration the algorithm must start and finish. This is very useful when the algorithm needs to be stopped and later restarted.
• Number of data sets. Number of walks that took place to collect data.

• Iteration where each data set is included: for those cases in which data sets do not converge without a prior and want to be included later on.

• Start the algorithm at the map generation stage (FootSLAM) or at the total map generation stage (When the FootSLAM maps are already available)?: a boolean to choose whether the algorithm needs to start from the running of FootSLAM stage. When the FootSLAM maps are already available, time can be saved jumping to the total map generation stage.

• Visualize the maps?: a boolean to activate or deactivate the visualization of the FootSLAM process.

• Store images of the FootSLAM process?: a boolean to choose if the FootSLAM images want to be saved.

• Number Of Particles: the number of particles for the particle filter.

• Coordinate System (Hexagon Matrix): the dimensions of the matrix and its base.

• Radius of the Hexagons.

• Initial Prior Weakening Factor: initial value for the weakening factor for the prior maps, that will be decreased over the iterations.

• Initial Prior Filter Factor: initial value for the filter factor for the prior maps, that will be increased over the iterations.

• Hexagon Correlation Factor: the value for $\beta$ to be used in equation (5.36).

• Decay factors for the range: a value smaller than 1 to multiply the range, so that it is smaller for every iteration.

• Decay factors for the deltas: a value smaller than 1 that multiplies the deltas used when exploring the range, so that steps to explore the range are always smaller over the iterations.

• Number of iterations when looping the loops: the number of iterations after which the iteration over the loops is stopped if the threshold increment for the correlation is not reached.

• Path for each data set: the path where the odometry data sets are located in the hard disk of the computer.

• Assignation of the data sets to a name (D1,D2, D3, etc.): each odometry set is assigned a name.

• Angular Transition Per Meter Travelled: an internal parameter of FootSLAM that is rooted in the locality assumption of human motion in constraint environments.
6.13 Logger

A .log file has been used to store every single step of the "Turbo" FeetSLAM Algorithm. The Starting Conditions for a FootSLAM process at any iteration for any data set, or the best likelihood value between two maps, even if they were not chosen to be combined, etc. can be checked. This was one of the most important resources when debugging the behavior of the "Turbo" FeetSLAM Algorithm.

6.14 Naming System

The reader might be already aware of the complexity of the algorithm, so a naming system for the all the files was defined, following EBNF (Extended Backus Normal Form), which provides a formal way to express a context-free grammar [24].

A brief introduction to the grammar is now presented:

- The equal sign = indicates definition.
- The quotation marks " . . " indicate a terminal string.
- The vertical line or pipe " | " represents the possibility to choose between the symbols that it separates.
- The curly brackets { . . } indicate that the symbol is optional and can be repeated.
- The brackets ( . . ) indicate grouping.
- The comma , indicates concatenation.
- The semicolon ; indicates termination.

Here we present the symbols of our grammar:

digit excluding zero = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
zero = "0";
digit = zero | digit excluding zero;
natural number = digit excluding zero, {digit};
iteration = "_it", (zero | natural number);
data set = "D", {natural number};
extension = ".txt";
post = "mapPosterior";
sc = "StartingConditions";
tr = "Transformation";
prior = "priorMapFor";
best = "BestMap";
corr = "Correlation";
tot = "Total";

combination = (data set, "{", data set);
6.15. COMPLEXITY ANALYSIS

multiple combination = combination | ( "(" , combination , ")" , "}" , data set )
 | ( data set , "{" , "(" , combination , "")" )

individual aggregated map = post , data set , iteration , extension;
best map = best , data set , iteration , extension;
total map posterior = post , tot , iteration , extension;
combined map posterior = post , multiple combination , iteration , extension;
starting conditions = sc , data set , iteration , extension;
prior map = prior , data set , iteration , extension;
correlation = corr , multiple combination , iteration , extension;
transformation = tr , multiple combination , iteration , extension;
individual map transformation = tr , data set , iteration , extension;

Some terminal string examples for our naming system are:

individual aggregated map : mapPosteriorD3_it2.txt
best map : BestMapD1_it0.txt
total map posterior : mapPosteriorTotal_it9.txt
combined map posterior : mapPosterior(D3{D4){D1_it1.txt
starting conditions : StartingConditionsD5_it0.txt
prior map : priorMapForD4_it3.txt
correlation : CorrelationD1{(D2{D4){D5)_it6.txt
transformation : Transformation((D4{D3){D1){D2_it0.txt
individual map transformation : TransformationD1_it1.txt

6.15 Complexity Analysis

To analyze an algorithm is to determine the amount of resources necessary for its execution. Most algorithms are designed to work with inputs of arbitrary length. Usually the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or storage locations (space complexity) [30].

In theoretical analysis of algorithms it is common to estimate their complexity in the asymptotic sense, i.e., to estimate the complexity function for arbitrarily large input. Big O notation, omega notation and theta notation are used to this purpose [29].

6.15.1 Big O Notation

Big O notation is a convenient way to express the worst-case scenario for a given algorithm [30]: informally, an algorithm can be said to exhibit a growth rate on the order of a mathematical function if beyond a certain input size \( n \), the function \( f(n) \) times a positive constant provides an upper bound or limit for the run-time of that algorithm. In other words, for a given input size \( n \) greater than some \( n_0 \) and a constant \( C \), the running time of that algorithm will never be larger than \( C \cdot f(n) \).
6.15.2 "Turbo" FeetSLAM Performance

For our case of study, let's analyze the following pseudo-code for the "Turbo" FeetSLAM Algorithm, which we are reproducing again here:

```
FOR iteration FROM 0 TO #iterations-1
    FOR data set FROM 1 TO N_W
        map(data set, iteration) = FootSLAM(prior(data set, iteration-1), data set)
    END FOR
    total map(iteration) = combination(maps (iteration))
    FOR data set FROM 1 TO N_W
        prior(data set, iteration) =
            obtain prior(maps(iteration) \ map(data set, iteration), total map)
    END FOR
END FOR
```

The order of growth with the number of iterations ($N_{it}$) is linear:

$$Time(N_{it}) = O(N_{it})$$

But when we want to analyze the impact of the number of data sets on the time it takes to finish one iteration, we need to study the "combination" and "obtain prior" calls, which need $(N_W - 1)^2$ comparisons (equation (6.1)) and $N_W$ comparisons (the $N_W$ individual maps are transformed to fit the total map) respectively, that is, $(N_W - 1)^2 + N_W$ comparisons per iteration. For a sufficiently large $N_W$, we can state that:

$$Time_{comparisons}(N_W) = O(N_W^2)$$

Furthermore, we need to run $N_W$ FootSLAM algorithms per iteration, that is,

$$Time_{FootSLAM}(N_W) = O(N_W)$$

so

$$Time(N_W) = Time_{FootSLAM}(N_W) + Time_{comparisons}(N_W)$$

We can finally state that for a $N_W$ sufficiently large,

$$Time(N_W) = O(N_W^2)$$  \hfill (6.2)

The reader must note that for this derivation, the use of threads for the comparison of the maps has not been considered.

**Use of Trees** Since the impact of the number of data sets $N_W$ has the greatest impact on the performance of the "Turbo" FeetSLAM Algorithm, the following procedure can be use when the number of data sets is too high:

1. Take the $N_W$ data sets in groups of $N_D$ data sets.

2. Apply the "Turbo" FeetSLAM Algorithm to these $N_G = ceil(N_W / N_D)$ groups.
3. Make $N_W = N_G$ and then repeat steps 1. and 2. until only one last total map is left.

In Figure 6.3, an example for $N_W = 17$ walks and $N_D = 3$ has been represented. The height of the tree (the number of "floors" for the "Turbo" FeetSLAM Algorithm) is $h = \text{ceil}(\log_{N_D} N_W)$.

![Tree structure](image)

Figure 6.3: Tree structure for a map merging case in which the number of data sets is too high. $N_W = 17$, $N_D = 3$.

The benefits of proceeding like that are that the "Turbo" FeetSLAM algorithms have a much lower complexity $O(N_D^2)$ and can be run in parallel in different machines.

The drawback is that instead of running one single time the "Turbo" FeetSLAM Algorithm, it needs to be run $\sum_{j=1}^{j=h} N_W^{1-N_D} = N_W^{1-N_D} - \frac{1}{N_D - 1} = N_W \frac{1 - \frac{1}{N_D}}{N_D - 1}$ times.

A deeper study of the complexity of this new approach should tell us the minimum number of data sets $N_W$ for which it is specially convenient, and the number of data sets $N_D$ to put in each group.
CHAPTER 6. "TURBO" FEETSLAM ALGORITHM
Chapter 7

Experimental Verification

We have tested our map merging algorithm in two different indoor locations: the upper floor of the DLR building TE01 in Oberpfaffenhofen, Bayern (Germany) and the MIT premises in Boston, Massachusetts (USA). All walks were undertaken by Patrick Robertson, who also converted the raw odometry into data sets that can be used by FootSLAM to obtain a map.

We want to present the results in the following sections by showing the resulting maps over the iterations.

7.1 Data Collected at DLR Premises

Five different data sets from five different walks that took place in 2009 and 2011 have been used. All the walks have the main corridors in common, but they differ in the extension of the offices visited.

The zeroth iteration is run with no prior and the Starting Conditions are read from a manually written file that states how accurate they are. The resulting maps are shown in Figure 7.1 and labeled as $M^0_d$ with $d = \{1, 2, 3, 4, 5\}$ symbolizing each one of the five data sets. As we can see, the data set number 5 did not converge without a prior. We will see that using the information obtained in this iteration will make it converge in the next one.

For this iteration, the resulting winning combination to generate the total map was $(((D2\{D3\}{D1\{D4\}}{D5}$. The combined map is also shown in Figure 7.1, under the name $M^0_C$.

For the first iteration, using the priors built in the previous iteration, the maps are shown in Figure 7.1.

And the total maps for this second iteration and the next seven iterations are shown in Figure 7.1.

7.1.1 Starting Conditions and Transformations During Iterative Processing

The evolution of the average values of the Starting Conditions over the iterations for data set number 4 are now shown, as well as the evolution of the transformation applied to
them as an illustration of how after some iterations, the process remains quite static. See Figures in 7.1.1 and 7.1.1 respectively.

7.1.2 Hexagon Transition Error Rate During Iterative Processing

Using our Find Best Fit for Plan (section 5.11) algorithm, the curves in figure 7.1.2 have been obtained.

As can be seen, running one more iteration does not necessarily mean that the ratio of crossed walls decreases. The curves show the existence of a threshold for the ratio of crossed walls and furniture that is quickly reached after running two or three iterations. Nevertheless, an improvement of almost 9 times less crossed walls and furniture for the total MAP map and 4 times less crossed walls and furniture for the individual map of data set number 4 is achieved after 10 iterations.

The order for the ratio of crossed walls and furniture is $10^{-2}$.

7.1.3 Comparison With the Ground Truth

Figure 7.7 shows the comparison with the ground truth after ten iterations.

The only mismatch is due to the best map of data set number 1, which is shown in Figure 7.8. The right side shows a corridor that does not exist. The Aggregated Map does not show this corridor but the winning particle went through it. This effect did not show in the previous iterations, and hence shows how FootSLAM itself is still to be improved.

This is the best representation of the power of the FeetSLAM algorithm. A map that fits almost perfectly the building plan and furniture arrangement has been obtained. This map includes information about all the rooms and corridors that were visited by five pedestrians.

7.1.4 Discovery: The wall that was misplaced

After running the FootSLAM for some of the data sets and comparing them with the true ground floor we saw that we crossed unceasingly a certain wall. We thought that could be due to the accuracy of FootSLAM. But after merging the maps we were able to confirm that the pedestrian's had been actually able to go through that wall, because that wall was not really there.

This is a great example of how plans are often outdated and how FootSLAM will help maintain maps up to date.

7.2 Data Collected at MIT premises

In this case only four walks were taken into consideration, but they overlap in less degree. These walks took place in March 2011 and were much longer walks than the ones at DLR. All of them have in common a central part. Again the zeroth iteration is run with no prior maps and with a manually written file that contains the Starting Conditions for each one of the walks.

The maps after iteration 0 are shown in Figure 7.2.
After transforming them, the total map is the one shown in ??, and it is the result of this winning combination: ((D3D2)D4)D1.

Nevertheless, after running FootSLAM using the information provided by the previous iteration, the individual look like shown in picture 7.2.

And the total maps over the iterations are shown in 7.2:

As we can see, the resulting maps are much more accurate, reflect less hypothesis and the total map show all the "walkable" areas that the four pedestrians were able to visit.

7.2.1 Starting Conditions and Transformations During Iterative Processing

We now show in Figures 7.2.1 and 7.2.1, respectively, the evolution of the average Starting Conditions and the transformations applied to them over the iterations for data set number 4. These figures illustrate again the static profile the FeetSLAM process exhibits after running two or three iterations.

7.2.2 Comparison with the Ground Truth

Because we do not possess the original plan drawn by the architect who design the MIT building, but only a design, the goodness of the maps over the iterations cannot be computed. Nevertheless, a design of the building was available to us and it is shown in Figure 7.14 with the Best Total Map after 9 iterations for the data collected at MIT.

7.3 Used Resources (CPU, Memory)

The map merging algorithm is very expensive both in terms of CPU and memory.

We have used the Intel® Xeon® Processor X5690 [28]. This processor has six dual cores and a clock speed of 3.46 GHz. It takes 37 and 42 hours to run, respectively, ten iterations for the 5 DLR data sets and the 4 MIT data sets with 90000 particles.

CPU use has been optimized during the generation of the total map, allowing the comparison of 4 pairs of maps at the same time. We have not introduced any parallelizing processing during the FootSLAM process itself because running FootSLAM on one data set with 90000 particles requires 10GB of heap memory.

7.4 Observations

- When generating the total map with the pool of individual maps, the winning combination for each stage involves the combined map that was generated in the previous stage. This is because the combined map has more transition counts.

- Using the priors provided by other data sets when running FootSLAM makes the map more accurate and eliminates less strong hypothesis.

- Except for iteration 0, we always obtain the same winning combination over the iterations for the maps involved. We could use this information to avoid performing
$(N_W - 1)^2$ comparisons from iteration 2 or 3, but only the ones that we already know that won, reducing the order of growth of the "Turbo" FeetSLAM Algorithm.

- FootSLAM gives preference to those hypothesis that have a smaller scale factor, so to avoid obtaining smaller and smaller maps over the iterations, FootSLAM must be run with $\sigma_s=0$ for all iterations.

- From iteration 4 the transformation for the individual maps are almost null. This is why from iteration 4 we just add the maps when generating the prior maps and do not transform the Starting Conditions for the next iteration. This also saves time, since we do not need to compare the individual maps to the total map.

- We need to use a very many particles to ensure the FeetSLAM algorithm works properly. The number of particles can only be chosen empirically.
Figure 7.1: (a) to (e) Individual maps obtained running FootSLAM with no prior (zeroth iteration) for the "DLR data". (f) Total combined map.
Figure 7.2: (a) to (e) Individual maps obtained running FootSLAM using the priors obtained in the previous iteration for the "DLR data". (f) Total combined map.
Figure 7.3: Total combined maps at the end of iterations 2 to 9 for the "DLR data".
(a) The average $x$ coordinate over the iterations.

(b) The average $y$ coordinate over the iterations.

(c) The average scale factor over the iterations.

(d) The average heading of the pedestrian over the iterations.

Figure 7.4: The Starting Conditions for data set number 4 of "DLR data" over 10 iterations.
7.4. OBSERVATIONS

(a) The x shift over the iterations.

(b) The y shift over the iterations.

(c) The scale factor over the iterations.

(d) The rotation over the iterations.

Figure 7.5: Transformations applied to the average Starting Conditions (mean x, y, scale factor and rotation) over 10 iterations for data set number 4 of "DLR data".
(a) The ratio of crossed walls and furniture for the DLR total MAP over the iterations.

(b) The ratio of crossed walls and furniture for the DLR MAP of data set number 4.

Figure 7.6: Ratio of violated walls and furniture by the total DLR MAP map and the MAP for data set number 4 over the iterations.
Figure 7.7: Best fitting between the total MAP map after 10 iterations and the ground truth and furniture arrangement at the time of the walks.

Figure 7.8: Aggregated (on the left) and Best Map (on the right) of data set number 1 of the "DLR data" at iteration 9. The Best Map shows a double corridor on the top right side. This is not an error, but a result of the stochastic process that FootSLAM is.
Figure 7.9: (a) to (d) Individual maps obtained running FootSLAM with no prior (zeroth iteration) for the "MIT data". (e) Total combined map.
Figure 7.10: (a) to (d) Individual maps obtained running FootSLAM with the prior maps obtained at the zeroth iteration for the "MIT data". (e) Total combined map.
Figure 7.11: Total combined maps at the end of iterations 2 to 9 for the "MIT data".
7.4. OBSERVATIONS

(a) The average x coordinate over the iterations.

(b) The average y coordinate over the iterations.

(c) The average scale factor over the iterations.

(d) The average heading of the pedestrian over the iterations.

Figure 7.12: The Starting Conditions for data set number 4 of "MIT data" over 10 iterations.
CHAPTER 7. EXPERIMENTAL VERIFICATION

(a) The $x$ shift over the iterations.

(b) The $y$ shift over the iterations.

(c) The scale factor over the iterations.

(d) The rotation over the iterations.

Figure 7.13: Transformations applied to the average Starting Conditions ($\text{mean } x, y, \text{scale factor and rotation}$) over 10 iterations for data set number 4 of "MIT data".
Figure 7.14: Total MIT MAP map at iteration 9 on top of the original plan of the MIT premises where the walks took place.

Figure 7.15: A zoom of Figure 7.14 with the area that was visited by the four walks.
Chapter 8

Achievements, Conclusions and Further Work

8.1 Achievements

We have created a fully automated FeetSLAM implementation. The algorithm only needs the recorded data from two or more walks as an input along with the Starting Conditions and a coordinate system for the walks. We have proved the algorithm to work perfectly in two different scenarios and with different degrees of overlapping area, obtaining more precise and accurate individual maps and a complete map of the walkable areas.

Figure 8.1 shows the main tasks that have been undertaken for this thesis. The letter P between brackets means that Patrick Robertson, the supervisor of this thesis at DLR, was the one who was in charge of the task, instead of the author of this thesis, María Jesús García Puyol.

Figure 8.1: Main tasks undertaken for this thesis. (P) indicates that the task was developed by Patrick Robertson.
8.2 Conclusions

FootSLAM is a new technique that addresses the challenges of mapping and localization in a simultaneous fashion in environments where GNSS signals do not reach or are strongly distorted, i.e. indoor and underground locations. FootSLAM maps can be later used by pedestrians to localize themselves within these environments, making the localization problem exhibit less error growth.

FootSLAM is the beginning of Collaborative FootSLAM, whereby multiple pedestrians collaborate to generate an extensive map of an indoor environment, allowing the generation of continuously updated maps. We have developed a fully automated algorithm that takes a number of odometry data sets and process them in an iterative fashion in order to obtain a global map of the walkable areas and more accurate individual maps. We have borrowed an heuristic approach from a family of error correction coding schemes from digital communications theory to develop a suboptimal but more adequate to memory restrictions solution for our FootSLAM problem.

When doing so, we have faced scale, rotation and translation ambiguities that are often inherent in SLAM. We have solved them transforming the maps on an edge by edge basis and transforming the Starting Conditions for the FootSLAM process. Using our iterative approach, the ambiguities are averaged out and the maps are not in need of further transformations. GPS anchors will help improve the performance of the "Turbo" FootSLAM, since the Starting Conditions will be more accurate from the beginning.

We have also developed an objective measure of the goodness of FootSLAM maps that counts how many transition counts cross a wall or a piece of furniture. Our results show, after 10 iterations, an improvement of 4 times for individual maps and an improvement of 9 times for the total map. We are facing a ratio of crossed walls and furniture of the order of $10^{-2}$ after 10 iterations.

We have shown the power of FootSLAM to generate a complete map of the walkable areas of two scenarios. The results are remarkably good for both scenarios:

- We have proven the ability of FootSLAM to increase the convergence and accuracy of a single FootSLAM map, more visible when it does not converge without a prior map.

- We have shown the ability of FootSLAM to help notice misplacements of walls or furniture with respect to the original building plan, helping generate an updated map of the area.

We believe different patterns of walks (i.e. with no loop closure), should also work in this collaborative scenario, even if none of the data sets converges on its own. The iterative processing of the FootSLAM algorithm will help the walks converge and finally generate a total map.

8.3 Further Work

Further work should address:
8.3. FURTHER WORK

- On-line map merging and real-time processing: we expect the advanced in technology make the memory constraints less so and be able to parallelize the running of FootSLAM for the different data sets.

- GPS anchor: so far we have used a manually written file to read the Starting Conditions for each walk. We will substitute this manually written file with the coordinates provided by a GPS anchor, to avoid any kind human intervention.

- The mathematical basis for the correlation function: The derivation of the correlation function is still challenging and will need to be improved.

- Computational requirements, time consumption.

- A better algorithm to find the best fit for the plan and the map.

- 3D: FootSLAM will soon be extended to 3D. 3D map merging should also be implemented.

- More realistic walk patterns, in which the pedestrian does not close loops or where the walks are much shorter and have less overlapping areas with the other walks.

- A stopping criteria for the "Turbo" FeetSLAM Algorithm: a criteria based on how the maps (individual and total maps) change from one iteration to the next.

- FeetSLAM Maintenance: so far the work has focused on FeetSLAM bootstrap, that is, the generation of a global map with no previous information. The next step should be the introduction of a hierarchical priorities system that allows the introduction of new data sets when new walks take place, to help updating the maps.
Bibliography


Appendix A

Visualization

This appendix presents the main Graphical User Interfaces (GUI) that have been developed for this thesis work.

A.1 Hexagon Map Drawing Panel Extension

In this section we want to introduce the extensions to the Hexagon Map Drawing Panel presented in 5.1.6.

A.1.1 Hexagon Maps

The Hexagon Map Drawing Panel (Section 5.1.6) is a panel where Best or Aggregated maps can be plotted. The first extension to its performance is to plot transition maps for individual hexagons, that is, to show the transition counts for each edge of a hexagon and the total number of transition counts for that hexagon. This has been implemented so that when the user clicks on a hexagon, a hexagon map panel is opened. The Java® that allows this is called "Hexagon Panel".

An example of this can be seen in Figure A.1, where a hexagon of a "vertical" corridor has been represented. The lower the number of counts for an edge is, the darker it is plotted, symbolizing that that transition is less probable. As can be seen, edges 0 and 3 (see Figure 5.1) hold almost all the transition counts of the hexagon, whereas edges 1, 2, 4 and 5 have none or almost none, meaning that there are probably walls constraining the motion of the pedestrian. On the top, the coordinates of the center of the hexagon are drawn, so that it is easy to localize the hexagon within the map.

This application represents the best tool to understand how FootSLAM maps are built and how transformation and projection affect the counts of the edges.

A.1.2 Map Viewer

The Map Viewer application allows to explore the file system and choose any bestMap*.txt or mapPosterior*.txt file to plot on a Hexagon Map Drawing Panel. The advantage is to avoid typing in the whole path for the map that wants to be plotted.

As a result of a bug in javax.swing.JFileChooser used for this application, the "File name", "File type" and other text fields are in Spanish. This is because JFileChooser
Figure A.1: Transition map for a given hexagon: the total transition counts for the hexagon (in the center) and the transition counts for each edge.

Figure A.2: The Map Viewer application that allows the user to explore the file system and choose the map that wants to plot.

uses the language of the operating system where Java is running, which in this case was Spanish.

A.1.3 Graphical Representation of Subsets of FeetSLAM Maps

The tool called "Plot Results from FeetSLAM" allows the user to plot some of the maps (Aggregated, Best and Total Map) for some of the data sets over some of the iterations. This is a very useful application to see the bigger picture of the performance of FootSLAM. One can click on any of the maps shown to open it in a new window (a Hexagon Map Drawing Panel window). An example is shown in Figure A.3.
Figure A.3: The panel for the "Paint the Results" application with the resulting Aggregated and Best maps obtained with the "DLR data" at iteration 1.

A.1.4 Saving the Maps as Images

The "Saving the Maps" application is just a method that saves all the maps in a folder into *.png images with the same name. It is a very useful tool to make the animation of the "Turbo" FeetSLAM Algorithm.

A.2 Transformation Viewer

Transformation Viewer is a GUI used to draw a FootSLAM Aggregated Map or a building plan and manipulate them. This viewer allows the transformation and projection or filtering of a given map and the immediate visualization of the resulting map or the transformation of the ground truth and furniture arrangement of a building, depending on the application the user needs to run.

A.2.1 Transformation Viewer for a FootSLAM Map

Figure A.4 shows the disposition of the buttons and fields for the transformation of a FootSLAM map. This is the case when an Aggregated Map needs to be transformed to fit another map.

As we can see in the figure, one can type in the parameters of the transformation and click on the "Transform" button or type in the increment values and click on the + and - buttons. One can type in the distance filter factor and click on "Filter" or type in the angular filter factor and click on "Angular filter". Furthermore, one can
reset the transformation or the filter that was applied to the map by clicking on "Reset Transformation" or "Reset filter" buttons respectively.

It is also possible to store the map at any time by clicking on the "Save into File" button. The original file is never overwritten. And the type of angular projection can be also chosen: when "Check Angular Projection" is checked then the projection onto six edges is used. Otherwise, the projection onto two edges is used.

As an additional feature, one can type the strength for the projection factors in the two last fields at the bottom of the window.

The advantage of this tool is that the effect of a transformation or a filter factor on a given map can be immediately seen.

A.2.2 Transformation Viewer for the Ground Truth and Furniture

This is the case in which the building plan and its furniture are transformed to fit a Best Map. In Figure A.5 the application to transform the plan to make it fit the Best Map is shown.

In this case, only the transformation buttons are necessary.

A.3 Violated Walls and Furniture Viewer

The Hexagon Map Drawing Panel has been extended to draw walls in the form of lines and pieces of furniture as polygons. In Figure A.6, the application that allows the transformation of the building plan and the furniture to fit a Best Map is shown. It is called "Find Best Fit for Floor Plan".
A.4. Correlation Viewer

The Correlation Viewer is the next logical step after the Transformation Viewer that was presented in A.2. This application allows the user to draw two maps, transform them using a Transformation Viewer for each one and see the numerical value of the asymmetric normalized augmented log likelihood function for the Accounted Map (equation 5.37) using a Hexagon Correlation Factor $\beta$ that the user can type in. The correlation viewer window is shown in Figure A.7.

The user can click on the "Get correlation" button to generate a Correlation Map that shows with colors how good the correlation for a given hexagon is: yellow represents the threshold value $\log\left(\frac{1}{6}\right)$ for when two maps do not overlap, green represents a better
Figure A.7: Window with different buttons to obtain the correlation between two maps, insert the hexagon correlation factor, run the loops or obtain the combined map.

correlation between the maps and red, a worse correlation. In Figure A.8 we have shown an example for two of the maps of the "DLR data".

Figure A.8: An example of the correlation panel that shows, on a hexagon level, how well two maps fit. Green represents better fit than the threshold value for when the hexagons do not overlap (yellow). Red represents worse correlation.

Furthermore, clicking on any hexagon of the correlation map opens a hexagon correlation panel where counts for both maps and the correlation per edge and hexagon are shown, as illustrated in Figure A.9. For each edge, the transition counts for the Accounted Map, the transition counts for the Underneath Map and the value of their asymmetric normalized augmented log likelihood value (after the arrow "->") are shown. The asymmetric normalized augmented log likelihood value for the hexagon is shown in red in the center. The coordinates of the center of the hexagon are drawn on the top of the panel. The hexagon correlation panel has been implemented in the "Hexagon Panel" class.

The correlation viewer also allows the user to combine two maps and plot the result by clicking on the "Get combined map" button of the correlation viewer window in Figure
A.5. Chart Generation

The charts that show the evolution of the ratio of crossed walls and furniture over the iterations or the augmented log likelihood function behavior are drawn with a Java class that relies on the use of JFreeChart [31].
A.6 Production of Video Sequences of FootSLAM

Adobe® Premiere® Pro [32] is a very useful tool with which the results of the "Turbo" FootSLAM Algorithm can be shown in the form of an animation.

In order to make a FeetSLAM animation, the individual videos for each FootSLAM process need to be produced. Premiere allows the combination of different videos within one video. One can choose the position and orientation of the individual videos and add images, titles and transitions.

The FootSLAM algorithm allows the storage of images in a *.png format during the FootSLAM process itself. Then each FootSLAM video can be easily produced with these images using the following code lines in a Linux OS running machine:

```
ls -t -r *.png > lst.txt
mencoder mf://@lst.txt -mf w=1024:h=944:fps=25:type=png -ovc lavc
-lavcopts vcodec=mjpeg: -oac copy -o video.avi
```

The first line writes into lst.txt the names of the .png images by date of modification, first the oldest ones and then the newest. The second line takes all the pictures listed in lst.txt and generates an .avi video with mjpeg codification. The width and height of the video are 1024 and 944 respectively and the speed is 25 frames per second (fps).
Appendix B

Code

This appendix intends to provide the reader with a bigger picture of the main Java® classes that have been written and the relationships between them.

B.1 Concepts

A brief introduction to a few concepts of Java® programming is now presented [33]:

**Class:** a class is a type that defines the implementation of a particular kind of object. A class definition defines instance and class variables and methods, as well as specifying the interfaces the class implements and the immediate superclass of the class.

**Interface:** an interface is a reference type, similar to a class, that can contain only constants, method signatures, and nested types, but no method bodies. Interfaces cannot be instantiated: they can only be implemented by classes or extended by other interfaces.

**Implementation of an interface by a class:** a class that implements an interface must implement all the methods declared in the interface. The word *implements* is used to this end.

**Inheritance:** in an object-oriented programming context, the name inheritance refers to the ability of an object to inherit characteristics from another object, that is, an object is able to pass on its attributes and behaviors to its children. This is achieved using the word *extends*.

**Association:** the association relationship is a way of describing that a class holds a reference to another class. In association diagrams, the indicators of Table B.1 are used to show the multiplicity of the association.

B.2 Package and Class Structure

The Java® workspace used for this thesis is divided into the following folders:

- bayesian
APPENDIX B. CODE

<table>
<thead>
<tr>
<th></th>
<th>Multiplicity Indicators for Association Relationships Between Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>Zero or one</td>
</tr>
<tr>
<td>1</td>
<td>One only</td>
</tr>
<tr>
<td>0..*</td>
<td>Zero or more</td>
</tr>
<tr>
<td>1..*</td>
<td>One or more</td>
</tr>
<tr>
<td>n</td>
<td>Only n (where n &gt; 1)</td>
</tr>
<tr>
<td>0..n</td>
<td>Zero to n (where n &gt; 1)</td>
</tr>
<tr>
<td>1..n</td>
<td>One to n (where n &gt; 1)</td>
</tr>
</tbody>
</table>

Table B.1: Multiplicity indicators for association relationships between objects.

- ChartApp
- GPSConnection
- Logging
- mNAVv2
- moscito
- rtree
- xsensMT

All the new classes or interfaces were stored under the bayesian folder, in the dlr.solo.particle.-core.movement.GridFootSLAM package. They are:

- Convergence Frequency
- Coordinate System Interface
- Correlation and Map Names Container
- Correlation Interface
- Correlation Result
- Correlation Viewer
- Find Best Fit for Floor Plan
- Find Best Transformation
- Find Best Transformation Viewer
- FootSLAM Aggregated Map Correlator
- FootSLAM Map Info
- FootSLAM Map Interface
- Hexagon Panel
B.2. PACKAGE AND CLASS STRUCTURE

- Histogram
- Log Likelihood Function
- Log Likelihood Function Curve
- Loop Deltas
- Loop Loops
- Loop Parameters
- Loops Viewer
- Map Viewer
- Not Valid Loop Parameter Exception
- Paint Map
- Plot Results From FeetSLAM
- Ratio Curve
- Ratio Viewer
- Restrict Loop Factors
- Restrict Loop Range Factors
- Restrict Loop Step Factors
- Run Best Fit Iteratively
- Save all Maps As Pictures
- Starting Conditions and Transformations Curve
- Too Big Exception
- Transform
- Transformation GUI
- Transformation Interface
- Transformation Range
- Transformation Viewer
- Turbo FeetSLAM
- XY Chart

Some classes and interfaces written by P. Robertson were only extended, such as:
• Hexagon Aggregated Transition Map
• Hexagon Correlation Drawing Panel
• Hexagon Map Drawing Panel
• Hexagon Probability Map Abstract
• Hexagon Prob Map Transition Bayes
• Hexagon Transition Aggregated Soft
• Hexagon Transition Aggregated Soft Interface
• Hexagon Transition Integer Interface
• Hexagon Transition Probabilities Interface
• Hexagon Transition Probabilities Transition

And the rest of the classes, all written by P. Robertson, that were used, but not extended are:
• Hexagon
• Hexagon Center
• Hexagon Map Aggregation Processor
• Hexagon Map Frame
• Hexagon Map Interface
• Hexagon Map MAP Aggregation Processor
• Hexagon Map Posterior Aggregation Processor
• Hexagon Map Size Related Parameter Store
• Hexagon Matrix
• Hexagon Prior Map Abstract
• Hexagon Prior Map Interface
• Hexagon Prior Map Transition
• Hexagon Prob Map Dynamic Abstract
• Hexagon Prob Map Interface

Three more classes were extended, but reside in different packages:
• FootSLAM Starting Conditions class, also written by P. Robertson and stored in the bayesian folder, but in the dlr.solo package.
• Polygon Manager Doubles and Wall Manager classes, written by M. Khider and stored in the moscito folder under the moscito.visualization package.
**B.3 UML Diagrams**

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created, by the Object Management Group [34]. UML includes a set of graphic notation techniques to create visual models of object-oriented software systems.

In figures B.1 to B.5 some inheritance UML diagrams are shown. In figures B.6 to B.8, association UML diagrams are shown. These UML diagrams have been generated with the Eclipse plug-in called UML2 [35].

All the classes are contained in the package dlr.solo.particle.core.movement.GridFootSLAM, except when explicitly indicated under the name of the class, like in the case of the Logger class (under dlr.solo package), the FootSLAM Starting Conditions class (under dlr.solo.simulator package) and the Polygon Manager Doubles and Wall Manager classes (under moscito.visualization package).

The letter I means "interface" and the letter C means "class".

In an inheritance diagram, an arrow that goes from one class to an interface means that the class implements the interface. An arrow going from one interface to another interface or from one class to another class means that the first extends (inherits from) the second.

In an association diagram, an arrow going from one class or interface to another means that the first contains an instance of the latter. There is a multiplicity indicator on every arrow (see Table B.1) stating how many instances there are.

**B.3.1 Inheritance Diagrams**

In this subsection, a brief presentation of the main classes and interfaces shown in the inheritance diagrams is provided. In Figure B.1, the classes and interfaces for the implementation of transitions across the edges of one single hexagon are shown.

- Hexagon Transition Probabilities Interface: a common interface for the two kinds of hexagon transitions across the edges: integer and aggregated.

- Hexagon Transition Aggregated Soft Interface: an interface for the Hexagon Transition Aggregated Soft class, the one used for Aggregated Maps.

- Hexagon Transition Integer Interface: an interface for the Hexagon Transition Probabilities Transition class, the one used for Best Maps.

- Hexagon Transition Aggregated Soft: the class that implements the methods of the Hexagon Transition Aggregated Soft Interface. It is used to compose an Aggregated Map, in which the transitions across the edges of the hexagons are the result of the addition of the "integer" transition counts of all the particles that were used to generate the map.

- Hexagon Transition Probabilities Transition: the class that implements the methods of the Hexagon Transition Integer Interface. It is used to compose a Best Map, in which the transitions across the edges of the hexagons are just the number of times the particle with the maximum likelihood [19] crossed each one of them.
In Figure B.2, the classes and interfaces for the generation of the different kinds of maps are shown. The most important classes are:

- Hexagon Prob Map Transition Bayes: a class that implements Best Maps.
- Hexagon Aggregated Transition Map: a class that implements Aggregated Maps.
- Hexagon Prior Map Transition: a class that implements Prior Maps.
- Hexagon Probability Map Abstract: an abstract class that implements the FootSLAM Map Interface used in most of the applications implemented for this thesis.

In Figure B.3, some of the classes and interfaces for the transformation, correlation and coordinate system are shown. The Transform class implements the Transformation Interface and the main methods refer to the manipulation of the four parameters of the transformation (rotation, scale factor and x and y shift). The Hexagon Matrix implements the Coordinate System Interface, which itself does not have any method signatures. The Correlation Interface serves as an interface for the user to obtain the symmetric and asymmetric parts of the correlation between two maps.

Figure B.4 shows the main classes used for visualization of results. The main classes here were explained in Appendix A.

Finally, Figure B.5 shows the rest of the most important classes that were implemented for the "Turbo" FeetSLAM algorithm and its testing and their main methods.

### B.3.2 Association Diagrams

This section aims to provide some examples for the association between some of the classes that were implemented for this thesis. Figure B.6 shows the association diagram for the Log Likelihood Function class. The Log Likelihood Function class relies on the use of two FootSLAM Map Interfaces and a Hexagon Matrix that provides a frame for the operations. The Log Likelihood Function class is instanced in the Correlation Viewer class, which also uses a FootSLAM Map Interface for a combined map and the same Hexagon Matrix for the coordinate frame.

Figures B.7 and B.8 show two more examples of these association diagrams, visually more complex, but that follow the same principle that was explained for the Log Likelihood Function class.
Figure B.1: UML inheritance diagram for the transition counts on a hexagon level.
Figure B.2: UML inheritance diagram for the transition counts on a map level.
Figure B.3: UML inheritance diagram with for the transformation, correlation and coordinate system.
Figure B.4: Illustration of the main visualization classes and their most important methods.
Figure B.5: Illustration of some other main classes and their most important methods.
Figure B.6: UML association diagram for the log likelihood function.

Figure B.7: UML association diagram for the Transformation Viewer application.
Figure B.8: UML association diagram for the Hexagon Matrix class.