Human following behavior for an autonomous mobile robot

Caroline Queva

KTH Datavetenskap
och kommunikation

Degree project in
Computer Science
Second cycle
Stockholm, Sweden 2013

KTH Computer Science
and Communication

Human following behavior for an autonomous mobile robot

QUEVA CAROLINE

August 2013
Master's Thesis at NADA
In collaboration with AWABOT (France)
Supervisor: Patric Jensfelt
Examiner: Stefan Carlsson

Abstract

Robots are used more and more to change people's life. In recent times, some robots have found their place in houses. Those robots are specific ones as for example robotic vacuum cleaners. The next step in robotics' evolution is to make people live with robots in everyday life.

This project attempts to make a robot follow a person. To achieve that, a people detector using a RGB-D sensor is implemented. The result of this detector is sent to a tracking algorithm using depth segmentation and a region growing algorithm. At the end, a navigation algorithm containing path planning and obstacle avoidance is used to make the mobile robot move.

This method can make the robot follow a person that walks slowly in front of it. Suggestions are presented to improve the method and adapt it to everyday life.

Thanks to this study, mobile robots can be used in personal home and assist people. An application could be for example to follow a person that needs medical assistance and help this person when necessary.

Referat

Människoföljande beteenden för autonoma mobila robotar

Robotar används mer och mer för att förbättra människors liv. På senare tid har robotarna anlänt i vara hus. Dessa robotar är specialiserade, till exempel är de robotiserade dammsugare. I denna robotikevolution, är nästa steg att få människor att leva varje dag med robotar.

Det här projektet försöker få en robot att följa en person. För att uppnå målet, har en detektor för människor implementerats, baserat pa en RGB-D sensor. Denna detektors resultat sänds till en följningsalgoritm som använder segmentering baserat pa djupinformation och region growing. Till slut, används ett navigeringssystem som innehåller vägplanering och hinderundvikande för att få den mobila roboten att röra sig säkert i omgivningen.

Den här metoden får roboten att följa en person som går sakta framför roboten. Flera förslag till förbättringar föreslås för att hantera alla de komplikationer som vardagliga miljöer erbjuder.

Tack vare det här projektet, kan mobila robotar användas i husen och hjälpa personer. En tillämpning är till exempel att följa en person som behöver medicinsk bistånd och hjälpa den här personen när det är nödvändigt.

Preface

This report is a part of a degree project performed at the School of Computer Science and Communication (CSC) at KTH and in collaboration with Awabot, a French robotics company.

Many thanks to my supervisor at Awabot, Renaud Didier, for his help to make the project possible and for his valuable feedbacks during the project. I also want to thank all my collaborators in the company (in particular Mathieu Turiot, Lucie Ringeval, Alexis Troncy and Charles Mazars) that helped me and advised me during this project.

Then, I want to thank my supervisor at CSC, Patric Jensfelt, for his help and for providing me feedbacks during all this project.

At the end, many thanks to my family and Thomas Cauwet for supporting me during these six months.

Figure 1 shows one of the robot created by Awabot company. This robot is not the one used in this project, the robot used for testing is described in chapter 6.

Figure 1. Emox, the augmented robot : Robot created by Awabot.

Contents

Chapter 1

Introduction

Robotics is today a quickly evolving world. There are more and more tasks in the world which are done by robots. Robots are used in many fields to assist humans in difficult tasks. In 1997, the robot Pathfinder landed on Mars for an exploration mission. Space exploration was one of the first field in which robots were needed to assist humans. Then robots became increasingly used. In the military area, robots are used for exploration but also with the development of tools like powered exoskeletons, robotics can assist soldiers. Another type of robots are medical robots, which are robots that help surgeons and improve some of their abilities. And then, the first robots have now found their place in some houses, these are robotic vacuum cleaners.

1.1 Problem statement

The number of people that show interest in autonomous robots, as robotic vacuum cleaners or robots that autonomously mow the lawn for example, is increasing every year. The challenge is now not only to make robots help people in everyday life but to make them live with people. The first step is to create a robot which is able to live autonomously in people's houses. Because robots are becoming entire individuals and not tools that people would turn on from time to time and forget in a closet at all other times. The second step is to create robots' behaviors to make people consider robots as living entities and not only as objects. This evolution of robotics is a very huge area to study. In this project, the topic is to make a robot follow a person. To be able to do that, the project first studies how to make a robot navigate autonomously inside a house and then how to detect and track a human shape thanks to a RGB-D (Red,Green,Blue and Depth) sensor. A RGB-D sensor is a camera using color images and depth images.

1.2 Contributions of the thesis

Many algorithms have been done for localization, mapping and navigation. For the navigation part, the work of the thesis is to tune the algorithm's parameters to adapt it to the robot used for testing. Concerning people detection, a specific algorithm is studied and tested. The results are then compared with those of another algorithm to find the drawbacks of each algorithm. The main focus of the thesis is the design and implementation of the tracking algorithm that uses the people detector's output. At the end, everything is integrated together to make the robot follow people in a house.

1.3 Assumptions and delimitations

- The mechanical part of the robot is done and working well. All the drivers are already implemented.
- The robot only moves on a planar ground and in an indoor environment.
- The map of the environment is done in a static world but is used in a dynamic one (only people are moving).
- The robot is never lifted up and put in another place.
- The robot is used with adults and not with children.
- During human following, the human is walking and not running.

1.4 Hardware

A specific constraint of this study is the hardware components. The robot can move thanks to a mobile base using two differential motors. A computer is used to run all the algorithms, but this computer is limited, so the algorithms' consumptions have to be minimized. For mapping and navigation, the robot has a laser range-finder, a good odometry and bumpers. For people detection and tracking, a RGB-D sensor is used.

1.5 Outline

In Chapter 2 related work is described. First, theory about mapping and navigation algorithms is explained. Secondly, references to several people detection and tracking algorithms are discussed. In Chapter 3 the Rao Blackwellized particle filter is described and the particularities of the chosen algorithm are explained. At the end, a section explains how to improve navigation using a RGB-D sensor. In Chapter 4 a people detector using depth data is studied. Results are compared with another algorithm to deduce the drawbacks of the algorithm. Chapter 5 is the main

part of the project. A tracking algorithm is designed, implemented and tested. In Chapter 6, implementation of all the algorithms is described. Results and possible improvements are discussed.

Part I

Theory

Chapter 2

Related Work

Robotics and image analysis are fields in which technology improves every day, thus many papers are published. The following sections try to summarize some important papers and methods about topics studied in this report.

2.1 Mapping and navigation

Some of the main issues for autonomous robots are mapping and navigation. A robot needs to be able to navigate in its environment in a safe and reliable way. That means a robot needs to reach its goal avoiding static and dynamic obstacles.

2.1.1 Localization and Mapping

An autonomous robot needs to create a map of its environment and to localize itself in this map by comparing the perceived environment to the map. These two steps are commonly done in a single algorithm called SLAM (Simultaneous Localization And Mapping). SLAM is considered to be a complex problem because on one hand a robot needs a good estimation of its position to acquire a map and on the other hand it needs a precise map to get the position.

Map representation

Different approaches to SLAM problem exist and they can be first categorized by the type of map representation they use. Two common ways to represent a map are occupancy grids and landmark-based maps. Landmark-based maps assume that the environment is made of geometrical features. This assumption leads to a very compact map but as it relies on predefined feature extractors, it is necessary to know in advance some structures of the environment. Whereas occupancy grids can represent arbitrarily structures. These grid maps are a discretization of the environment into cells that can be free, occupied or unknown (see Figure 2.1). The main advantage of occupancy grids is the high level of accuracy obtained if an appropriate resolution of the grid is used; but the cost of this is a need for a huge

amount of memory. Occupancy grids can be 2D or 3D. 3D occupancy grids do not limit the use of the map to planar robots but few sensors can construct a 3D map (vision can be used for example) and these maps take even more memory. In this project, there is no information about the environment so the chosen map representation is an occupancy grid. Then as a planar robot is used and memory is limited, a 2D map is used.

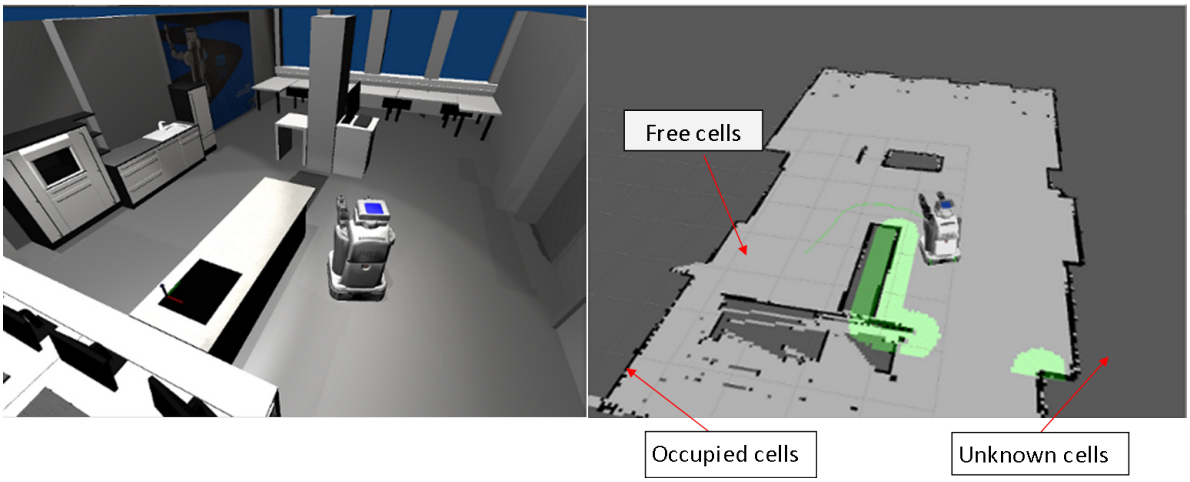


Figure 2.1. Example of a 2D occupancy grid. On the left, a simulated robot in a room. On the right, the map of the room.

Odometry and uncertainty

SLAM algorithms need to estimate both the map and the position of the robot in this map. To deduce the robot's pose, both the odometry and the observations of the environment are used. The odometry defines the relative robot pose between two successive points of time. And the information about the environment is given by sensors like a laser range-finder or a camera.

In a known environment, the uncertainty of the robot's pose is low because each observation of the environment can be compared to the known map. But in an unknown environment, the uncertainty is larger. The odometry is not perfect and the error increases over time. To reduce this error, the information of the sensors needs to be compared to the map, but the map contains also uncertainty because it is not a-priori known.

SLAM formulation

As explained in [1] the parameters for the SLAM problem are :

- $\mathbf{x}_k$: The state vector describing the location and orientation of the robot.

- $\mathbf{u}_k$: The control vector, applied at time k-1 to drive the robot to a state $\mathbf{x}_k$ at time k. This control vector uses the odometry to give a first estimate of state $\mathbf{x}_k$.
- $\mathbf{U}_{0:k}$: Control inputs from time 0 to time k.
- $\mathbf{m}$: The map, for a landmark-based map this should be a vector, for a grid this should be a matrix.
- $\mathbf{z}_k$: The observation vector at time k.
- $\mathbf{Z}_{0:k}$: Recorded observations from time 0 to time k.

In SLAM, it is necessary to compute for all time k the following probability distribution

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0)$$

This probability distribution describes the map estimation and the robot state at time k given the recorded observations and control inputs up to time k and the initial state of the robot.

To find this probability, first an observation model and a motion model are defined. The observation model is the probability of having an observation $\mathbf{z}_k$ knowing the map and the robot pose.

$$P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m})$$

The motion model is the probability of the state $\mathbf{x}_k$ knowing the previous state $\mathbf{x}_{k-1}$ and the control input $\mathbf{u}_k$.

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k)$$

SLAM is a problem that can be solved by many different algorithms. The most used algorithms are Extended Kalman Filter (EKF), graph-based SLAM and particle filters.

Extended Kalman Filter

The Extended Kalman Filter is the most popular SLAM algorithm. This algorithm is the extended version of the Kalman Filter for non-linear problems. Maps in EKF SLAM are typically feature-based, a state vector is then defined which comprises both the robot's pose and the position of the n features in the map :

$$\mathbf{y}_k = [\mathbf{x}_k, \mathbf{m}_0, \mathbf{m}_0, ..., \mathbf{m}_{n-1}]$$

As described in many different papers (for example in [2], [3], [4], [5]) the algorithm can be divided into two parts : prediction update and correction update.

During prediction update, the robot's pose is predicted using the last estimate $\mathbf{x}_{k-1}$ and the model of motion of the robot $\mathbf{u}_k$.

The correction update is composed of multiple steps. First the observation step during which the robot collects sensor data and extracts features. Then, the matching step during which the robot computes the best matching between extracted

features and expected features from the estimated position. Finally, the estimation step consists of using the matches to update the robot state and map estimations.

A strong assumption of this method is that the odometric error model and the measurement error model are both Gaussian distributions. Then, at each step of the algorithm, the mean and covariance of these Gaussian distributions are updated. The EKF algorithm is known to be an efficient and accurate algorithm but the assumption and the need for uniquely identifiable landmarks for matching are important drawbacks. Moreover, the computation cost is important and increases quadratically with the number of features.

In this project, an occupancy grid is used for which an EKF is not well suited. In conclusion another algorithm needs to be found.

Graph-based SLAM

In the graph-based method, the SLAM problem is interpreted as a graph of nodes and constraints between nodes ([2], [6], [7]) . Nodes correspond to poses of the robot and sensor measurements. Every edge between two nodes corresponds to the spatial constraints between them. The constraints are both the relative position between consecutive robot poses and the relative position between robot's poses and features observed from those locations (see Figure 2.2).

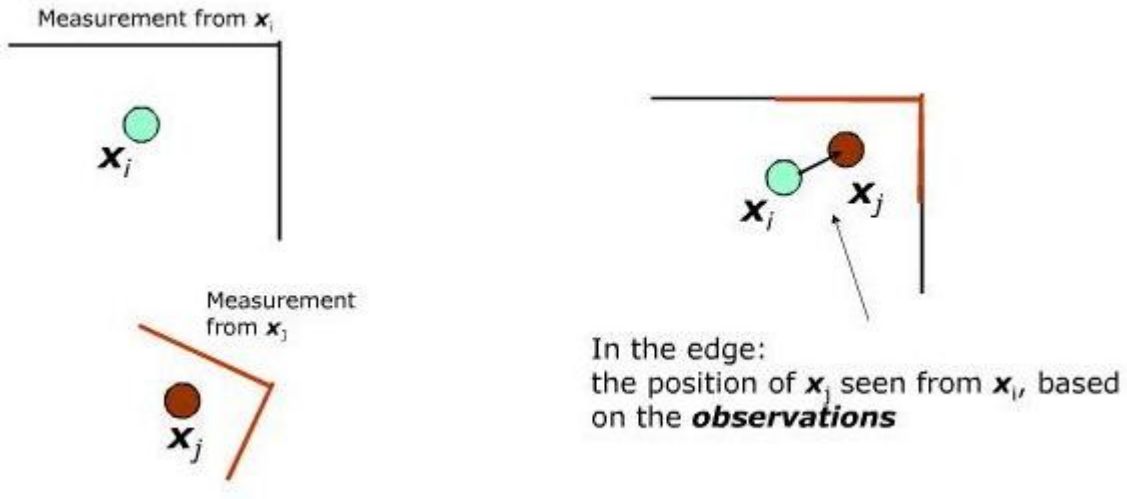


Figure 2.2. On the left : Measurements from the sensor at time i and j. On the right : Construction of the edge based on measurements (Image from [8]).

The solution of the SLAM problem is computed as the configuration of nodes that minimizes the error introduced by the constraints. One of the method generally used is Least Square optimization. When the solution is found, which means the successive poses of the robot are estimated, the map can be deduced from the graph.

The advantage of graph-based SLAM is that the update time of the graph does not increase so much with the size of the map. However, the final graph optimization (step during which the optimal configuration of nodes is deduced) can become computationally costly when the robot's path is long.

Particle filter

As explained in papers [2] and [9]; particle filter is the third major algorithm used for SLAM. In the EKF algorithm the probability distribution is represented in a parametric form (a Gaussian). Whereas in particle filter methods, the probability distribution is represented as a set of particles and these particles are drawn randomly from the parametric form of the distribution (see Figure 2.3).

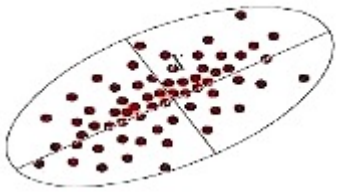


Figure 2.3. Particles obtained from a Gaussian distribution (Image from [10]).

Each particle contains an estimate of the robot path $\mathbf{x}_k$ and an estimate of the map. The different steps of this algorithm are :

- The robot moves and a new location $\mathbf{x}_k$ is generated for each particle using the motion model and the last state $\mathbf{x}_{k-1}$.
- The robot makes an observation $\mathbf{z}_k$, the 'importance factor' $\mathbf{w}_k$ is computed for each particle. This factor corresponds to the probability of having $\mathbf{z}_k$ given the particle and all previous observations $\mathbf{Z}_{0:k-1}$. Particles whose predictions match the real measurements are given a higher weights.
- Re-sampling is done. The current set of particles is replaced by a new set using this 'importance factor' as a weight.
- The map estimations are updated.

The huge difference between particle filters and EKF algorithms is that a particle filter can represent non-Gaussian distributions. Moreover state and observation models can be non-linear. But the major drawbacks of particle filters are the high number of particles required to build an accurate map and the particle depletion problem. The particle depletion problem is the possibility to eliminate correct particles during re-sampling step.

Conclusion

EKF can not easily be used in this project as an occupancy grid was chosen. To avoid graph optimization in graph-based SLAM, the particle filter method is selected. In [9], a method is described to reduce both the number of particles and the risk for particle depletion. This method uses a Rao-Blackwellized particle filter and will be explained in more details in the next chapter.

2.1.2 Path planning

Once the map is created, the robot needs algorithms to navigate inside this map. For navigation, the robot requires two components : path planning and obstacle avoidance. Path planning involves calculating the best trajectory to reach a given goal.

A lot of approaches have been proposed for solving path planning. The two major groups of algorithms are graph search and potential field algorithms. In this project, only graph methods are studied because these methods are simple to use with occupancy grids.

Graph search techniques involve two main steps which are graph construction and graph search. During graph construction, nodes are placed and edges are created to connect the nodes. During graph search, an optimal solution to go from one place to another is computed.

Graph construction

Graph construction can be done in different ways, the goal is to create a graph that enables the robot to go anywhere in the free space while limiting the size of the graph. Two methods are generally known as road map approaches, Visibility graph and Voronoi diagram. Another method uses approximate cell decomposition to construct the graph.

In a visibility graph (Figure 2.4), nodes of the graph are the initial position, the goal position and the vertices of the obstacles. All nodes that can see each other are connected by straight-line segments. The edges are the roads, and the path planner goal is to find the shortest path using these roads.

The advantage of visibility graph is that the resulting path is a minimum-length solution and the method is simple. But the roads come as close as possible to obstacles, for this reason generally the obstacles are inflated by the robot radius. This last solution helps keeping the robot at a minimum security distance from the obstacles.

Another problem with visibility graph methods is the assumption that each obstacle is a polygon. In the map, some obstacles can for example be circular.

On the contrary, Voronoi diagrams (Figure 2.5) try to maximize the distance between the robot and the obstacles. The edges are computed as points that are equidistant from two or more obstacles.

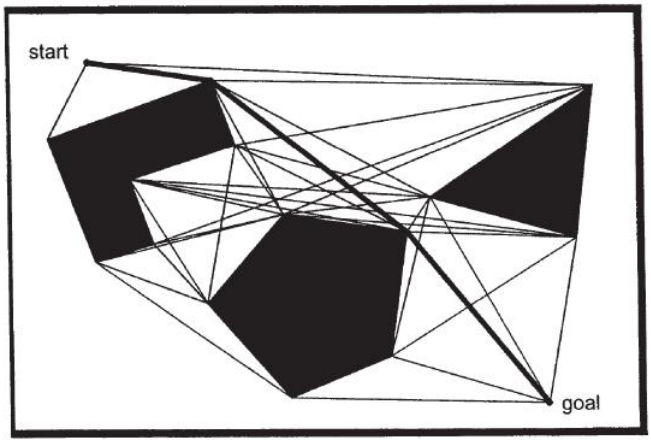


Figure 2.4. Visibility graph (Image from [2]).

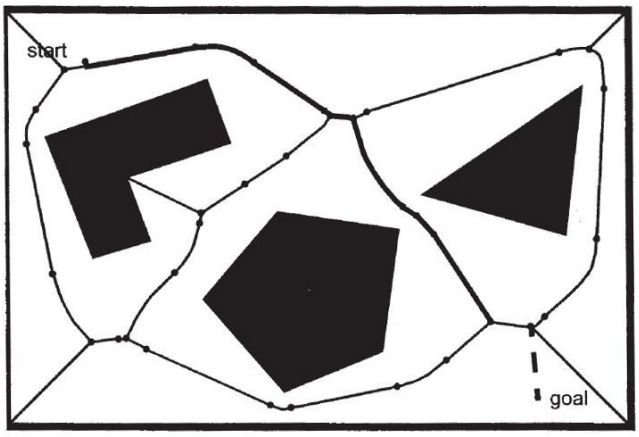


Figure 2.5. Voronoi diagram (Image from [2]).

This method can be easily used by robots with long-range sensors. In order to move along the edges of the Voronoi diagram, these robots need only to use their sensors to stay at equal distance from all obstacles. But the disadvantage is that long range sensors are necessary. As the path stays as far away as possible from obstacles, the short-range sensors would tend to not see the obstacles. Another disadvantage is that if an obstacle is moved in the environment, the changes in the graph can be important

Approximate cell decomposition (Figure 2.6) is another method which separates the image between free and occupied areas. This method creates an approximation of the real map. With an occupancy grid, the map representation is a fixed-size

decomposition, so approximate cell decomposition is an easy and popular method.

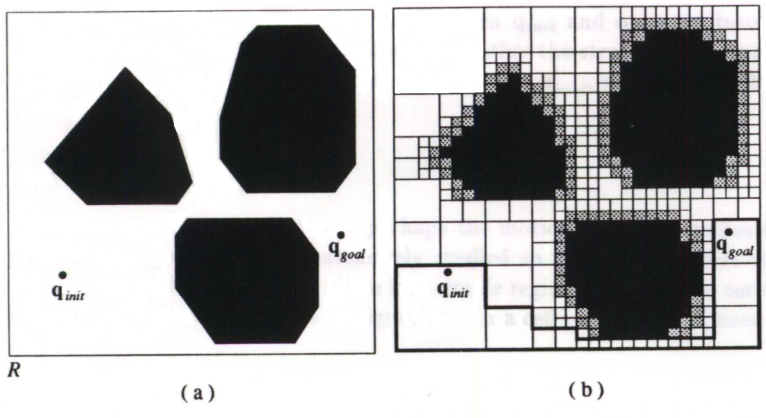


Figure 2.6. Approximate Cell Decomposition with a variable sized grid (Image from [2]).

The cell size does not depend of the size of obstacles. One disadvantage is that narrow spaces can be lost by the map's approximation.

Generally the obstacles can be inflated by the radius of the robot (in gray in Figure 2.6) to be sure the robot keeps a minimum distance to the obstacles. This additional step is another risk of loosing narrow spaces. One advantage of this method is a low computational cost.

Graph search

Once the graph is constructed, an algorithm is necessary to compute the shortest path from initial point to goal point. As many topics in robotics, a lot of researches have been made on graph search algorithms. Three of the most known methods are explained here : Breadth-first search, Dijkstra's algorithm and A*.

Breadth-first search explores all nodes, beginning with the start node and then looking all of its neighboring nodes, and their neighboring nodes, and so on. The algorithm follows this node expansion method until it reaches the goal node (Figure 2.7).

A popular example of breadth-first search is the wavefront expansion algorithm. This approach uses wavefront expansion from the goal position, marking all cells with its distance to the goal position, until the initial position is reached (Figure 2.8). Then computing the shortest path is easy. This algorithm is known to be efficient, and the search is linear in the number of cells.

Dijkstra's algorithm is similar to breadth-first search (BFS). But nodes are not visited in the same way. Instead of taking the neighboring node which has the

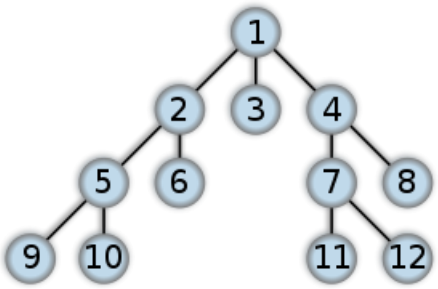


Figure 2.7. Breadth First Search.

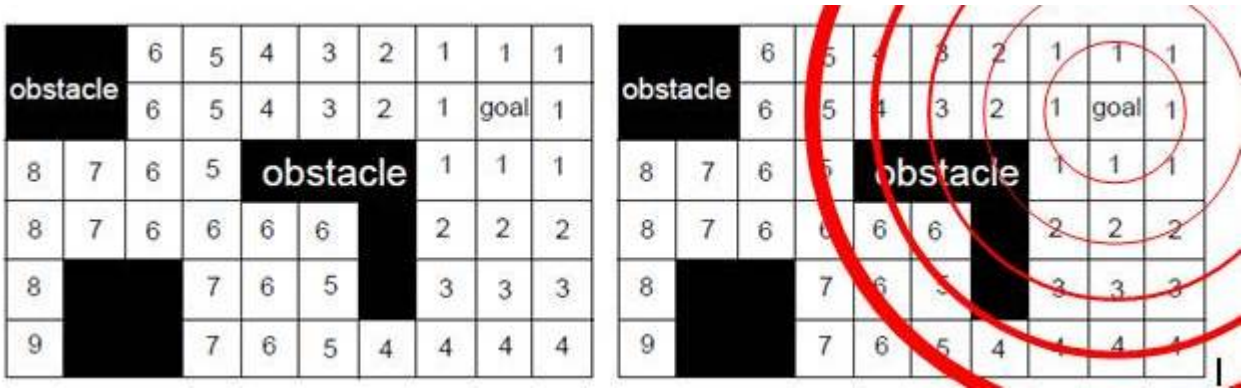


Figure 2.8. Wavefront Expansion.

shortest number of edges in the tree (as done in BFS), Dijkstra's algorithm places the node in a tree structure called heap. Nodes in the heap are then reordered according to the expected total path from the initial node to that given node. This expected total path is known as the cost function. And then the first node of the heap is taken and expanded. This process continues until it reaches the goal point. The optimal solution can then be backtracked from the goal.

A* algorithm is an alternative to Dijkstra's algorithm. It adds an heuristic function h(n) to the cost function g(n). Nodes are then reordered using h(n)+g(n). The heuristic function contains additional information about the graph, for example it could contain the distance between the cell and the goal in absence of any obstacle. The time complexity of A* depends on the heuristic h(n) but better performance can be expected with this algorithm.

2.1.3 Obstacle avoidance

Obstacle avoidance is to avoid collisions with obstacles while following a chosen trajectory. Obstacle avoidance algorithms are local and use sensors to avoid obstacles

during robot motion. Obstacle avoidance is an important part of navigation, even more important when the environment is dynamic. In a dynamic environment, the map is done in a particular state of the environment. This map is then used with path planning methods to calculate the shortest path to the goal point. But if a person appears in front of the robot, and this person was not here during mapping, the robot needs to use its sensors to modify the path in order to avoid this person.

In the mobile robotics community, a great number of obstacle avoidance methods has been presented. One of the simplest is the bug algorithm [11]. The idea of this algorithm is to follow the contour of each obstacle the robot finds on its way.

Then, Borenstein and Koren [12] developed the vector field histogram. This technique creates a local map of the environment around the robot and generates a polar histogram to indicate the probability to find an obstacle in this direction.

Another technique is called Dynamic Window Approach and takes into account the kinematics of the robot. A velocity-space is used, this space represents all possible tuples (v,w) where v is the linear velocity and w is the angular velocity. Given the current robot speed, the algorithm first selects a dynamic window which contains all points in this velocity space the robot can reached within the next period (see Figure 2.9). The next step of the algorithm is to reduce the window, keeping only the speeds that ensure no collision with an obstacle. Then, a new motion direction is chosen in order to maintain a minimum distance from the obstacles and move towards the goal.

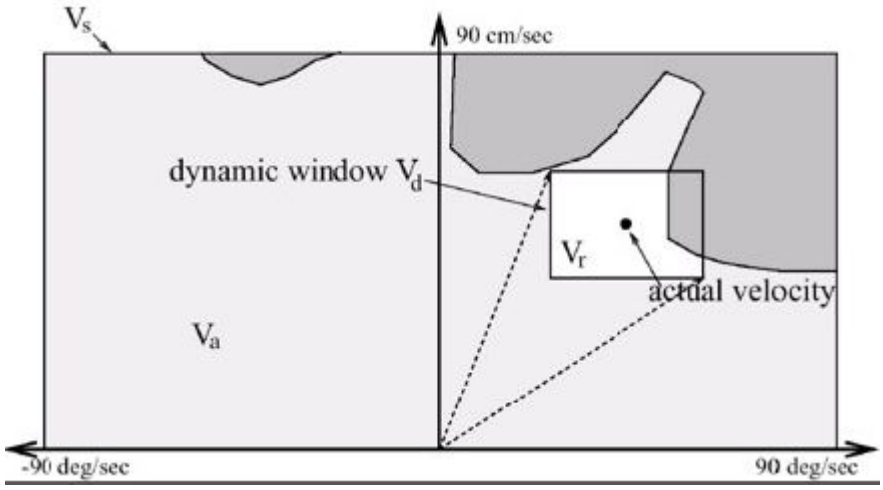


Figure 2.9. Dynamic Window Approach : the rectangular window represents the possible speeds (v,w) (Image from [2]).

2.2 People detection and tracking

Detecting people is one of the most important challenges for many applications where human interaction is involved. People detection is the process of finding people in the robot's environment. The person can be found using data from multiple sensors. In this project, hardware was defined in advance and a camera must be used. People detection is redefined as the process of finding people in an image obtained from a camera.

The first algorithms described in this section are using a color image taken from one or several cameras that can be static or moving.

2.2.1 Learning methods to detect people

Some state of the art methods rely on supervised learning methods. Work made by Andriluka [13] and Pischulin [14] are also based on Pictorial structures model. This model represents the body with an appearance model.

The problem of learning methods is to acquire training data. These methods are efficient for a static camera. In this project, the robot is moving, so it is difficult to acquire all relevant variations necessary to detect people.

2.2.2 Movement detection using background subtraction

Other methods use movement detection to find dynamic things in the image. With this method, people, animals or moving objects are segmented from the rest of the image. Movement detection compares a reference background frame and the current input frame or compares the last frame with the current frame. Once the moving object is detected, a classifier is used to know if this moving object is a human or not.

Movement detection is a lot more difficult to do with a moving camera, as all the scene is moving, so it is not an optimal solution for this project.

2.2.3 Use RGB-D data to detect people

The invention of RGB-D sensors has created a revolution in image algorithms. The RGB-D sensors produce both a color image and a depth image. The latest gives information about the depth value of each pixel in the image.

Several algorithms have been made to detect people. In [15], Munaro describes a people detector using sub-clustering and an Histogram Of Oriented Gradients (HOG). In [16], Spinello uses an Histogram Of Oriented Depths (HOD) to create a people detector. This HOD takes inspiration from the HOG used in image analysis.

The method presented in [15] will be described in more details in Chapter 4.

2.2.4 People tracking using appearance and motion models

People tracking is generally done using people detection. First, people detection algorithms are used, then the people detected are classified using appearance and motion to associate each person with the persons detected in the previous frame.

In [15], the classifier uses color appearance and is based on Adaboost. Then a motion model is taken into account to have a more precise classification. The results are a probability for a detected person to be the same than another person detected in the previous frame.The drawback of this method is that the people detector is needed at each frame. The tracking algorithm proposed in this project needs the people detector only for initialization. This can reduce the processor's cost.

Chapter 3

Mapping and navigation

Mapping and navigation are not the central parts of this project. However, to have a robot that follows a person in a dynamic environment, it is necessary for the robot to be able to navigate in a safe way, avoiding obstacles.

3.1 Rao Blackwellized particle filter

Rao-Blackwellized particle filter (RBPF) is an extension of particle filters explained in Chapter 2. Each particle in a RBPF represents a possible state of the robot and a map.

In RBPF, the posterior probability of state $\mathbf{x}_k$ given the observations $\mathbf{Z}_{0:k}$ and the odometry measurements $\mathbf{U}_{0:k}$ is first calculated. And then this probability is used to deduce the posterior probability of both the states and map :

$$P(\mathbf{x}_{0:k}, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}) = P(\mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{x}_{0:k}) * P(\mathbf{x}_{0:k}, |\mathbf{Z}_{0:k}, \mathbf{U}_{0:k})$$

This can be done because the map's probability can be easily computed when the trajectories and the observations are both known.

The posterior probability of the robot's states given the observations and odometry measurements is similar to a localization problem. To estimate this probability, a particle filter can be used where each particle represents a potential trajectory of the robot. An individual map, built from the observations and the potential trajectory, is also associated to each particle.

The RBPF algorithm (described in [9]) contains the following steps :

- Sampling : The next generation of particles is obtained from the current generation. This new generation is calculated by sampling a function, generally the probabilistic odometry motion model. This function is known as the proposal distribution.
- Importance Weighting : An importance weight is calculated for each particle. The weight takes into account the fact that the target distribution just cal-

culated by sampling is not perfectly equal to the proposal distribution. The weight is generally calculated from the previous weight.

- Re-sampling : Re-sampling is done using the previous weights. At the end of this step, all particles have the same weight.
- Map update : For each particle, the map estimate is computed using the trajectory and the observations.

As the maps are quite memory expensive, especially with occupancy grids, and each particle maintains its own map; it is important to optimize the algorithm in order to keep the number of particles small.

3.2 Mapping algorithm

In [9], Grisetti proposed two concepts to improve RBPF implementation. The mapping implementation used in this project is based on these concepts.

In the RBPF algorithm, a function is used to sample the next generation of particles. Generally, the probabilistic odometry model is used. The first improvement proposed by Grisetti is to use a better proposal distribution for sampling. This can improve the algorithm if the sensor is more precise than the motion estimation based on the odometry. In this project, a laser range finder is used, it seems important to take it into account in the calculation of this function. The proposal distribution should approximate the true distribution :

$$P(\mathbf{x}_{0:k}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k})$$

In the paper discussed before, Grisetti obtains a new proposal distribution instead of the odometry model and the results are good (see Figure 3.1).

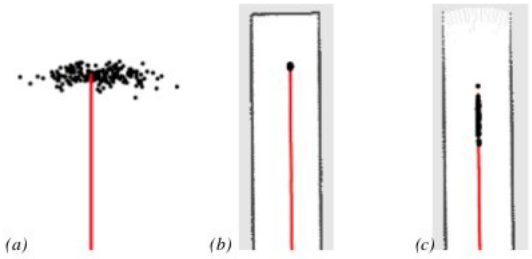


Figure 3.1. Proposal distributions typically observed during mapping. In a featureless open space the proposal distribution is the raw odometry motion model (a). In a dead end corridor the uncertainty is small in all of directions (b). In an open corridor the particles distributes along the corridor (c). (Image from [9]).

As the re-sampling step is a key part of the RBPF algorithm, the second improvement tries to optimize this step. During re-sampling, particles with a low

importance weight are replaced by samples with a high weight. The problem that arises is the possibility to remove good particles. Grisetti proposed a way to find an optimal time to perform the re-sampling step, in order to decrease the risk of particle depletion.

In Appendix A, a description of the overall algorithm is presented.

3.2.1 Parameters tuning

To test this algorithm, the open source implementation called gmapping is used. Some parameters have to be tuned to optimize the use of the algorithm with the robot.

First some parameters have to be adapted to the laser range-finder, for example the parameter maxRange should be equal to the maximum range of the sensor. This parameter is used to mark space as free if no obstacle is seen by the sensor within this range.

Then, the gmapping algorithm takes into account the odometry error of the robot. As the robot used for testing has an accurate odometry, the odometry error needs to be reduced.

At end, the most important parameter is the number of particles. The higher this number,the more accurate the mapping process is. But, as said before, each particle maintains its own map so if the number of particles is large, the cost is high.

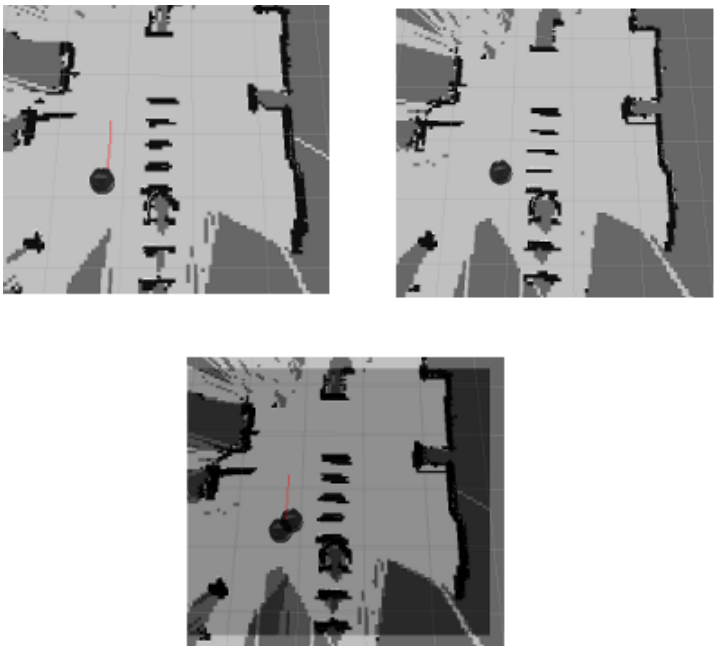


Figure 3.2. Top-left : Map using 10 particles, Top-right : Map using 100 particles, Bottom : superposition of both maps.

In Figure 3.2, two maps have been created using 10 and 100 particles. In these maps, the black circle is the robot, the light-gray is the free space, the dark-gray is the unknown space and the black is the obstacles (except for the robot).

The first thing to observe is that the map using 100 particles is more accurate than the map map using 10 particles, especially looking at the walls that are thicker in the map with less particles. Then, using 100 particles, the memory cost is higher, but the localization is more accurate and the robot is faster to map the environment. The third part of Figure 3.2 is the superposition of both map. This picture shows that even if in the map using 100 particles the obstacles are less thick, the global map is similar using 100 or 10 particles.

The number of particles chosen for mapping is 80 in order to get a compromise between an accurate map and a high cost. The map obtained is shown in Figure 3.3. In Figure 3.3, the part corresponding to Figure 3.2 is represented by the red square.

Figure 3.3. Map of the office.

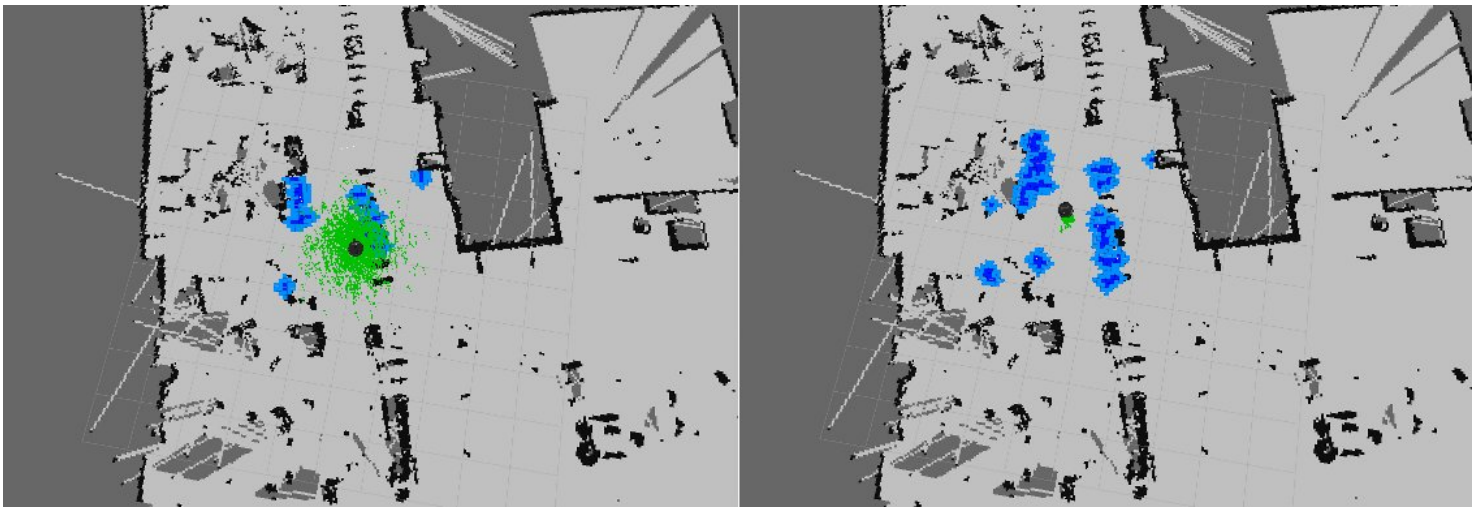


Figure 3.4. Localization thanks to a particle filter, the black circle is the robot, the green points are the particles. The blue part is the local map, explained in the next section. On the left, the robot's pose is far from accurate. On the right, the robot has moved several meters and used sensor data to reduce the uncertainty, the green particles are less spread.

3.3 Navigation

Navigation is done with a known map, created with the gmapping algorithm and saved. The map is then loaded and used for localization and path planning.

3.3.1 Localization

The robot needs to localize itself precisely first. For localization, as for mapping, a particle filter is used. The localization algorithm does not manage global localization, so when the robot is put at a specific place, the user has to give an approximate position to the algorithm. Then the particle filter uses the odometry measurement and the sensor data to improve the approximation of the position. This is illustrated in Figure 3.4.

3.3.2 Global and local planner

For navigation, two different planners are used. The first one calculates the path in a global way. The second one does obstacle avoidance in a local way.

The global planner uses the map created by the mapping algorithm to calculate the path. This map is static and not updated during navigation. A dynamic map is added to the static one in order to take into account moving obstacles like humans. This dynamic map is local and is used by the local planner to avoid obstacles.

In Figure 3.5, the static and dynamic maps are illustrated. As previously, the black circle is the robot and the green points are the particles. The light-gray is the

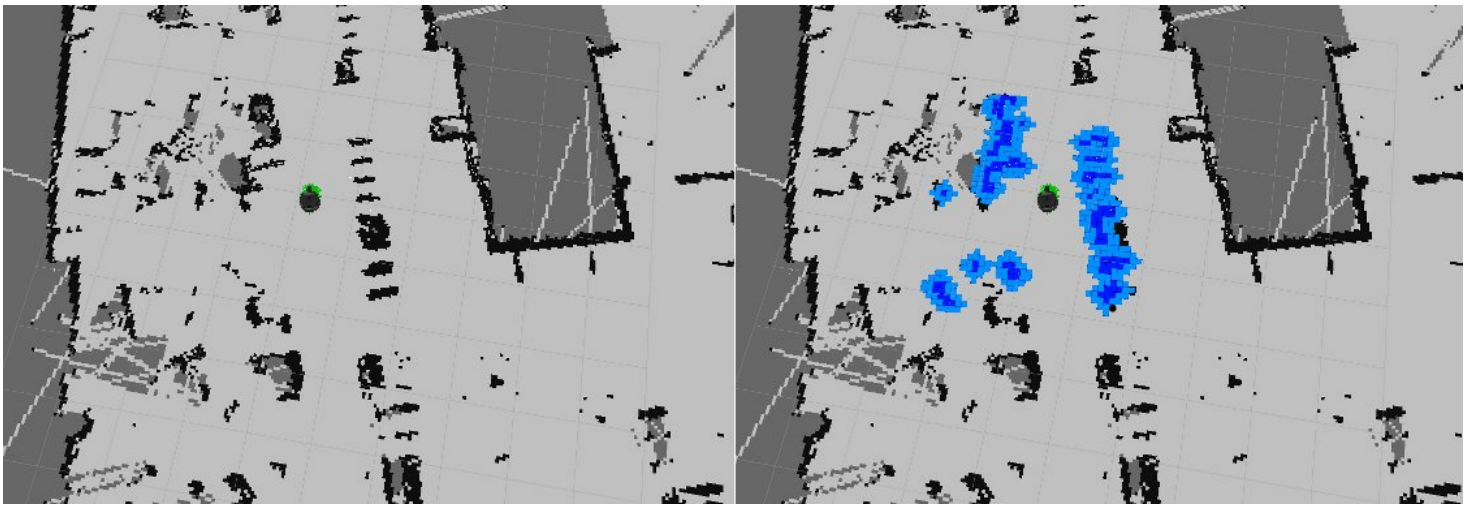


Figure 3.5. Comparison of a map with only the static map and a map with both static and dynamic maps. Blue is the dynamic map.

free space, the black is the obstacles and the dark-gray is the unknown space. On the left, only the static map appears; on the right, the dynamic map is superposed. In the dynamic map, the dark blue is the obstacles and the light blue is the inflated obstacles. The dynamic map is done only from sensor data and is local. Obstacles are marked only in the robot's neighborhood. The dynamic map is used for obstacle avoidance, if an obstacle was not in the static map but is detected by the laser sensor, the local planner is used to avoid it.

The global planner used for testing implements the Dijkstra's algorithm, and the local planner implements the Dynamic Window Approach.

3.3.3 Use of a RGB-D sensor

The dynamic map is created thanks to the sensor data from the laser range-finder.

As the laser range-finder gives only planar information and the height of the robot is non-zero, a problem appears with the previous method. If an object (for example a table) is above the laser range-finder, it is not detected by this sensor. Moreover if the object is not high enough for the robot to go below, a collision happens. If the laser is the only sensor used for obstacle avoidance this can not be solved, as illustrated in Figure 3.6

To avoid this problem, a RGB-D sensor is added. The RGB-D sensor is put above the laser range-finder. There are two methods to use this sensor.

- The simplest way is to use it as a planar sensor (see Figure 3.7).
- The second method is to take information given by the sensor between two heights (heightMin and heightMax), and to aggregate these 3D data in a 2D plan.

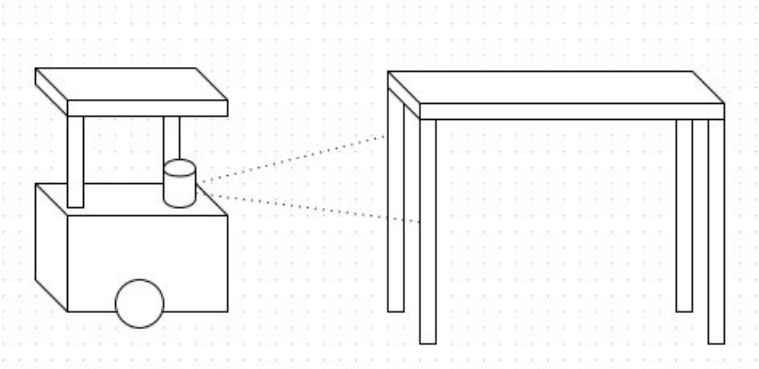


Figure 3.6. Illustration of possible collision if only a laser range-finder is used.

The first method is easier to implement, but the same problem could appear if an obstacle is between the laser plan and the camera plan. In the second method a value for heightMin just above the laser has to be chosen, so that an obstacle between the two sensors is seen by the camera.

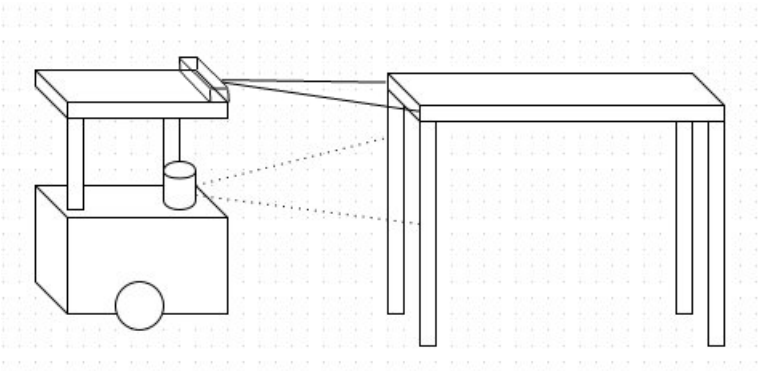


Figure 3.7. A RGB-D sensor is used to avoid some of the obstacles.

The second method is chosen for this project. To implement this method, the camera's data need to be added to the dynamic map used by the local planner. The local planner uses a 2D map called costmap. However, the original costmap contains only the data from the laser and no information about the height of the obstacle or their vertical position. To add data from the camera in a costmap two possibilities :

- Add the data in the same way that it is done for laser's data. Laser data and camera data are mixed in the costmap.
- Use a costmap that takes into account information about the vertical position of the obstacle.

The first way to add data can not be used because, as the map is dynamic, the laser can erase obstacles seen by the camera and vice versa. Below is an example of what can happen :

- The camera sees an obstacle, this obstacle is put in the 2D costmap.
- The laser does not see this obstacle, so it thinks there is free space.
- The laser puts free space in the costmap at the position of the obstacle. The obstacle is erased.

To avoid this problem, the costmap with information about the vertical position is chosen. The difference with a 3D map is that for this costmap, vertical position is only used to know which sensor (camera or laser) saw the obstacle. The final costmap is 2D but the knowledge about the vertical position helps to avoid that one sensor erases obstacles seen by another sensor.

In Figure 3.8 the improvement due to the RGB-D sensor is illustrated.

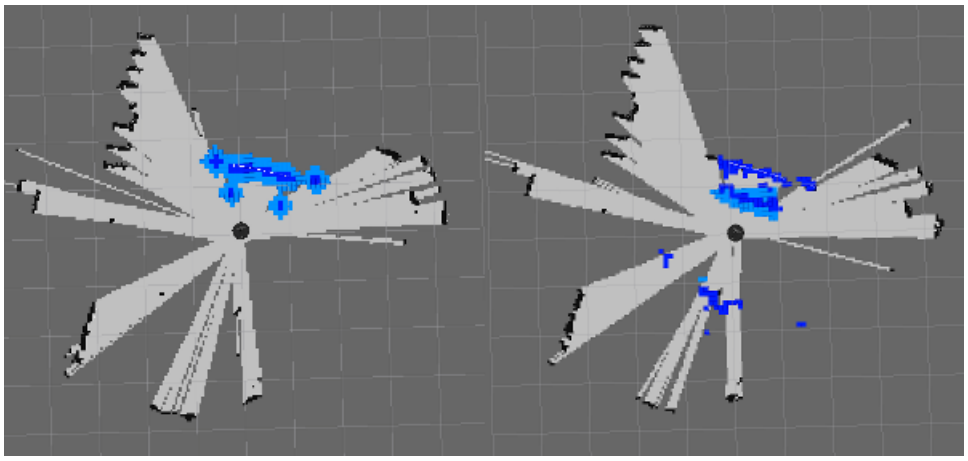


Figure 3.8. Effect of the RGB-D sensor on the costmap. On the left, the costmap without the camera, the blue is the wall behind a table, the two table's legs are just in front of the wall. On the right, the costmap using the RGB-D sensor, all the table appears as a blue obstacle.

The drawback of this method is the computation cost, lag issues appear during navigation using this new costmap.

Part II

Method and Implementation

Chapter 4

People detection

The main part of this project is to design an efficient algorithm to follow people for a robot with a laser range-finder and a RGB-D sensor. The navigation part was described in Chapter 3, the next step is to detect people in an image. As explained in Chapter 2, a RGB-D sensor is very useful to detect people. This sensor provides color data as a simple camera but also depth data.

4.1 Use of Point Cloud data

The depth data can be sent in two different ways : in a depth image or in a Point Cloud.

The depth image is similar to a color image except that the value of each pixel represents a depth (for example in millimeter or meter).

The Point Cloud is a set of points in a 3D-coordinate system. These points can also contain other information as for example the color of the corresponding pixel in the color image. When the color information is added, the point cloud becomes 4D (see Figure 4.1). With a RGB-D sensor, depth information and color information are used together to get the point cloud. Each pixel of the depth image is put in the point cloud using its image coordinates (x,y) and its depth value. The color image is used to add color information to each point in the point cloud.

For a people detection algorithm, both representations of depth data could be used. In this project, a point cloud is chosen because it represents each component of the 3D space and also the color of each pixel. With this representation, finding all the pixels of a plane (as for example the ground or the ceiling) is easier than in a depth image.

4.2 People detector in theory

In [15], Munaro proposes an algorithm to detect multiple people assuming they are moving on a ground plane. The main steps of the algorithm are the following ones :

Figure 4.1. Examples of Point Clouds with color information (Image from [17]).

- Data filtering, to reduce the size of the point cloud. This is done to improve real time performance.
- Ground removing, to separate parts of the point cloud from each other.
- Clustering, to put exactly one person in each cluster.
- People detector based on Histogram of Oriented Gradient, to classify each cluster into people or not people.

In the following sections, the algorithm will be explained in more details.

4.2.1 Filtering

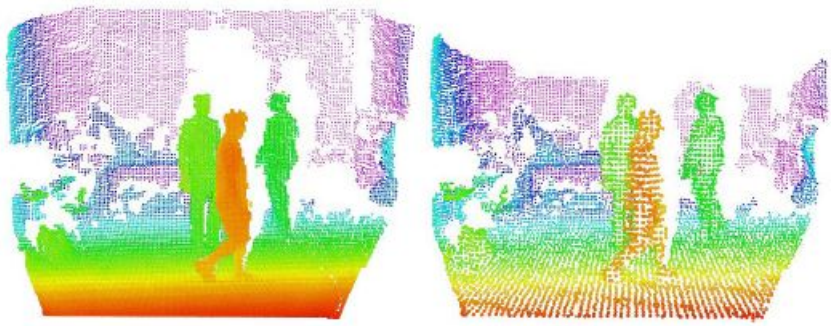
RGB-D sensors give images in a high resolution (640*480 pixels), so the number of points in each point cloud is high. As a high number of points involves a high computation cost, the goal of the filtering part is to reduce the size of the point cloud.

The filtering strategy consists of subdividing the 3D space into a set of fix-sized volumetric pixels (voxels). All the points in one voxel are merged into one point, and this point has the same coordinates as the centroid of the voxel.

Another advantage of this filter is to get an image with constant density. Because, in point cloud data, the number of points in a cluster depends on the distance between the cluster and the sensor. After filtering, as the size of the voxel is predefined, the density becomes constant. Then, the size of a cluster is directly proportional to the number of cells it contains. Figure 4.3 illustrates the effect of this filter.

4.2.2 Ground removing

When doing clustering in point cloud data a problem appears because most parts of the 3D space are connected through the ground. To be able to separate clusters, the



(a) Raw Kinect point cloud. (b) After voxel grid filtering.

Figure 4.2. The effect of the voxel filter (Image from [15]).

ground pixels need to be removed from the 3D space. In [15], an iterative method (RANdom SAmple Consensus) is used to estimate the ground coordinates using three ground points given by the user. In this project, the position and orientation of the camera on the robot are known, so the ground coordinates can be calculated using these data and the intrinsic parameters of the camera.

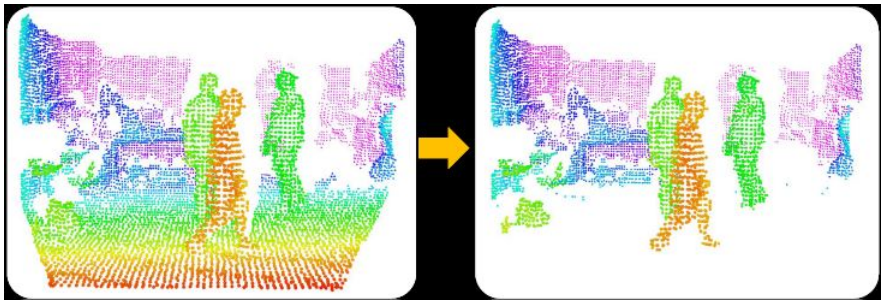


Figure 4.3. Ground removal (Image from [15]).

4.2.3 Clustering

To identify the different clusters in an image, the Euclidean distance is used. Based on this distance, the neighboring points are put in the same cluster. This can lead to two problems :

- A person can be subdivided into multiple clusters, due to partial occlusions for example.
- A cluster can contain more than one person, if those persons are very close to each other.

To avoid these problems and improve the clustering part of the algorithm, three steps are added.

First, in the group of clusters obtained, pruning is done based on the height of each cluster from the ground plane. A maximum height for a human person is defined and each cluster whose height is bigger than this maximum is removed.

Then, to avoid the first problem (a person in multiple clusters), clusters that are close to each other in ground plane coordinates are merged. To achieve that the center of each cluster is projected on the ground and compared to other clusters.

At the end, to separate several people in the same cluster, a head detection is done. This step is based on the assumptions that each person has exactly one head and the head is the highest part of the body. A height map is constructed, and the local maxima are found. For each local maximum, a cluster is created and all pixels near to the local maximum in ground plane coordinates are put in this cluster. Then, the clusters with too few points or whose height is less than the minimal height defined for a human are removed. See Figure 4.4.

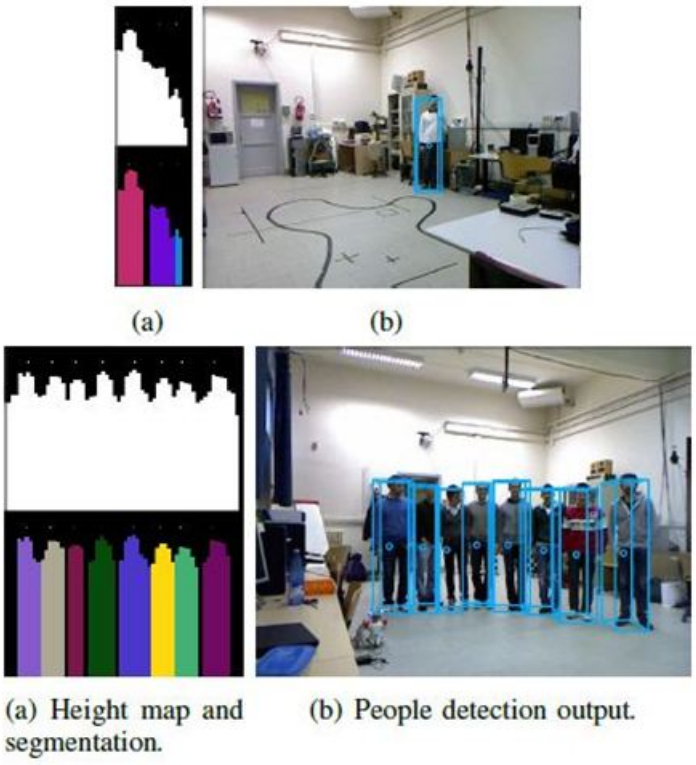


Figure 4.4. Sub-clustering of two different clusters : the top one contains a person previously merged with the background, the bottom one contains eight persons previously in the same cluster (Image from [15]).

Once those problems are solved, another step is necessary to counteract possible occlusions. Remaining clusters are extended to the feet of the person. If the feet of the person are not seen, the cluster is extended to the ground (Figure 4.5).

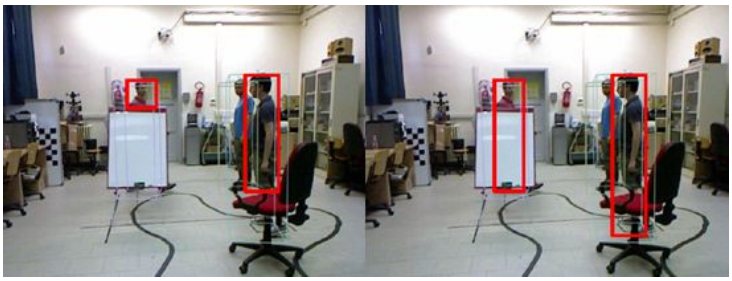


Figure 4.5. Extended clusters : improve robustness to occlusions (Image from [15]).

4.2.4 HOG people detector

The Histogram of Oriented Gradients

An Histogram of oriented Gradients is a feature descriptor used in image analysis and constructed as follows :

- The image is subdivided into regions and those regions are divided into cells.
- For each cell, the angle and magnitude of the gradient are computed.
- The angle and magnitude of each cell are used to compute a weighted histogram for the region. The result is then used as a descriptor.

Figures 4.6 and 4.7 illustrate the construction and result of an Histogram Of Gradient descriptor.

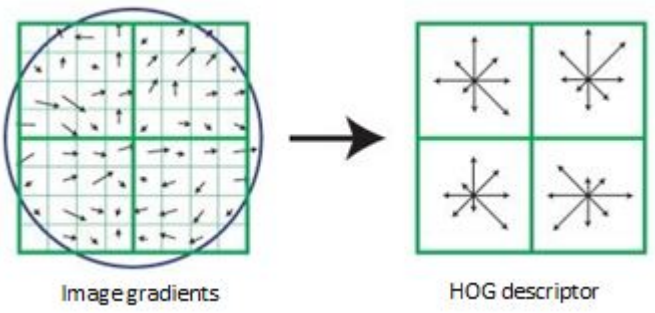


Figure 4.6. Construction of an Histogram of Oriented Gradient (Image from [18]).

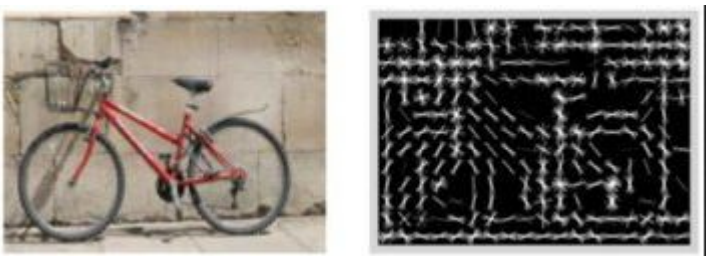


Figure 4.7. Result of an HOG descriptor, the brightness of each vector is proportional to the magnitude of the gradient. The left image is the original picture. The right image is the result of the HOG descriptor, the shape of the bicycle can be recognized. (Image from [18]).

People detector

As described in [19], the basic idea of using a HOG descriptor to detect people is that local appearance and shape can be characterized rather well by the distribution of local intensity gradients (as seen in Figure 4.7). After computing the HOG descriptor, a normalization is usually done to improve illumination invariance.

Once this normalized histogram is calculated, it is used to describe the cluster in a classifier. This classifier uses the descriptor to decide if the cluster represents a person or not.

A method of supervised learning called Support Vector Machine is used. A set of training examples is created. These examples are both images containing people and images without people. Each example in the training set is labeled as positive or negative. The classifier is first trained with the labeled examples before being used to classify new ones.

For each training example, the HOG descriptor is calculated and the data point is projected into a high dimensional space. Once each labeled example has been put in the high dimensional space, an hyperplane is calculated to separate the two classes of points (is a person or not). The hyperplane is chosen so that the distance to the nearest data point on each side is maximized as illustrated in Figure 4.8. Each new example position is compared to the hyperplane and classified as positive or negative example.

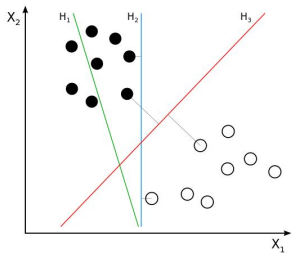


Figure 4.8. Illustration of the Support Vector Machine. Hyperplanes H2 and H3 separates the classes, but the margin is maximized for H3 (Image from [20]).

The result of the classifier is a number called confidence. If this confidence is high, the probability of having a person in the cluster is high. The threshold above which the cluster is classified as a person is difficult to set. A compromise needs to be done between false positives (if the threshold is too low) and false negatives (if the threshold is too high).

After applying the HOG detector, most of the clusters that do not contain a person are removed. The results are illustrated in Figure 4.9.

Figure 4.9. Left : clusters before applying HOG Detector, Right : resulting clusters after applying HOG Detector.

4.3 Results and comparison

Two main criteria are used to define the performance of the algorithm. The first one is the detection performance. To know if the algorithm has a high detection performance, it is necessary to know if the algorithm detects people, if it generates false positives and if it is robust. The second criterion is the execution time and cost.

4.3.1 Detection results

As seen in Figures 4.10 and 4.11, the algorithm detects accurately people in the image. It works for both people seen from the front or from behind. Moreover, the detector also gives good results when more than one person is in the image.

Figure 4.10. Detection of a person from point cloud data.

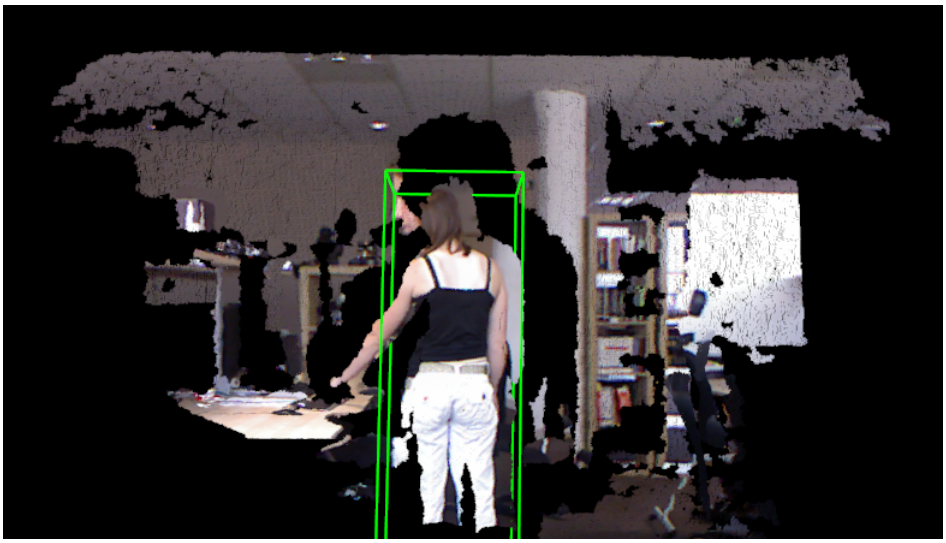


Figure 4.11. Detection of a person seen from behind.

However, Figure 4.12 illustrates both false positive and false negative detections. The false positive detections do not stay for a long time in the image. The detector is used on each frame of data given by the camera and the camera gives data at a frame-rate of 30 images per second. A false positive appears when the HOG detector gives a high value for a cluster that is not a person. Generally, on the following frame, the HOG detector returns a more accurate value, and the previous false positive example is not positive anymore. Whereas, when a person is detected, the detection is persistent.

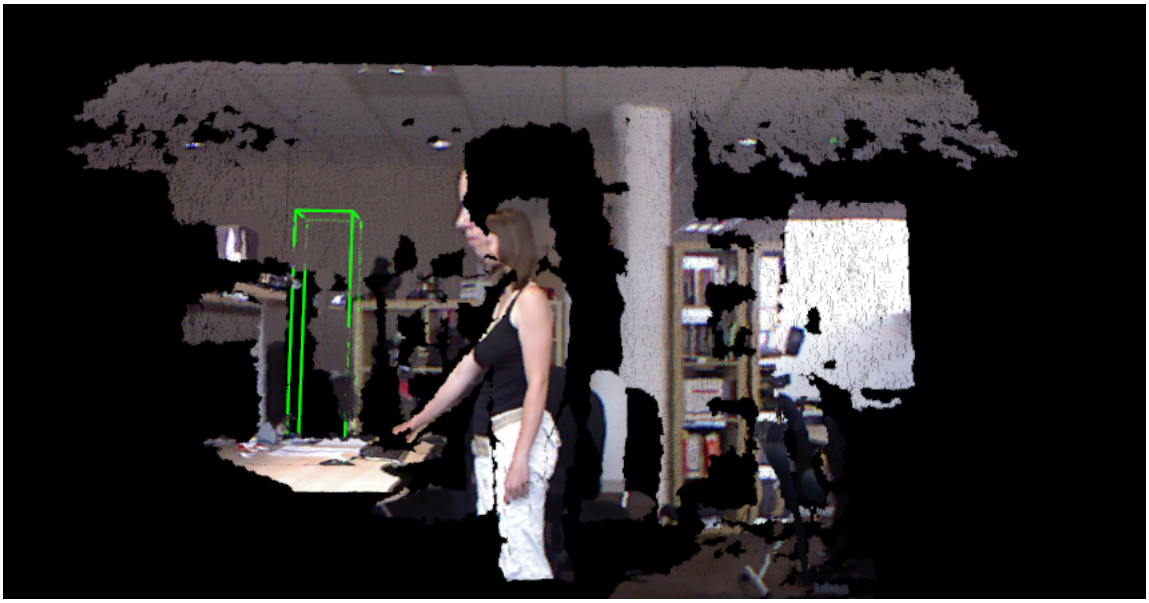


Figure 4.12. Example of a false positive detection.

Tests have been performed to deduce the robustness to partial occlusions and to know if the algorithm can detect people on the side.

The algorithm has been studied and improved to deal with partial occlusions. Thanks to that, it is possible to detect people that are behind an object, as illustrated on the left picture of Figure 4.13. However, as illustrated on the right picture, the risk of non detection of a person is still high when some parts of the person are occluded by an object.

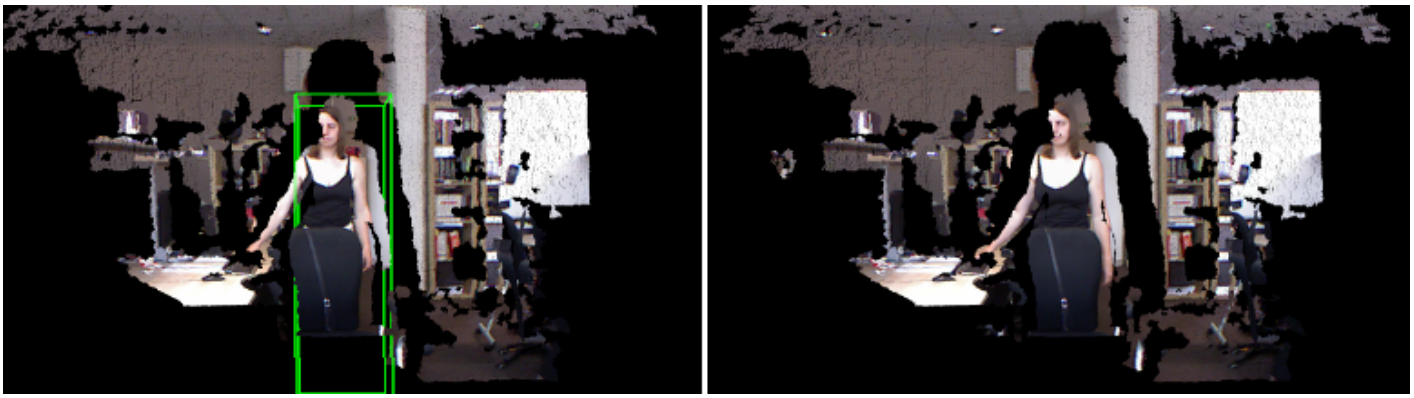


Figure 4.13. Example of robustness to partial occlusions.

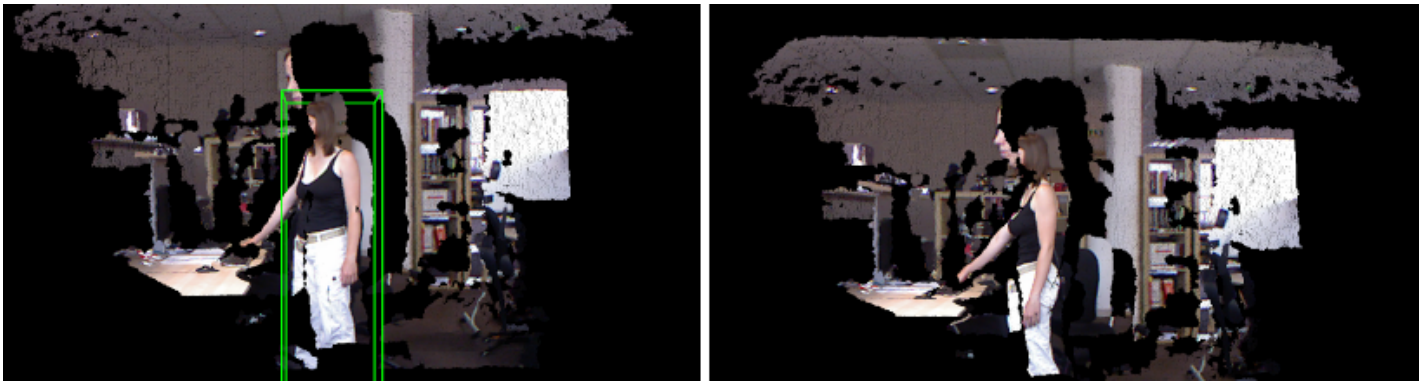


Figure 4.14. Example of robustness with a person turned to the side.

At the end, the robustness to people pose has been tested. It has been seen before (in Figures 4.10 and 4.11) that the detector is able to detect people if the front or the back are seen. Concerning people turned on the side, in Figure 4.14 it can be seen that if the person is not exactly on the side the detector gives a good result. However if someone is perfectly turned on the side the detector can generate false negatives. This behavior has been observed on several frames. A possible reason for this problem could be for example that there were too few examples of persons turned on the side in the training examples of the classifier. An improvement could be to increase the number of training examples.

4.3.2 Comparison with another algorithm

To evaluate the detection performance of the algorithm, similar tests have been done with another detector called OpenNI Tracker. This detector is developed by the NITE company and is not open-source. It is used to deduce the advantages and drawbacks of the detector described in the last section. In this part, results of the OpenNI tracker are detailed and at the end conclusions are drawn about which detector is more adapted to this project.

This detector returns the joints corresponding to the user's skeleton. The joints are drawn as three axes (green, red and blue) in the following images.

Figures 4.15 and 4.16 illustrate the result of the detector with a person seen from the front or from behind.

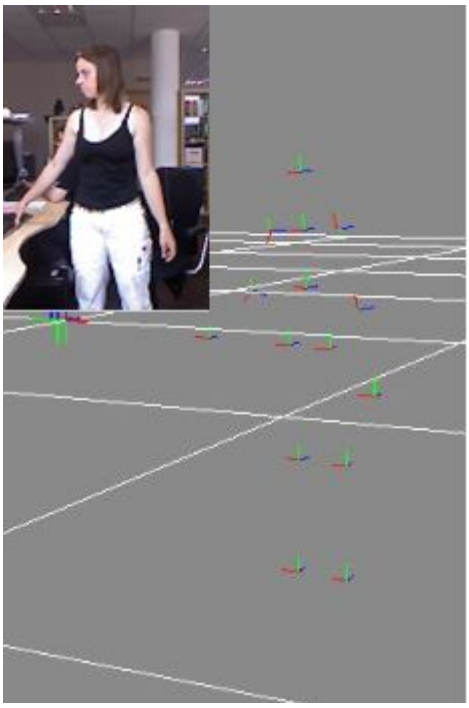


Figure 4.15. Detection of a person.

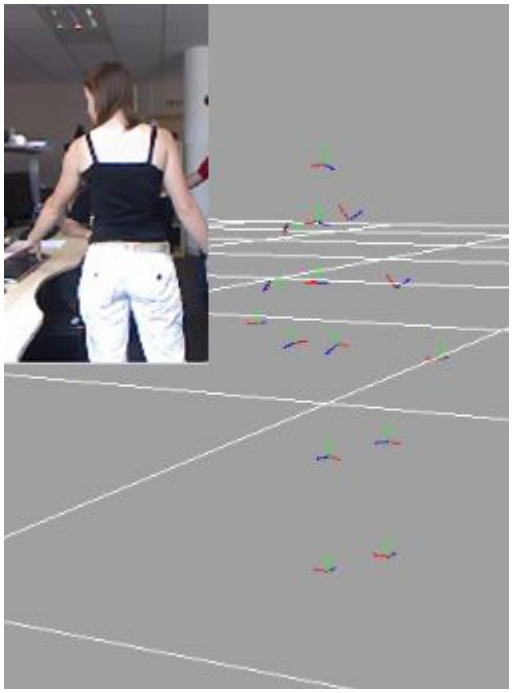


Figure 4.16. Detection of a person seen from behind.

Figure 4.17. Example of robustness to partial occlusions.

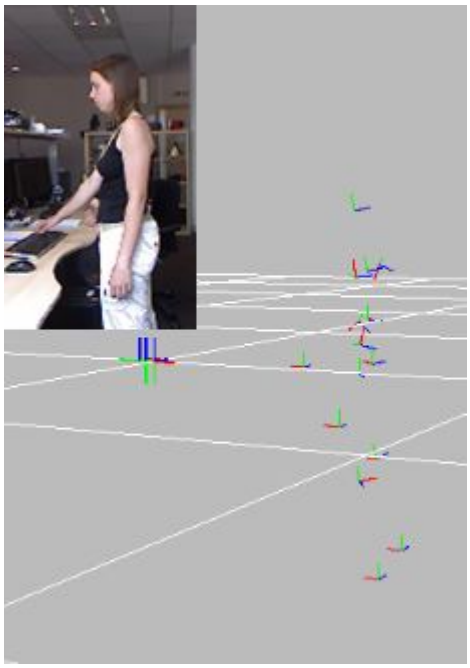


Figure 4.18. Detection of a person turned on the side.

Concerning occlusions, the OpenNI detector is able to detect people even with partial occlusions (Figure 4.17).

Concerning people on the side, this detector is more efficient than the one studied in this project (Figure 4.18).

These tests show that the detector using point cloud data can be improved to become more robust.

However, all the previous tests have been done with a static camera. The main advantage of the detector studied in this project in comparison with the OpenNI detector is that it is able to detect people when the camera is moving and not only with a static camera. As the robot needs to move, the PCL detector is more adapted to this project than the OpenNI detector.

4.3.3 Execution time and CPU

The execution time of the algorithm is calculated with the computer used on the robot (Intel Core i5, 2.6 GHz). The frame-rate of the RGB-D sensor is 30 FPS (Frames Per Second), this value is considered to be the optimal value because it is not useful to have a frame-rate for the detector higher than for the sensor. The people detector algorithm gives a frame-rate between 20 and 25 FPS. This frame-rate is close to the optimal value expected.

The algorithm uses 25 percent of the total CPU power. This value is reasonable if only few applications are launched. However, as the robot needs to navigate (using

both the laser range-finder and the RGB-D sensor), to detect people and to track people at the same time, computation becomes difficult to handle. It is even more difficult if in the future, the robot needs to be able to do other things than following a person at the same time.

Chapter 5

People following

In the previous chapters, two parts of the project have been solved. The first one is to make a robot navigate safely in a map created before. The second one is to detect people in an image using a RGB-D sensor. In this chapter, an algorithm to track people's position in an image is proposed. Once the robot is able to track people in the image, the last part of the chapter explains how to make the mobile base move to follow the user, after converting the position in the image in position in the world.

In Chapter 2, methods of people tracking are described. But most of the existing methods use a people detector to track the person. This means that at each frame, the sensor gives an image, first the people detector is used to detect people. Then the people tracking algorithm is used to identify the detected persons and associate them to the persons in the last frame.

The main disadvantage of this method is the execution cost and CPU consumption of all these algorithms that run at the same time. If other algorithms than people detection (as navigation and other applications) are run at the same time, the robot would need very high CPU performance to be able to handle everything.

The goal of this project is to find an algorithm that makes the robot follow people. To avoid problem with the CPU power, an algorithm that does not need a people detector at each frame was designed.

5.1 Method

In the proposed method, the tracker receives messages from the people detector at the first frame a person is in the field of view of the camera. Then, for the next frames, the people detector is not run anymore. The tracking algorithm is used to know from frame to frame where the person is in the image. In theory, if everything is perfectly done, the tracker would never need the people detector anymore. In reality, the tracker can lose the person or the person can exit the image. In these cases the tracker goes back to the people detector which looks for new people in the image.

In the following section, the overall algorithm is described. Then few sections explain in more details the different steps of the algorithm. And the last section is about all the special cases encountered and how they are solved.

5.1.1 Overall algorithm

The algorithm is compound of two parts. First the initialization step uses the people detector to find people. Then the main loop runs without the people detector and tries to track the person. If the person is lost, the algorithm returns in the initialization step. Figure 5.1 illustrates this.

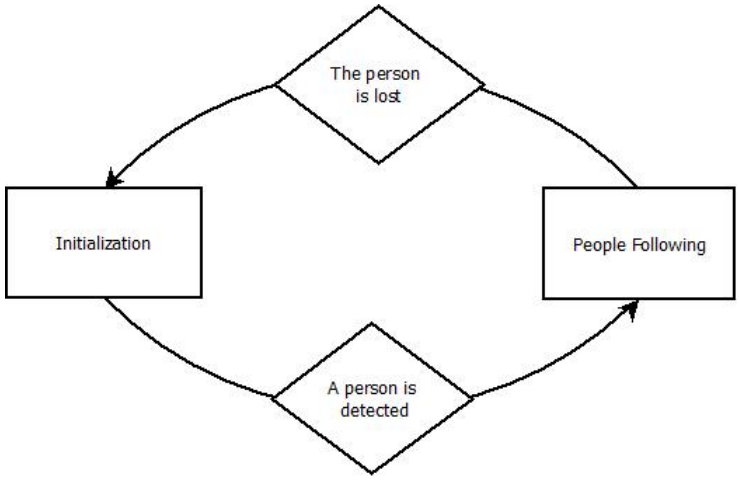


Figure 5.1. Organization of the algorithm.

5.1.2 Initialization

The first part of the algorithm is the initialization step. The people detector finds all the persons in an image. The detector needs to be modified in order to send data about only one person. From the result of the detector, specific data are calculated such as the bounding box which is a rectangle that contains all the pixels of the detected person, but also some depth values of the detected person. Figure 5.2 describes the initialization step.

The filter that selects between multiple persons in the image needs a criterion. Many criteria can be used; for example choose the person which is :

- closer to the center of the image
- closer to the position of the last person the robot was following
- closer to the robot, considering depth value

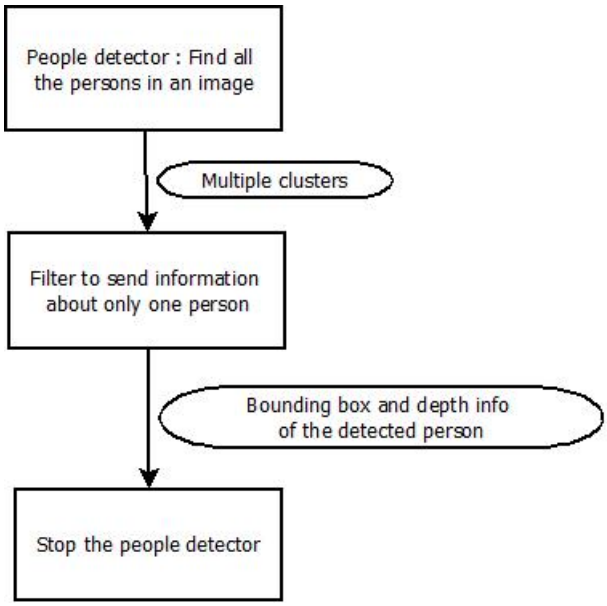


Figure 5.2. Overview of the initialization step.

In this project, the person which is closer to the robot in depth value is chosen. If two persons are at the same distance, the person closer to the center of the image is selected.

In Figure 5.3, two examples of results are illustrated. The detected user is in white; the white rectangle is the bounding box which is the box that contains all the pixels of the user. It can be observed that no other object or people are in the image anymore.

Figure 5.3. Examples of results at the end of the initialization step : the white pixels are the pixels of the user, the bounding box containing the user is drawn.

5.1.3 People following

At the end of the initialization, the people following part receives the data about the detected person : bounding box and depth median, min and max.

- Depth min is the depth of the pixel that is closer to the camera in terms of depth only.
- Depth max is the depth of the pixel that is farthest away.
- Depth median is the middle between depth min and depth max.
- The bounding box is a 2D bounding box, it contains all the points of the user.

The people following part also receives at each frame a new depth image given directly by the camera.

The main loop can be subdivided into four steps (see Figure 5.4) :

- First the depth image is prefiltered, so that each pixel whose depth is not between two predefined values is removed.
- A region growing algorithm is used with the depth image resulting of the prefiltering. With the region growing method, the image is clustered into different regions, and in each region the depth values of pixels are similar.
- Then the most probable region should be selected (if more than one region is returned) knowing the position of the person at the previous frame and the position of the region in the new image. This selection is based on the assumption that the user does not move a lot between two frames. In this project, this assumption seems reasonable as the driver frame-rate is 30 frames per second and the algorithm runs fast. Without this assumption, a motion prediction model would have been necessary to deduce the new position of the user.
- New data about the detected user are calculated : bounding box, depth values (min, max and median)

While the user is not lost, the program loops.

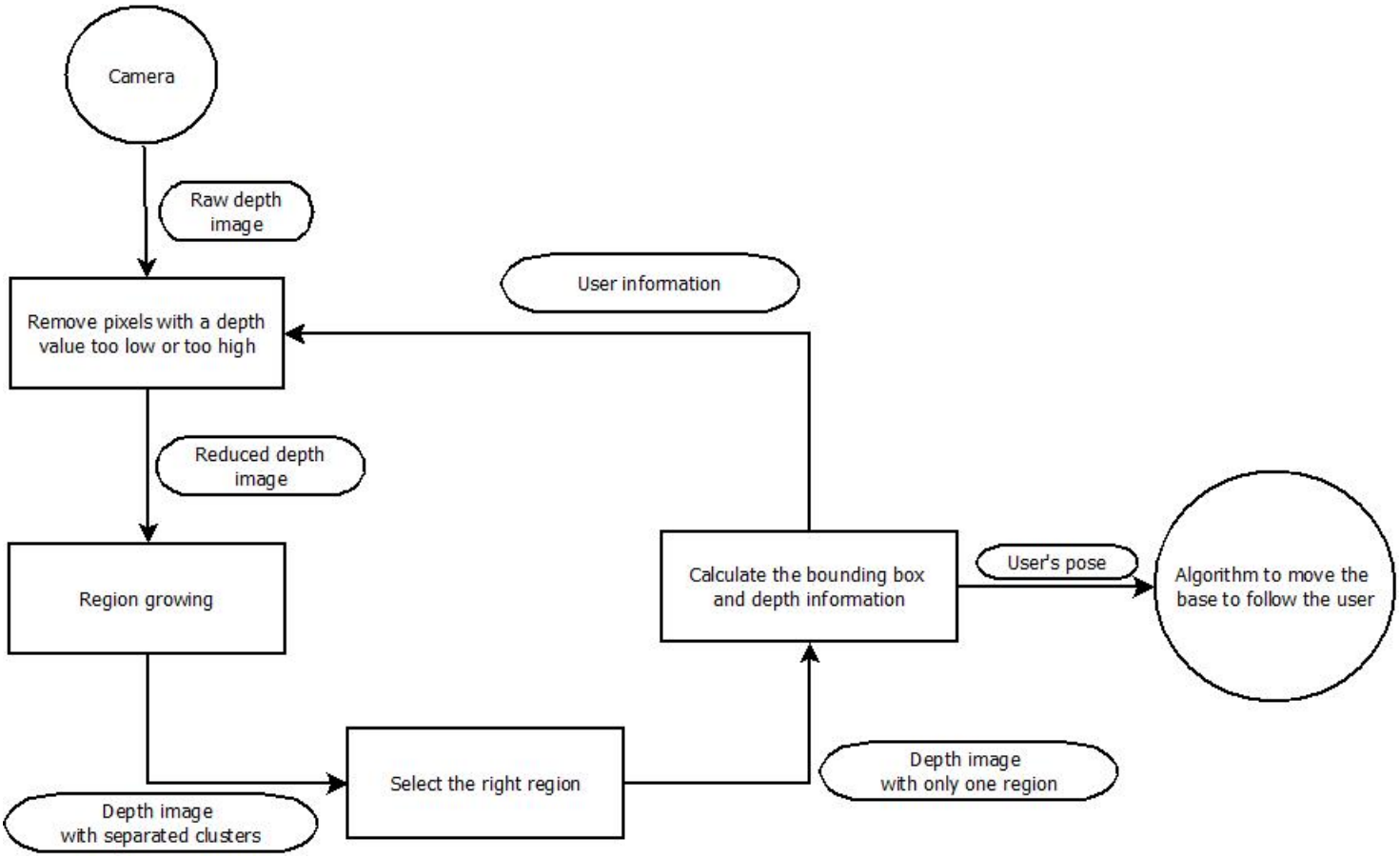


Figure 5.4. Overview of the main loop.

Depth image segmentation

This step assumes that the person the robot is following did not move more than delta millimeters. This assumption is reasonable if the robot is not moving or if the robot and the person are moving approximately at the same speed. So the distance between the depth value of the pixels of the person in the new image and the depth values obtained in the previous frame should not be higher than delta. As information from the previous frame contains the minimal and maximal depths of the user, the depth value of each pixel in the depth image is compared with those values :

$$(depthMin - delta) < depthPixel < (depthMax + delta)$$

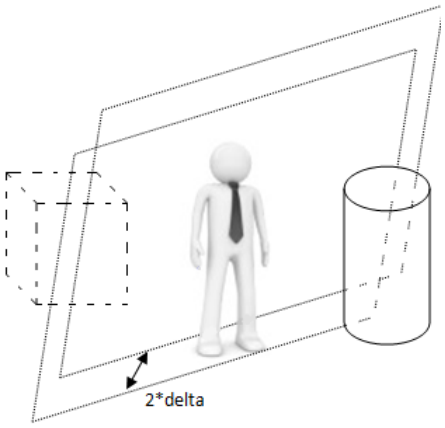


Figure 5.5. Illustration of the depth segmentation : After segmentation, the pixels of the cube are removed and only the person and some of the cylinder's pixels remain.

Region growing

Region growing is an algorithm for image segmentation. The goal is to partition an image into regions. A region is a set of pixels that all share the same criterion.

The region growing algorithm is composed of the following steps :

- Seed initialization. A specific number of seeds is chosen. These seeds are points in the image. Clusters are created, one for each seed. These clusters are the future regions.
- Compare the neighboring points for each region. If the criterion is respected, the neighbor can be classified into the cluster.
- The boundaries of the region are recomputed.
- The two last steps are repeated until all points are classified in a cluster.

This was the theoretical description of the algorithm. In this project, more details and criteria are used for the algorithm design :

1. The region growing algorithm is used to create regions in the depth image (image in which each pixel contains a value corresponding to its depth in camera frame). A region is a set of pixels in which all pixels do not differ from the mean depth value of the region more than a specific threshold.
2. The bounding box of the user in the previous frame is known. The assumption that this user did not move more than some threshold between two frames is used (as the frame-rate is 30 frames per second). With this assumption, it is known that the user in the new frame is not far from the bounding box in the previous frame. Consequently, during initialization, the seeds are placed in the bounding box. However, during region expansion the pixels of all the image are compared with the region and not only those of the bounding box.
3. The neighboring points of a region are calculated using 4-connectivity.
4. To decide if a pixel has to be added to a region or not, the criterion is based on the difference between the depth value of the pixel and the mean depth value of the region. If these values are close, the pixel is classified into the region.
5. When two regions are next to each other, the mean depth values of those regions have to be compared, if the difference is less than a threshold, the region have to be merged.

This algorithm contains several parameters that have to be tuned to get a good result. One of the important parameter is the threshold that is used to decide if two regions must be aggregated or not. Figures 5.6 and 5.7 illustrates the difference between a low threshold and a higher one. In Figure 5.6, a low threshold is used. It can be seen that several regions are not aggregated even if they belong to the user. This is because every part of the user do not have exactly the same depth value. For example, the depth value of the face and the arms can be different depending on the position of the user. In Figure 5.7, a higher threshold is used and every part of the user are merged together.

In Figures 5.6 and 5.7, the depth image used to test the algorithm was a raw depth image, without the depth segmentation step explained in the previous section.

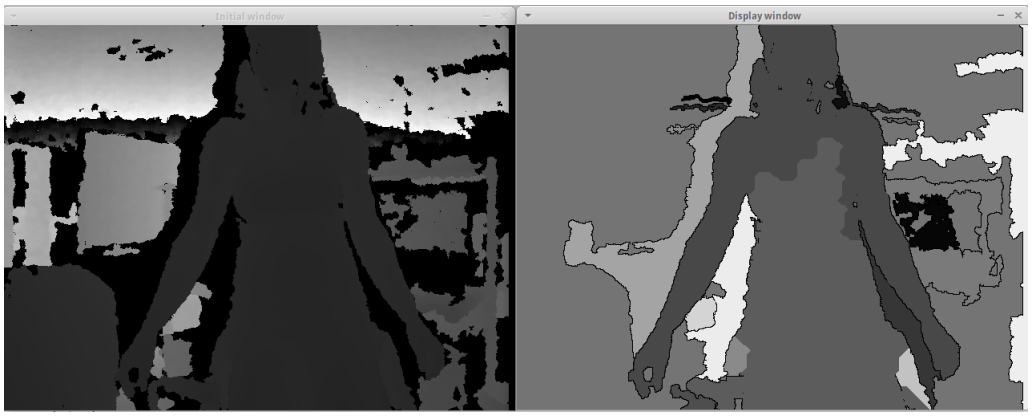


Figure 5.6. Region growing result with a threshold to aggregate regions equal to 50 mm. The left image is the initial depth image, the right one is the result after region growing algorithm.

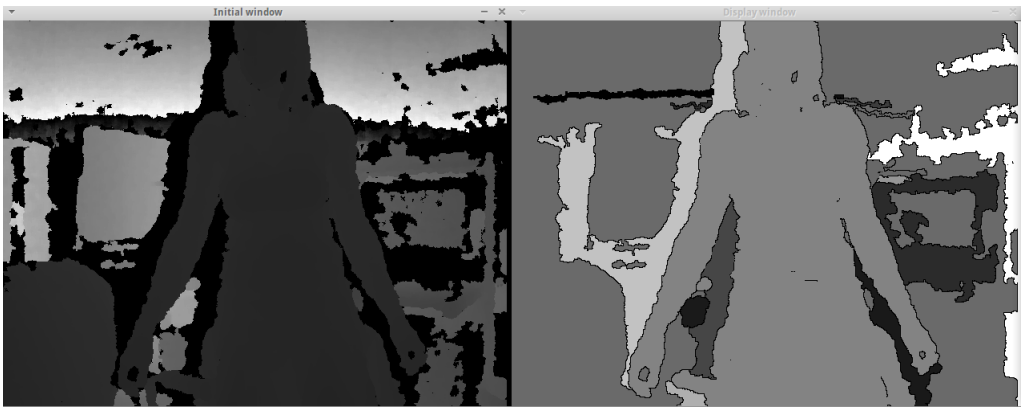


Figure 5.7. Region growing result with a threshold to aggregate regions equal to 100 mm.

Figure 5.8 shows the result of the region growing algorithm on an image that was filtered with the depth segmentation step.

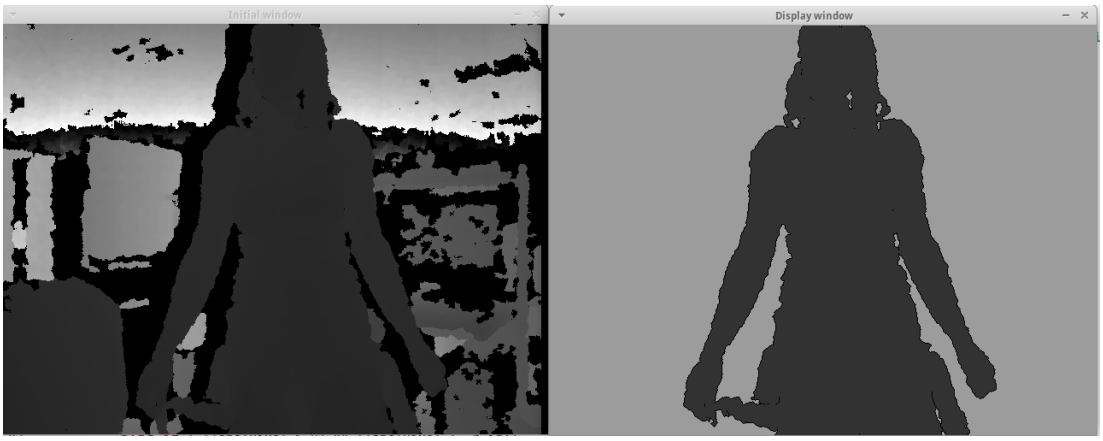


Figure 5.8. The left image is the initial depth image, the right one is the result after both depth segmentation and region growing.

A problem that can appear with this algorithm is to merge the user with an object, as in Figure 5.9. If the user is very close in depth to an object, the object can be classified in the same region than the user. The threshold used to decide if a pixel should be added to a region and the threshold used to aggregate regions have to be decreased to avoid this kind of cases. But if the threshold are too low, as seen in Figure 5.6, some regions or some groups of pixels are not merged with the user.

Figure 5.9. The left image is the initial depth image, the right one is the result after both depth segmentation and region growing, it illustrates the merge of an object with the user.

Region selection

At the end of region growing algorithm, a depth image which contains only pixels of the region selected should be returned. But, when using the algorithm as defined previously, the list of regions at the end of the algorithm can contain more than one region, even after aggregating the regions. The regions in this list are all regions which meet the following criteria :

- All the pixels in the region have a depth between (depthMin - delta) and (depthMax + delta), due to depth segmentation step.
- All the pixels in the region have a similar depth.
- At least one pixel in the region is in the bounding box of the previous frame (the seed that created the region at the beginning).

It is unfortunately possible that more than one region meets these criteria.

It is then necessary to select the right region. One simple criterion is to use the region with the maximum number of pixels. This can work in general cases but when the user is close to an object as a shelf or any other big piece of furniture, there is a high risk that the piece of furniture becomes the selected region.

To improve the criterion, a coefficient is calculated for each region. This coefficient represents the similarity between the position and size of the region and the bounding box of the previous frame. To calculate this coefficient a higher weight is put on the difference in position on X-axis, because the new region should not be far from the bounding box in position. The new region should not have a very different size from the previous frame, except in some special cases where another object is merged with the user. The Y-coordinate is not used in the coefficient because the height of the bounding box is typically very close to the height of the image.

The following formula is used

$$abs(\mathbf{width} - \mathbf{width}_{boundingBox}) + abs(\mathbf{minX} - \mathbf{minX}_{boundingBox}) * 2$$

$$+abs(\mathbf{maxX} - \mathbf{maxX}_{boundingBox}) * 2$$

where the variables are :

- **width** : width of the region, calculated using the pixel on the left boundary and the pixel on the right boundary.
- $\mathbf{width}_{boundingBox}$: width of the bounding box of the previous frame.
- **minX**, **maxX** : minimal and maximal X coordinates of points in the region.
- $\mathbf{minX}_{boundingBox}$, $\mathbf{maxX}_{boundingBox}$: minimal and maximal X coordinates of the bounding box.

The region that has the minimal coefficient is the region the most similar to the previous bounding box. This region is then selected as the region which contains the user.

In Figure 5.10, the region selection is illustrated. The top left image is the result of the region growing step. Four different regions have been returned by the region growing algorithm. The three other images contain the bounding box of the previous frame (in black) and the bounding boxes of three different regions (in colors). The region selection uses the size and position of all these bounding boxes to decide which bounding box contains the region with the user. Here the best region is in the top right image with the red bounding box. The result after selecting the region is shown in Figure 5.11.

Figure 5.10. Illustration of the region selection. The top left image is the result of the region growing algorithm. The three other images contain the bounding box of the previous frame (in black) and the bounding boxes of three different regions (in colors). The most similar bounding box is in the top right image.

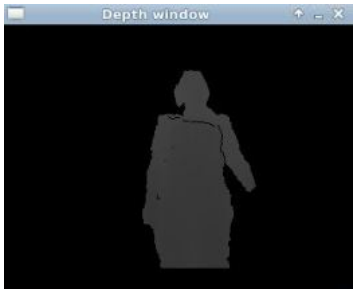


Figure 5.11. Result of the region selection step.

5.1.4 Problem and special cases

After the design of the algorithm and the first implementation, tests are done. Some special cases are identified.

The first problem was the fact that the selected region jumped from the user to an obstacle when the user is close to this obstacle. To solve this problem, the similarity coefficient was implemented to avoid important modifications of the bounding box from one frame to the next one. This was quite a good solution but not perfect as the bounding box width and positions were calculated in number of pixels. Using a number of pixels made this size dependent of the depth of the user. Consequently, when the user was walking away from the robot (the depth value increases) the size of the bounding box was changing a lot and the criterion is not met. To improve that, the size of the bounding box is then calculated using meters. This size is then independent of the depth value of the user.

The second problem appeared when a user was moving away from the robot in a fast way. This case can be easily solved using the delta parameter defined during depth image segmentation. When a user is moving fast and away from the robot, the depth value of the user is jumping from one frame to the other. The delta parameter has to be increased to be sure the user is still in the image after depth segmentation. The problem is that if the delta is too high, a lot of objects may still be in the image after depth segmentation and these objects would possibly be considered as regions at the end of region growing algorithm. A good compromise between how fast a user can move without being lost by the robot and the number of false positives due to a high value of delta have to be chosen.

The last problem encountered is the fact that, as the robot is not very tall, most of the time the robot was seeing the ground. If the user is far from the robot, the ground is then merged with the region corresponding to the user. To solve this problem, the ground was removed from the depth image. Two solutions can be used to remove the ground : calculate the ground plane equation and use it to find the corresponding pixels in the image or remove the pixels in the bottom part of the image. In this project, the second solution is used because it is easier and sufficient to solve the problem. The disadvantage of this solution is that the feet of the user disappear in the final depth image.

5.2 Robot motion

5.2.1 Follow the user

In this section, the last part of the project need to be solved. This part is about how make the robot move to follow the user.

The position of the user in camera frame is given by the tracking algorithm. The robot motion contains two behaviors. The first one is to make the camera look at the user at each frame, which means that the user stays close to the center of the image. This has to be done so that when the user moves he does not go out of the camera field. The second part is to make the robot base move to follow the user.

The first behavior is achieved thanks to two motors (pan and tilt). With these motors, the camera can move in almost all directions. The command sent to the motors are defined using the difference between the position of the user in the image and the center of the image. So that the camera moves in the direction that leads to a smaller difference.

Concerning the motion of the base. The navigation algorithm was studied in order to be used during this step. The position of the user is then sent to the path planner. The path planner calculates a path to go to this position. At the same time, obstacle avoidance is used to avoid collisions. At each frame, a new position of the user is sent and a new path is calculated.

A problem can appear sometimes because the position of the user is occupied space and the navigation algorithm aborts if it cannot reach the goal. This problem did not appear for each goal because of the uncertainty of tracking and mapping, but to avoid it most of the time the x-coordinate of the goal was modified to be equal to the x-coordinate of the user minus 0.2 (in the robot frame). This is done using the assumption that most of the time the space just between the robot and the user is free.

Part III

Implementation and conclusions

Chapter 6

Implementation and summary of results

6.1 Hardware used

To evaluate the algorithms designed, a robot is used. This robot consists of :

- A computer used to run the operating system and the algorithms.
- A mobile base which contains two differential motors, odometry measurement and bumpers (front center, front left and front right).
- A laser range-finder
- A RGB-D sensor and two motors (pan and tilt) to make it move.

6.2 ROS : Robot Operating System

ROS is an operating system for robots. It provides libraries and tools to help developers create robot applications. ROS was developed by Willow Garage company and is completely open source. This OS contains both low level device drivers and high level application blocks such as visualization tools or vision tools.

First, a master is started to manage communication between all other programs, this master is called "roscore". Software parts are organized as packages. Each package is a directory which contains one or more executables called nodes. A node is a program communicating with other nodes via two methods called topics and services.

When using topics, a node sends a message (data structure sent from one node to another) by publishing it to a specific topic. Another node that is interested by this message must subscribe to this topic (see Figure 6.1). This model is a many-to-many and one way transport. Publishers and subscribers are not aware of each other's existence, the master is responsible for message exchange.

Services are used for request and reply model. Services are defined by a pair of message structures, one for the request, the other one for the reply. A node provides the service, another node (the client) sends a request and wait for the answer.

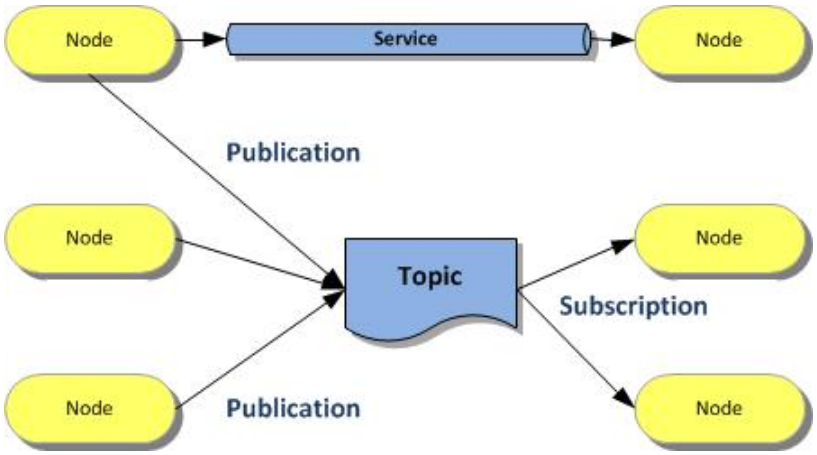


Figure 6.1. Illustration of communication using topics or services.

One of the main advantage of ROS is that all nodes are executed at the same time and can communicate either in a synchronous or asynchronous way. A second important advantage of ROS is all the open source packages provided and the huge community that maintains it. A lot of drivers are provided for example, so that it is not too long to set up a robot and start programming applications on it.

Thanks to the ROS community, all the drivers necessary to use the mobile base, the laser range-finder, the RGB-D sensor and the motors to move the camera are open source. To start using the robot, the only part to implement is the robot's description.

ROS uses URDF files to describe the structure of the robot. This file describes the 3D model of the robot, every positions of the sensors and how each non static parts can move. Some models of well known robots are available but the robot used in this project is a custom one. Consequently a new description file had to be created. This file was done based on the file from the Turtlebot robot which uses the same mobile base. All other sensors were added to the description file. Figure 6.2 illustrates the 3D model of the robot.

After this step, the implementation part is reduced to the main algorithms described in this project : mapping, navigation, people detection and people following.

6.3 Mapping and people detection implementation

6.3.1 Mapping and navigation

The mapping part of this project uses the gmapping algorithm. Gmapping is also an open source software provided by ROS. To implement the mapping part on the robot consists of using the open source implementation and tune parameters (as described in Chapter 3) to adapt the algorithm to the robot. Once the gmapping

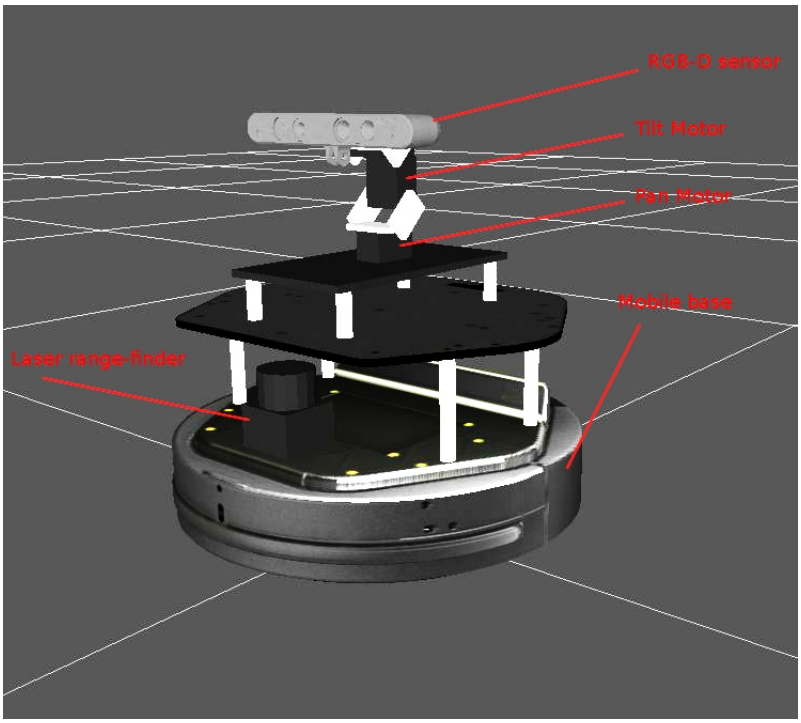


Figure 6.2. 3D model of the robot used for testing the algorithms.

algorithm is adapted to the robot, it can be used to create a map of the environment.

The navigation part uses the navigation package provided by ROS. Exactly as for mapping, the main thing to do is to adapt the parameters to the robot. Then, navigation can be used to make the robot move in the map created before.

For using the path planner to follow a person and not only to navigate in the map, a few parameters have to be modified. In particular, the rotation speed of the robot has to be modified in order to avoid that the user go out of the camera field of view. The user could exit the camera field of view if the mobile base rotates faster than the pan motor of the camera. To avoid that problem, the rotation speed of the mobile base has to be decreased. There is one disadvantage of this modification concerning following. If a person moves fast and perpendicularly to the robot, the mobile base may not rotate enough fast to follow the person.

A second important parameter is the goal error. The path planner uses a goal error to stop the robot at a distance less than this error. When following a person, the goal sent to the planner is the position of the person and the expected behavior of the robot is to stop at a range of 1.5m of the person. The goal error is then increased to 1.5m in order to achieve this behavior. When a person walks, the distance between the robot and the user is generally between two and three meters.

The following part works quite well with this implementation. The robot is able

to follow a person which walks at a low speed. The speed depends on the direction the person is going to. It can be around 1 meter per second if the person walks in the same direction than the robot. But if the person walks perpendicularly to the direction of the robot it should be decreased to around 0.5 meter per second.

6.3.2 People detector integration

The people detector implementation can be found in the Point Cloud Library (PCL). To test it on the robot, the PCL implementation had to be modified to use the messages and topics specific to ROS framework.

First, using ROS, the driver for the RGB-D sensor is different from the driver used by PCL. The driver used in ROS publishes the color image and depth image in topics. The messages published in these topics have to be used in the people detector. The message received has to be transformed (using a bridge between PCL and ROS) from ROS message type to PCL point cloud type.

The robot needs to be autonomous. In PCL's implementation, the user has to click on three ground points so that the ground plane coordinates could be calculated. As explained in Chapter 4, the implementation done with the real robot takes into account the position and parameters of the camera to deduce the ground plane coordinates without any help of the user.

Then, the people detector needs to choose one of the person detected and publish data in a topic so that it can be read by the tracking algorithm. This choice is made using the position of the last person seen.

6.4 People following implementation

The implementation of the people following algorithm is described in Figure 6.3.

- The camera driver takes images from the RGB-D sensor and publishes a color image in the topic /camera/color_image and a depth image in the topic /camera/depth_image. It also publishes a point cloud in the topic /camera/point_cloud.

- The people detector subscribes to the topic containing the point cloud data. Then the people detector uses these data to detect all the people in front of the camera. The detector publishes information concerning those persons in the topic /user_following/init. This message contains a list of all people detected.

- The initialization node subscribes to the topic containing the list of users. Then, it selects the closer person to the robot and it calculates the bounding box, depth minimal, depth maximal and depth median. These values are sent in the topic /user_following/run.

- The main loop part subscribes to this last topic. Then it runs the region growing algorithm to track the person found. The initialization node and people

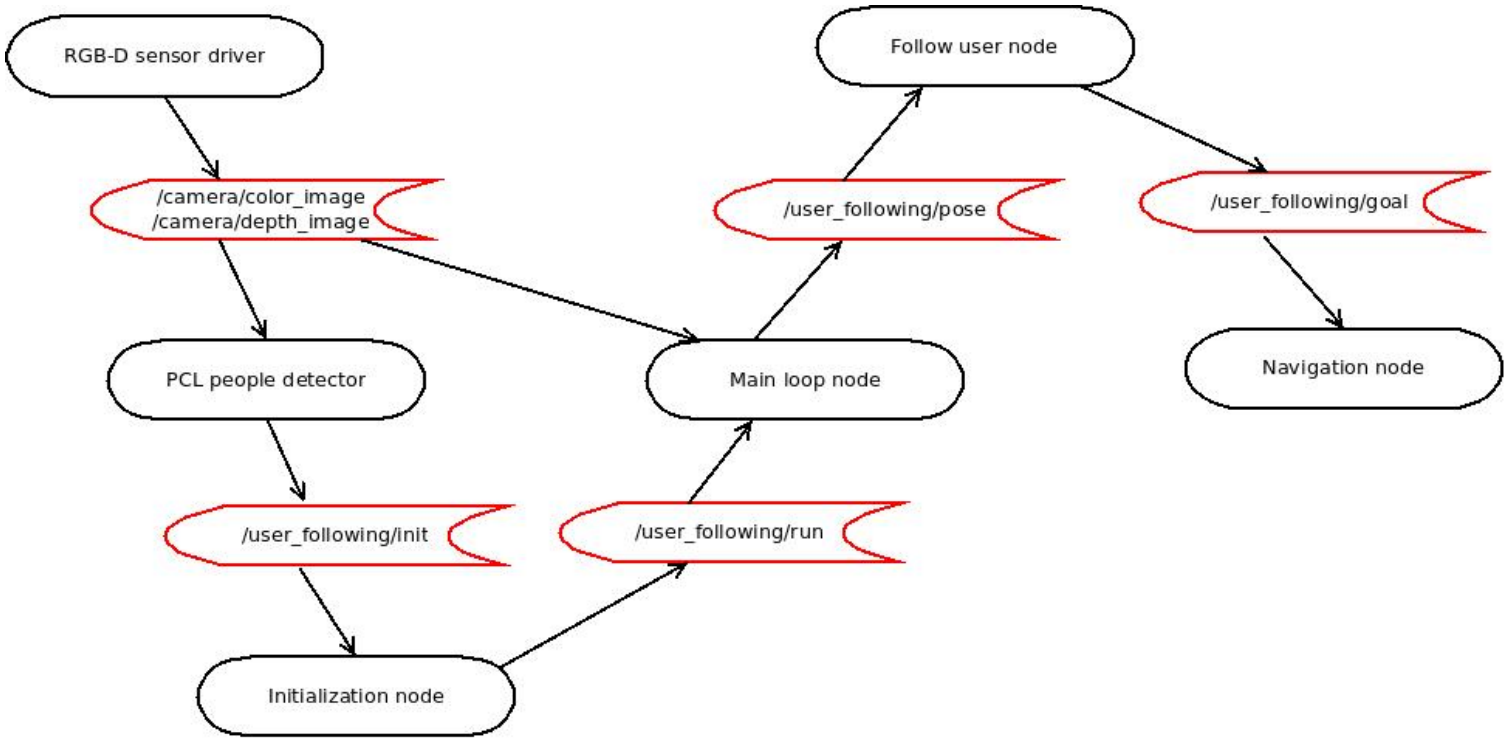


Figure 6.3. Architecture of people following implementation : nodes are represented by black ovals and topics are represented by red forms.

detector are set in a "pause" mode so that they do not use the CPU. If the user is lost by the tracking algorithm, then the people detector and initialization node are started again. The main loop node publishes the position of the user in the topic /user_following/pose.

- At the end, the navigation node uses the information in the topic /user_following/pose to send a goal to the path planner.

6.4.1 Lessons learned

Except for the people following part, most of the work for implementing the algorithms consists of using other algorithms and adapting them to the robot. ROS middleware is a very useful tool especially thanks to the important community which is maintaining all the programs.

However, to adapt the algorithms to the robot is not an easy task because some of the algorithms have many parameters and the documentation is not always very detailed. A lot of testing has to be done to understand most of the effects of the parameters.

To implement the people following part represents a lot of work. Some implementations of region growing are already done but these implementations do not use

the same criteria or do not work with depth image for example. Concerning region growing in this project, everything is implemented from scratch.

One of the most difficult parts of the implementation is the integration of all programs together. Thanks to ROS, nodes can communicate with each others. But this communication needs to be implemented in an efficient way because the project uses real time applications.

6.5 Summary of results

Results of the different parts of the project are described in Chapters 4 and 5. A summary is presented in this section. Additional results are also described.

6.5.1 Results of the people detector

As described in Chapter 4, the main disadvantages of the people detector are :

- The consumption is almost 100% of the CPU.
- A person turned on the side is difficult to detect.
- The algorithm is not totally robust to partial occlusions.

The people detector is used during the initialization step and when the user is lost. Thanks to that, this algorithm can be turned off to avoid CPU consumption during the main loop of tracking. The two other disadvantages can be avoided if the person is in front of the robot and facing it during initialization.

Moreover, the people detector has also some problems to detect a person that is very close because the robot is on the floor and the camera has to look at the user with low-angle shot. This involves that the user can not be closer than 1.5m to the robot.

6.5.2 Results of the overall tracking algorithm

The results of the region growing algorithm are described in Chapter 5. The main problem of region growing algorithm appears when the user is very close to an object. When this case happens, the user is seen but the object is merged with him. The next frame it sometimes happens that only the object is seen by the region growing algorithm. To avoid these problems, a filter was implemented in order to abort detection if the size of the bounding box jumps from a value to a really higher one.

The region growing algorithm has an execution time almost equal to 10 frames per second and the CPU consumption is close to 40% of the CPU. These results are acceptable. Concerning the execution time it could probably be improved to get a frame-rate higher than 20 frames per second.

After all the modifications due to the special cases presented in Chapter 5, some problems still appear :

- Sometimes some false positives appear and the camera looks at objects that are not people. But the number of false positives detected decreased a lot thanks to special cases modifications.
- The path planner is quite slow to calculate the path, sometimes the robot motion is not fluent.
- When too many obstacles are close to the robot, the path calculation is too complicated and the robot aborts, which means the robot will not try to reach the goal and continue to the next frame.
- If the user is too fast, in particular if the user moves on the side (more than 1 meter per second in the direction of the robot, more than 0.5 meter per second on the side), the robot is too slow to follow him.

6.5.3 Possible improvements

The first and most important improvement of this project would be to create some evaluation tools (using for example recorded data) to get some quantitative performance and not only qualitative performance.

Some data containing a human A moving in the office would have to be recorded with a camera held by a human B (or a teleoperated robot) that follows him. These recorded data could be used to know for example how many times the tracking algorithm would lose the person. The motion part is more difficult to test with recorded data because the people detection and tracking depends on motion, and vice versa.

Unfortunately, these tests could not be done in time.

The region growing algorithm was implemented using an iterative method. As the frame-rate is 10 frames per second and the frame-rate of the camera driver is 30 frames per second, a recursive method could be tested to compare the execution time and deduce which method is the best.

Regarding the robot behavior, the most important problem are the false positives. Several methods could be used to detect if the tracker looks at an object or at a human. First, an image mask which contains only the pixels of a human could be compared to the detected object shape in order to deduce if this object is a human or not. A second method would be to compare the position of the detected object to the map to find if an obstacle is known to be here. This last method assumes that the person must not go on a table or any other object. Currently the robot is able to follow a person moving slowly (around one meter per second) in front of him. Another work could be to improve the method to get a more efficient algorithm and so to be able to follow persons moving faster. In addition to that, the behavior of the robot when it has lost a person could be improved. For the moment, the robot stops moving and wait for a person. An more intelligent behavior could be for

example to use the map to guess why the user disappeared, if the user went behind a wall for example the robot can move to look behind this wall.

Moreover, the ROS implementation of the navigation was not really done for real time motion like used in this project. This implementation assumes that the user gives a goal to the robot and wait until this goal is achieved. However, in this project, the goal sent to the path planner is modified almost 10 times per second. The implementation could be improved to handle real time use.

Chapter 7

Summary

This project tries to solve the problem of making an autonomous mobile robot follow a person in an office. The main parts of this study are :

- How to make the robot navigate in its environment without colliding with any obstacle ?
- How to detect a person using a camera ?
- How to know at each frame the position of the person the robot is following ?

The first question contains two parts : create a map of the environment and navigate in this map. The map was created using an improved version of Rao-Blackwellized particle filter. To navigate, the global planner uses Dijkstra's algorithm, the obstacle avoidance algorithm uses the Dynamic Window Approach and the RGB-D sensor is added to the laser range-finder in order to avoid obstacles that are not detected by the laser range-finder. The laser range-finder gives planar information, the RGB-D sensor is used to detect obstacles above the laser range-finder and not high enough for the robot to go below.

The second question is solved using a RGB-D sensor and point cloud data. The people detector uses ground removing, clustering and a HOG detector to detect people in an image. This people detector gives acceptable detection results but the CPU consumption is very high. For this reason, the people detector can not be used at each frame of the following algorithm.

The third problem is the main part of this project. A tracking algorithm was developed using depth segmentation and region growing. The people detector is used only during initialization and then the person detected is tracked using region growing in the depth image. Thanks to that, the CPU consumption decreased significantly.

Once the position of the user is known for each frame, this position is sent to the navigation system that calculates the path to follow the user.

Chapter 8

Conclusions and Future Work

This project describes a way to make an autonomous robot follow a human person under specific constraints as a minimal CPU consumption. Some applications could be a mobile robot in a specific environment as an office, an hospital or in people home.

To use this robot in an environment as people home, future work could be to handle multiple people. For the moment the robot is only able to follow one person chosen randomly. It takes the first person detected. If another person come and cross the person followed there is a risk that the robot modifies its target and follow the other user. Things could be added to choose the person in order to keep all the time the same person.

Concerning mapping and navigation, in this project, the first step is to create a map of the environment using a mapping algorithm. To create the map, the robot moves thanks to keyboard teleoperation. An exploration algorithm could be added to the project so that the robot explores its environment autonomously to create the map. Moreover, for mapping, 3D maps created thanks to the RGB-D sensor could be studied. The advantages of 3D maps is that these maps contain more information about the environment which could be used for many applications, for example identify and put a label on each place of the house.

Bibliography

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping (SLAM) : Part I The Essential Algorithms," 2006.

[2] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, "Introduction to Autonomous Mobile Robots," 2011.

[3] U. Frese and G. Hirzinger, "Simultaneous Localization and Mapping - A Discussion," in *Proc. of the IJCAI Workshop on Reasoning with Uncertainty in Robotics*, (Institut fur Robotik und Mechatronik, Germany), pp. 17 – 26, 2011.

[4] O. Matsebe, M. Namoshe, and N. Tlale, "Basic Extended Kalman Filter - Simultaneous Localization and Mapping," in *Council for Scientific and Industrial Research, Mechatronics and Micro*, (Manufacturing Pretoria, South Africa), 2010.

[5] G. Welch and G. Bishop, "An introduction to the Kalman Filter," (University of North Carolina, Chapel Hil, United States), 2006.

[6] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on Graph-Based SLAM," 2010.

[7] L. Carlone, R. Aragues, and J. A. C. ans Basilio Bona, "A Linear Approximation for Graph-Based Simultaneous Localization and Mapping," in *Proc. of Robotics: Science and Systems VII*, 2011.

[8] G. Grisetti, "Robotics 2 : Graph-Based SLAM using Least Squares," 2006.

[9] G. Grisetti, C. Stachniss, and W. Burgard, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2443–2448, 2005.

[10] "Rao Blackwelized Particles Filter FastSLAM Robot State or pose," (Amirkabir University of Technology).

[11] V. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," in *IEEE Transactions on Systems, Man, and Cybernetics*, 1990.

[12] J. Borenstein and Y. Koren, "The vector field histogram - fast obstacle avoidance for mobile robots," in *IEEE Journal of Robotics and Automation*, 1991.

[13] M. Andriluka, S. Roth, and B. Schiele, "Pictorial Structures Revisited: People Detection and Articulated Pose Estimation," 2009.

[14] L. Pishchulin, A. Jain, C. Wojek, T. Thormählen, and B. Schiele, "Robust People Detection based on Appearance and Shape," 2011.

[15] M. Munaro, F. Basso, and E. Menegatti, "Tracking people within groups with RGB-D data," in *In Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Vilamoura (Portugal)*.

[16] L. Spinello and K. O. Arras, "People Detection in RGB-D Data," 2012.

[17] "Point cloud library." www.pointclouds.org.

[18] "Histogram of oriented gradients lesson." homes.cs.washington.edu.

[19] N. Dalal and B. Triggs, "Histogram of Oriented Gradients for Human Detection,"

[20] "Wikipedia." http://en.wikipedia.org.

Appendix A

Rao-Blacwellized Particle Filter

Algorithm 1 Improved RBPF for Map Learning

Require:
$\mathcal{S}_{t-1}$, the sample set of the previous time step
z_t, the most recent laser scan
u_{t-1}, the most recent odometry measurement
Ensure:
$\mathcal{S}_t$, the new sample set

$\mathcal{S}_t = \{\}$
for all $s_{t-1}^{(i)} \in \mathcal{S}_{t-1}$ **do**
$< x_{t-1}^{(i)}, w_{t-1}^{(i)}, m_{t-1}^{(i)} >= s_{t-1}^{(i)}$

// scan-matching
$x_t'^{(i)} = x_{t-1}^{(i)} \oplus u_{t-1}$
$\hat{x}_t^{(i)} = \text{argmax}_x \, p(x \mid m_{t-1}^{(i)}, z_t, x_t'^{(i)})$

if $\hat{x}_t^{(i)} =$ **failure then**
$x_t^{(i)} \sim p(x_t \mid x_{t-1}^{(i)}, u_{t-1})$
$w_t^{(i)} = w_{t-1}^{(i)} \cdot p(z_t \mid m_{t-1}^{(i)}, x_t^{(i)})$
else
// sample around the mode
for $k = 1, \ldots, K$ **do**
$x_k \sim \{x_j \mid |x_j - \hat{x}^{(i)}| < \Delta\}$
end for

// compute Gaussian proposal
$\mu_t^{(i)} = (0,0,0)^T$
$\eta^{(i)} = 0$
for all $x_j \in \{x_1, \ldots, x_K\}$ **do**
$\mu_t^{(i)} = \mu_t^{(i)} + x_j \cdot p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_t \mid x_{t-1}^{(i)}, u_{t-1})$
$\eta^{(i)} = \eta^{(i)} + p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_t \mid x_{t-1}^{(i)}, u_{t-1})$
end for
$\mu_t^{(i)} = \mu_t^{(i)} / \eta^{(i)}$
$\Sigma_t^{(i)} = \mathbf{0}$
for all $x_j \in \{x_1, \ldots, x_K\}$ **do**
$\Sigma_t^{(i)} = \Sigma_t^{(i)} + (x_j - \mu^{(i)})(x_j - \mu^{(i)})^T \cdot$
$p(z_t \mid m_{t-1}^{(i)}, x_j) \cdot p(x_j \mid x_{t-1}^{(i)}, u_{t-1})$
end for
$\Sigma_t^{(i)} = \Sigma_t^{(i)} / \eta^{(i)}$
// sample new pose
$x_t^{(i)} \sim \mathcal{N}(\mu_t^{(i)}, \Sigma_t^{(i)})$

// update importance weights
$w_t^{(i)} = w_{t-1}^{(i)} \cdot \eta^{(i)}$
end if
// update map
$m_t^{(i)} = \text{integrateScan}(m_{t-1}^{(i)}, x_t^{(i)}, z_t)$
// update sample set
$\mathcal{S}_t = \mathcal{S}_t \cup \{< x_t^{(i)}, w_t^{(i)}, m_t^{(i)} >\}$
end for

$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} (\tilde{w}^{(i)})^2}$
if $N_{\text{eff}} < T$ **then**
$\mathcal{S}_t = \text{resample}(\mathcal{S}_t)$
end if

Figure A.1. Description of RBPF algorithm from [9].

www.kth.se