

Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

**RGB-D people tracking by detection
for a mobile robot**

Candidato:
Filippo Basso

Relatore:
Emanuele Menegatti

Correlatore:
Matteo Munaro

*Alla mia famiglia
che mi ha sempre sostenuto.*

*A tutti quelli che mi sono stati vicini
in questi cinque meravigliosi anni.*

Abstract

In this work, we propose a fast and robust multi-people long-term tracking algorithm for mobile platforms equipped with RGB-D sensors. The approach we followed is based on the clustering of the scene by using 3D information in conjunction with a reliable HOG classifier to identify people among these clusters. For each detected person, we instantiate a Kalman filter to maintain and predict his location, and a classifier trained on-line to recover the track even after full occlusions. We also perform some tests on a challenging real-world indoor environment whose results have been evaluated with the CLEAR MOT metrics. Our algorithm proved to correctly track 96% of people with very limited ID switches and few false positives, with an average frame rate of more than 25 fps. Moreover, its applicability to robot-people following tasks has been tested and discussed.

Contents

1	Introduction	1
1.1	State of the art	1
1.2	Content	2
2	Components	3
2.1	Microsoft Kinect	3
2.2	Pioneer 3-AT	4
3	ROS - Robot Operating System	5
3.1	Computation graph architecture	5
3.2	Important libraries	6
4	System overview	11
4.1	Detection	11
4.2	Tracking	12
4.3	Mobile robot	13
4.4	Implementation	13
5	Detection	15
5.1	Ground estimation/removal	15
5.2	Clustering	16
5.3	Clusters evaluation	17
5.4	Implementation details	18
5.5	Results	21
6	Tracking	27
6.1	Motion model	27
6.2	Online classifier	28
6.3	Data association	32
6.4	Tracks management	32
6.5	Implementation details	33
6.6	Results	35
7	System evaluation	37
7.1	Classification effectiveness	37
7.2	Time performances	43
7.3	Tracking evaluation	43
7.4	Real-world application: people follower	45

8	Conclusions	49
8.1	Results analysis	49
8.2	Future developments	50

List of Figures

2.1	Microsoft Kinect sensor	3
2.2	Pioneer 3-AT	4
3.1	Graphical representation of nodes (ellipses) and topics (rectangles).	6
3.2	Microsoft Kinect RGB data.	7
3.3	PointCloud generated from the depth map.	8
3.4	Kinect RGB data and PointCloud combined.	8
3.5	A simple URDF model.	9
3.6	A model of a complex robot and the locations and orientations of each one of its links in the space.	10
4.1	General tracking-by-detection approach.	11
4.2	Our detection approach.	12
4.3	Our tracking approach.	13
4.4	Complete system overview.	13
4.5	The robot platform (a) and the corresponding URDF model (b).	14
5.1	Detection overview.	15
5.2	A simple overview of how RANSAC algorithm works.	16
5.3	A simple example of the creation of a k-d tree.	17
5.4	Geometrical view of some features.	18
5.5	Ground detection window.	19
5.6	Occlusion evaluation example.	20
5.7	The original point cloud.	22
5.8	The down-sampled point cloud.	22
5.9	The point cloud after the ground removal operation.	23
5.10	The point cloud after the clustering operation.	23
5.11	The detected clusters.	24
5.12	The results of the features evaluation on the selected clusters.	25
6.1	The error model of the Kinect.	29
6.2	Tracking via on-line boosting approach.	31
6.3	RGB output of the <code>tracker</code> node.	36
6.4	3D output of the <code>tracker</code> node.	36
7.1	The mobile platform we used for the experiments.	37
7.2	Situation from which the features are taken.	38
7.3	Best three features for track #1.	39

7.4	Best three features for track #2.	39
7.5	Best three features for track #3.	39
7.6	Prediction values given by the classifier of track #1.	40
7.7	Number of frames each distinct track remains in the scene.	41
7.8	An example of track recovery.	42
7.9	Average time for the main phases of the detection process.	44
7.10	Graph representing the evolution of four indicators as a function of the confidence.	46
7.11	People following test: hall.	47
7.12	People following test: corridor.	48
8.1	An example of a person detected even if only his head was visible.	50

List of Tables

5.1	Default parameters for the detector node.	21
6.1	Default parameters for the tracker node.	35
7.1	Average frame rates for our videos.	43
7.2	Tracking evaluation results for static tests.	45
7.3	Tracking evaluation results for moving tests.	45
7.4	Tracking evaluation results for two methods of selecting clusters.	45
7.5	Tracking evaluation results with/without the classifier.	46
7.6	Evaluation of the tracking performance by varying the confidence threshold. . . .	46

Chapter 1

Introduction

The efficient and robust tracking of people in complex environments is an important task for a variety of applications including video surveillance and human-computer interaction. It is a great challenge to design tracking algorithms that correctly distinguish people in complex scenes such as changes in the illumination, changes of the shape, changes in the viewpoint or partial occlusions of targets.

People detection and tracking are key abilities also for a mobile robot acting in populated environments. Such a robot must be able to differentiate people from other obstacles, predict their future positions and plan its motion in a human-aware fashion, according to its tasks. In this contest, the developed algorithms must pay attention to time constraints. Robots must perceive and react to changes in the environment as fast as possible so that they can interact with it.

Popular sensors in this field are RGB cameras and laser range finders. While both sensing modalities have advantages and drawbacks, their distinction may become obsolete with the availability of affordable and increasingly reliable RGB-D sensors that provide both image and range data.

1.1 State of the art

Many works exist about people detection and tracking by using RGB cameras only ([7, 26, 5]) or 3D sensors only ([18, 24, 25, 6, 19]). However, when dealing with mobile robots, the need for robustness and real time capabilities usually led researchers to tackle these problems by combining appearance and depth information.

In [2], both a PTZ camera and a laser range finder are used in order to combine the observations coming from a face detector and a leg detector, while in [17] the authors propose a probabilistic aggregation scheme for fusing data coming from an omni-directional camera, a laser range finder and a sonar system.

Ess *et al.* [8, 9] describe a tracking-by-detection approach based on a multi-hypothesis framework for tracking multiple people in busy environments from data coming by a synchronized camera pair. The depth estimation provided by the stereo rig allowed them to reach good results in challenging scenarios, but their algorithm reached real time performance only by supposing as given the output of a people detector that actually runs at 30s per image. Stereo cameras continue to be widely used in the robotics community ([1, 22]), but the computations needed for creating the disparity map always impose limitations to the frame rate, thus leaving less room for further algorithms operating in series with the tracking one. With the advent of reliable

and affordable RGB-D sensors a rapid boosting of robots capabilities can be envisioned. For example, the Microsoft Kinect sensor allows to natively capture RGB and depth information at good resolution (640×480 pixels) and frame rate (30 frames per second). Even though the depth estimation becomes very poor over eight meters of distance and this technology cannot be used outdoors because the infrared light projected by the sensor can be confused by the sunlight, it constitutes a very rich source of information that can be simply used on a mobile platform. In [16] a tracking algorithm on RGB-D data is proposed that exploits the multi-cue people detection approach described in [23]. It adopts an on-line detector that learns individual target models and a multi-hypothesis decisional framework. No information is given about the computational time needed by the algorithm and results are reported for some sequences acquired from a static platform equipped with three RGB-D sensors.

1.2 Content

In this work, we propose a multi-people tracking algorithm with RGB-D data specifically thought for mobile platforms. By assuming that people are moving on a groundplane, our method is able to robustly track them with a medium frame rate of 25 frames per second. We tested our approach on sequences of increasing difficulty, reaching very good results even when dealing with complex people trajectories and strong occlusions. The track initialization procedure, that relies on a HOG people detector, allows to minimize the number of false tracks and an online learnt person classifier is used every time a person is lost, in order to recovery tracks after a full occlusion.

The remainder of the thesis is organized as follows. Chapter 2 describes the two main hardware components that form the system: the Kinect sensor and the robot. Chapter 3 is focused on the various libraries used in the development of the code and, in particular, in the framework on which we based our system: ROS. The approach we followed to set up the whole system is introduced in chapter 4, while chapters 5 and 6 describe in details the two main phases namely detection and tracking. In the end, chapter 7 demonstrates the validity of our approach by showing both analytical results and a real-world application of the system.

Chapter 2

Components

2.1 Microsoft Kinect

Microsoft Kinect (Figure 2.1) is an accessory for the Microsoft Xbox 360 video game platform. It is considered a “controller-free gaming and entertainment experience”. In fact, it can interpret 3D scene information using a RGB camera, a depth sensor and a multi-array microphone. The depth sensor consists of an infrared laser projector (which creates a grid of points) combined with a monochrome CMOS sensor, which interprets the infrared light captured creating a depth map. It is also equipped with a motorized pivot that permits the sensor to tilt up to 27° either up or down and has an angular field of view of 57° horizontally and 43° vertically.

Video stream Kinect sensor outputs video at a frame rate of 30 Hz as a combination of two different streams. The first one is provided by the RGB camera which uses 8-bit VGA resolution (640×480 pixels) with a Bayer color filter, the other one, instead, is produced by the monochrome depth sensor with the same resolution and 2,048 levels of sensitivity (11 bits).

Software technology The software technology provided to the Xbox developers enables advanced gesture recognition, facial recognition and voice recognition as well as tracking up to six people simultaneously (declared but probably it is able to track how many people can fit in the field-of-view of the camera), including two active players for motion analysis with a feature extraction of 20 joints per player.



Figure 2.1: Microsoft Kinect sensor



Figure 2.2: Pioneer 3-AT

2.2 Pioneer 3-AT

The Pioneer 3-AT (Figure 2.2) is a highly versatile four wheel drive robotic platform. Pioneer 3-AT is a small four-motor skid-steer robot ideal for all-terrain operation or laboratory experimentation. It comes complete with one battery, emergency stop switch, wheel encoders and a microcontroller with ARCOS firmware, as well as Pioneer SDK advanced mobile robotics software development package. P3-AT's powerful motors and four knobby wheels can reach speeds of 0.8 meters per second and carry a payload of up to 12 kg. The P3-AT uses 100 tick encoders with inertial correction recommended for dead reckoning to compensate for skid-steering.

Optional accessories Pioneer 3-AT offers an embedded computer option, Ethernet-based communications, laser, DGPS, and other autonomous functions. Optional 8 forward and 8 rear sonar sense obstacles from 15 cm to 7 m. Its sensing extends far beyond the ordinary with laser-based navigation options, integrated inertial correction to compensate for slippage, GPS, bumpers, gripper, vision, stereo rangefinders, compass and a rapidly growing suite of other options.

Software technology The core software provides a framework for controlling and receiving data from all MobileRobots platforms, as well as most accessories. Includes open source infrastructures and utilities useful for writing robot control software, support for network sockets, and an extensible framework for client-server network programming.

Chapter 3

ROS - Robot Operating System

ROS (Robot Operating System) is a framework for robot software development, providing operating system-like functionality on top of a heterogeneous computer cluster. ROS was originally developed by the Stanford Artificial Intelligence Laboratory but, as of 2008, development continues primarily at Willow Garage¹ with more than twenty institutions collaborating in a federated development model.

ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly-used functionalities, message-passing between processes, and package management. It is based on a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages. The library is geared towards a Unix-like system (Ubuntu Linux is listed as “supported” while other variants such as Fedora and Mac OS X are considered “experimental”).

ROS has two basic “sides”: the operating system side called `ros` as described above and `ros-pkg`, a suite of user contributed packages that implement functionalities such as simultaneous localization and mapping, planning, perception, simulation etc. ROS is released under the terms of the BSD license, and is open source software. It is free for commercial and research use.

3.1 Computation graph architecture

As already mentioned, ROS is based on a graph architecture. From the computational point of view the graph is the peer-to-peer network of ROS processes that share data. So let’s have a quick overview of these graph concepts.

Nodes A node is substantially a process that performs computation: it is where all the main operations are done, the principal entity in the graph. Nodes communicate with each other by using streaming **topics**, **RPC services** or the **Parameter Server**.

Messages In ROS, a message is intended as a simple data structure comprising typed fields. Standard primitive types are supported, as are arrays of primitive types or previously defined messages.

Topics A topic is a named bus over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption and are intended for unidirectional, streaming communication. There can be multiple publishers and subscribers to a topic.

¹<http://www.willowgarage.com>

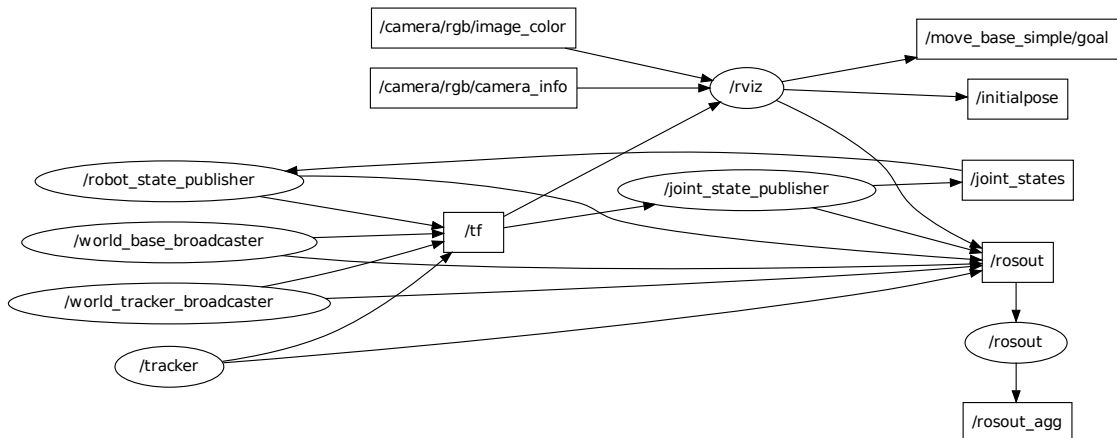


Figure 3.1: Graphical representation of nodes (ellipses) and topics (rectangles).

Services The publish/subscribe model is a very flexible communication paradigm, but it is not appropriate for RPC request/reply interactions. Request/reply is done via a service which is defined by a pair of messages: one for the request and one for the reply.

Master Name service for ROS. It provides name registration and lookup to the rest of the computation graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

Parameter Server It is a shared, multi-variate dictionary that is accessible via network APIs. Nodes can use this server to store and retrieve parameters at runtime.

Bags Bags are a format for saving and playing back ROS message data. They are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

An example of a graph created during a ROS session is visible in figure 3.1.

3.2 Important libraries

3.2.1 PCL

The **Point Cloud Library**² (or PCL) is a large scale, open project for point cloud processing [21]. The PCL framework contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. These algorithms can be used, for example, to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract keypoints and compute descriptors to recognize objects in the world based on their geometric appearance, and create surfaces from point clouds and visualize them. It is well integrated into ROS which also provides some functionalities as ready to use nodes.

PCL is released under the terms of the BSD license and is open source software. It is free for commercial and research use and it is supported by companies such as Google, NVidia and Toyota.

²<http://www.pointclouds.org/>



Figure 3.2: Microsoft Kinect RGB data.

3.2.2 Kinect drivers

The most important source of data, on which the whole tracking approach is based, is the previously described Microsoft Kinect. As we have already said it can output RGB (Figure 3.2) and depth based images at VGA resolution. Besides these two factory image types the sensor produces, ROS drivers give us the possibility to decrease the amount of data published by reducing the resolution of both the images, or rather one of the two, to QVGA (320×240) or QQVGA (160×120). Furthermore the drivers also permit to automatically generate a point cloud (Figure 3.3) starting from the depth map.

Then, since the two images come from two different sensors positioned a couple of centimeters one from the other, there is the necessity of calibrating it. There are no tools to perform this operation yet, but the Kinect has a factory calibration stored on-board that is enough for our necessities (Figure 3.4).

3.2.3 OpenCV

OpenCV³ (Free Open Source Computer Vision) is a library of programming functions mainly aimed at real time computer vision [4]. Example applications of the OpenCV library are Human-Computer Interaction (HCI), object identification, segmentation and recognition, face recognition, gesture recognition, motion tracking, ego motion, motion understanding, structure from motion (SFM), stereo and multi-camera calibration and depth computation, mobile robotics. As for PCL, OpenCV is completely integrated into ROS which also provides image type conversions between OpenCV and ROS formats. It has a BSD license and it is free for commercial or research use.

³<http://opencv.willowgarage.com/>

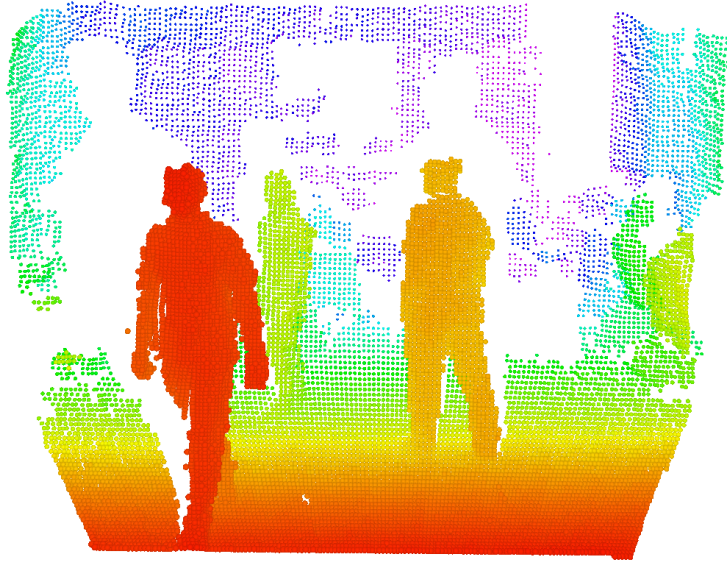


Figure 3.3: PointCloud generated from the depth map.



Figure 3.4: Kinect RGB data and PointCloud combined.

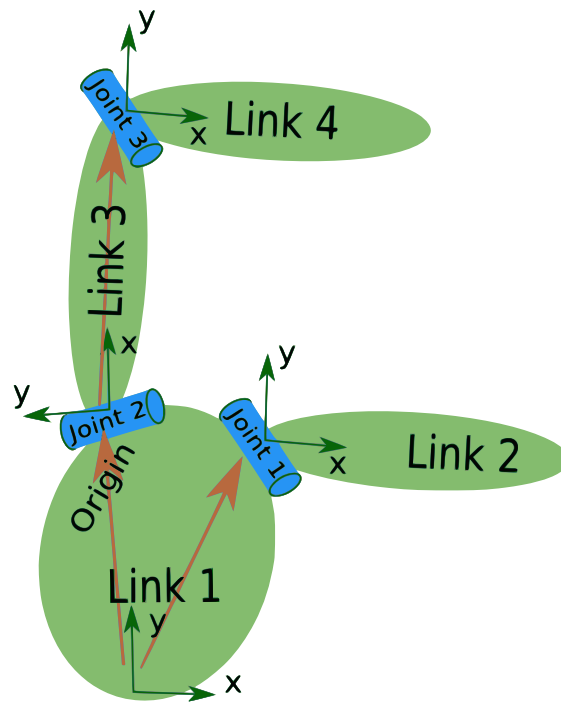


Figure 3.5: A simple URDF model.

3.2.4 URDF/Xacro

The **Unified Robot Description Format** (URDF) is an XML specification to describe a robot. This specification, kept as general as possible, considers only tree structured robots consisting of rigid links (or frames) connected by joints; flexible elements are not supported. A simple example of a tree structure that can be modeled with URDF is reported in Figure 3.5. URDF defines two types of elements: **links** and **joints**. Links are used to specify a rigid body by setting its visual, inertial and collision properties. Joints instead describe the kinematics and dynamics of a joint as well as its safety limits

Along with URDF, ROS provides **Xacro**, an XML macro language very useful when working with large XML documents such as robot descriptions.

3.2.5 TF

A robotic system typically has many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc. The **tf** library is used to keep track of all these frames over time. It maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc. between any two coordinate frames at any desired moment.

The **tf** library can be used in conjunction with URDF specifications to operate with robots in the space. They can be used for example to get the location of a specific frame (link) of the robot with respect to the world. An overview of the joint use of URDF and **tf** is visible in Figure 3.6. It shows a model of a complex robot and the locations and orientations of each link in the space.

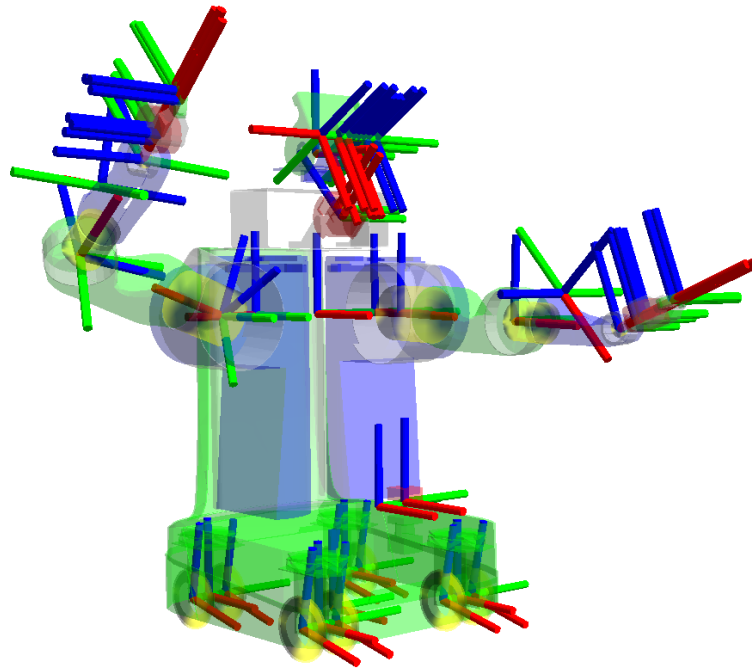


Figure 3.6: A model of a complex robot read from its URDF description and the locations and orientations of each one of its links in the space.

Chapter 4

System overview

The tracking approach we decided to follow in order to accomplish our objectives is the so called **tracking-by-detection**. Such approach involves the continuous application of a detection algorithm in individual frames and the association of detections across frames (Figure 4.1). The main challenges when using an object detector for tracking are that the resulting output is unreliable and sparse, i.e., detectors only deliver a discrete set of responses and usually yield false positives and missing detections.

Our purpose of performing this operation from a mobile robot makes this task even more challenging. We need to face problems like obstacle avoidance or the conversion from the local coordinates system to the world one.

The chapter is organized as follows. Sections 4.1 and 4.2 explain briefly the approaches we followed to create a detector and a tracker by combining existing methodologies with the data generated by our sensors, in particular the Kinect. Then, in section 4.3, we describe how the use of a mobile robot can influence the tracking process and which are the major tasks we planned for the robot.

4.1 Detection

Pedestrian detection in RGB images is an active research area in computer vision and, in recent years, numerous approaches have been proposed. Among them, there are consolidated techniques that recognize people really well and consist in scanning the whole frame and “fire” if the image features inside the local search window meet certain criteria.

One of these techniques make use of a Support Vector Machine (SVM) to create a robust classifier based on **Histograms of Oriented Gradients** (HOG) features [11]. The drawback of this approach is that the performance can be easily affected by background clutter and occlusions and, not less important, it usually can’t run in real time: an exhaustive search need to analyze

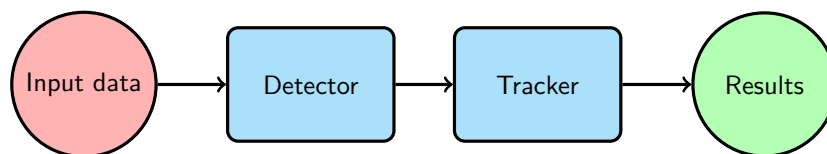


Figure 4.1: General tracking-by-detection approach.

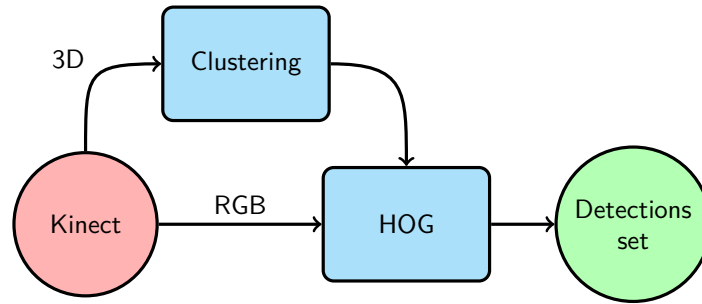


Figure 4.2: Our detection approach.

thousands OF windows to scan the whole image for pedestrians.

Our idea is to use the 3D information given by the Kinect to limit the number of local searches in order to maintain the robustness of the HOG-based classifier but at the same time drastically reduce the time spent to analyze the scene. For this purpose, we need to subdivide the point cloud in clusters and project each cluster back to the image. At this point we can evaluate HOG features on all (or only some of) the resulting windows and decide whether they are people or not. A simple overview of our approach is visible in Figure 4.2.

4.2 Tracking

Once we have a set of human detections for each frame, we need a method to concatenate them over time and, above all, to assign every detection to the correct track. Well known and studied approaches to perform this operation are based on bayesian estimators like **Kalman filters** or particle filters. Both have pros and cons. Usually a particle filter is more precise and correct than a Kalman filter but it is also much more time consuming. On the other hand there exist a lot of different implementations of Kalman filters that, respect to their relative simplicity, perform almost as well as a particle filter but are considerably less computationally expensive [2].

Within our system, the detector tries to determine if each cluster corresponds to a person or not. However, it does not reach perfect classification results. It could happen that some people are not always well classified and the classical approach of using only those windows which are considered pedestrians does not make our system different from the other ones available in the literature. In fact, we use also information coming from the 3D point cloud to know exactly where a cluster is, even when it is not classified as a person. This important step led us not to have misses when people are visible, even partially, and let us use Kalman filters rather than particle filters.

Problems can arise when dealing with full occlusions (e.g. a person can go out of the field of view of the camera and then return back inside). In this case, since our objective is a long term tracking, we need a method to recover back the old track because filtering is not sufficient. Our idea consists in finding a component that learns on-line to distinguish people one from the other, without a previous knowledge of the appearance of the different targets. A rather new solution in this way is the on-line **AdaBoost** classifier introduced by Oza [20] and adapted to the tracking problem by Grabner [12]. An overview of the tracking approach followed is visible in Figure 4.3, while a detailed description of the method is reported in chapter 6.

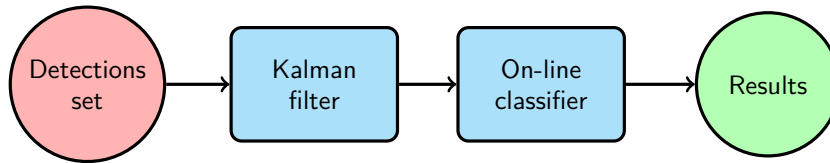


Figure 4.3: Our tracking approach.

4.3 Mobile robot

The challenge of tracking people from a mobile robot forces us to pay attention to at least two things. The first one is that the location of a person or rather an object with respect to the camera cannot be used as input for the Kalman filter, both because movements of people and robot at the same time make the estimation of position and velocity erroneous and because this approach does not permit to use more than a sensor at a time easily. The other thing to pay attention to is that the robot must move in a safe way, without colliding with people or obstacles present in the scene. There are two methods for doing this, one consists in teleoperating the robot, the other instead requires to give the robot a goal and provide it with all the necessary instruments to reach that goal autonomously. At the end the system created should be like the one in Figure 4.4.

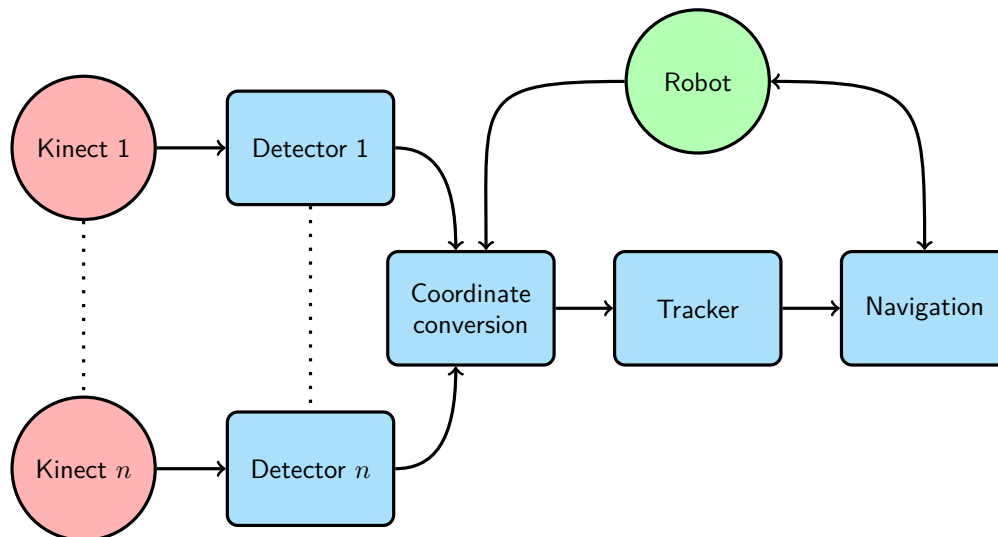


Figure 4.4: Complete system overview.

4.4 Implementation

The whole system has been developed to run exclusively with ROS and to use all of its potentialities as much as possible. Thus, in the remainder of the text, we use the words *program* and *node* indistinctly to refer to a stand-alone executable. When we talk about *publishing* or *subscribing* it means we use the message-passing architecture to exchange information among different nodes.

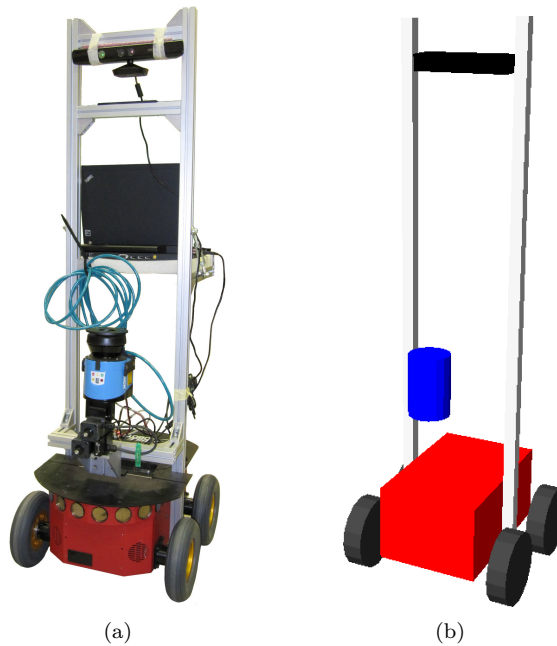


Figure 4.5: The robot platform (a) and the corresponding URDF model (b).

4.4.1 ROS Kinect node

The main source of data is the Kinect. We run it as a simple node that publishes both the RGB image and the point cloud as described in section 3.2.2. While the time needed for the publication of the RGB image is negligible, the generation and publication of a point cloud are quite a computationally expensive task: it takes about one second to process 640×320 points. This time can be decreased significantly by reducing the number of points. For this reason, due to our real-time constraints, we use the RGB image at the maximum resolution while we maintain the depth map at the minimum one.

4.4.2 Robot model

To let the robot move in the environment, we created a model by using URDF and Xacro. As described in chapter 3, the model can be used in conjunction with the tf library and the odometry information published by the robot to place any point of which we know the position with respect to the robot, in the world. Specifically, the points we are interested in are those referred to detected people. For them we know the location in the space with respect to the sensor frame. Indeed, to get their position with respect to the world, we added the Kinect to the robot model by measuring its position empirically. The original robot and the resulting model are visible in Figure 4.5.

Chapter 5

Detection

The detection phase has the objective to divide the scene in clusters and, for each one, calculate some features necessary for the subsequent tracking phase. For the first operation, we need the clusters to be separated one from the other and, by looking at the point clouds generated by the Kinect, we can see that the only means that connects them is the floor. So, before performing any sort of operation, we need to remove all the points that belong to the ground so that we can easily separate the clusters. Obviously, to remove them, we need to know the equation of the plane corresponding to the floor and we can use a deterministic approach or rather a sample consensus method to get it. Only after these simplifications we can take each cluster, calculate its features and send the results to the tracker.

5.1 Ground estimation/removal

As we said, we need the equation of the floor plane to remove the ground from the point cloud. We can calculate it by taking three points $A = (x_1, y_1, z_1)$, $B = (x_2, y_2, z_2)$ and $C = (x_3, y_3, z_3)$ of the floor and solving the following system of equations:

$$\begin{cases} ax_1 + by_1 + cz_1 + d = 0 \\ ax_2 + by_2 + cz_2 + d = 0 \\ ax_3 + by_3 + cz_3 + d = 0 \end{cases}$$

This approach has three small issues. The first is that we need a way to select the three points from the ground, the second is that the points we select are affected by noise and the equation

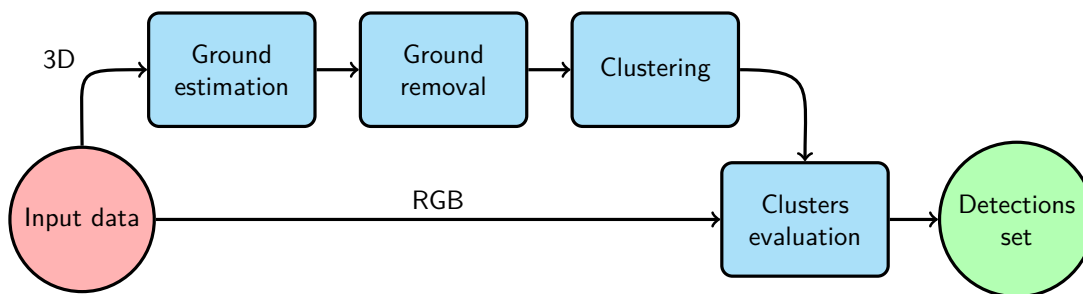


Figure 5.1: Detection overview.

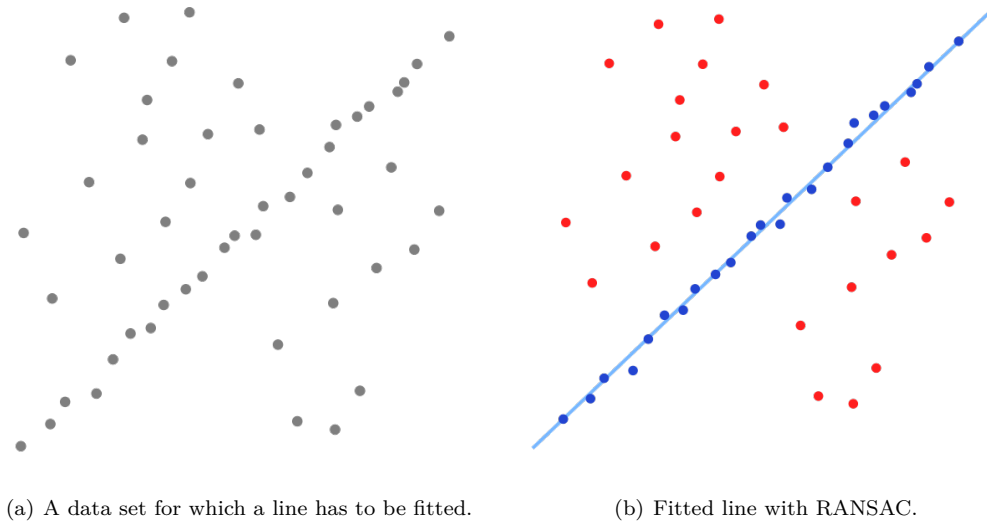


Figure 5.2: A simple overview of how RANSAC algorithm works.

extrapolated from them could be wrong. The last but not least problem is that there can be small changes in the floor slope and also oscillations of the robot during its movements.

The other method to estimate the ground equation is to use a sample consensus approach like **RANSAC** (RANdom SAmple Consensus). RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, as depicted in Figure 5.2. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability. This indeterminism makes it not directly applicable to the ground removal application, in particular because we are not sure it fits the correct plane at each frame.

The solution we adopted to overcome the problems of both methods is to combine them. At the beginning (during the initialization of the system) the user is asked to select three points from the floor and from them we extract the plane equation. Then, at each frame, we use a consensus method to refine it and remove all the points that fit the model.

5.2 Clustering

To create clusters of points starting from the point cloud, it is sufficient to group neighboring points evaluating their euclidean distance. The most efficient way to perform this operation is to put the points in a data structure that is convenient for searches involving a multidimensional search key (e.g. range searches) like a **k-d tree**. A k-d tree is a space-partitioning data structure for organizing points in a k-dimensional space: it is substantially a binary tree in which every node is a k-dimensional point. Every non-leaf node can be thought as implicitly generating a splitting hyperplane that divides the space into two parts, known as subspaces (see Figure 5.3).

Obviously the maximum distance between two points to belong to the same cluster depends on the distance between the points of the point cloud. There are thus two ways to manage this issue. The first is to create an algorithm that performs the clustering operation by considering the distance from the sensor, the second is to perform a down-sampling of the points so that every

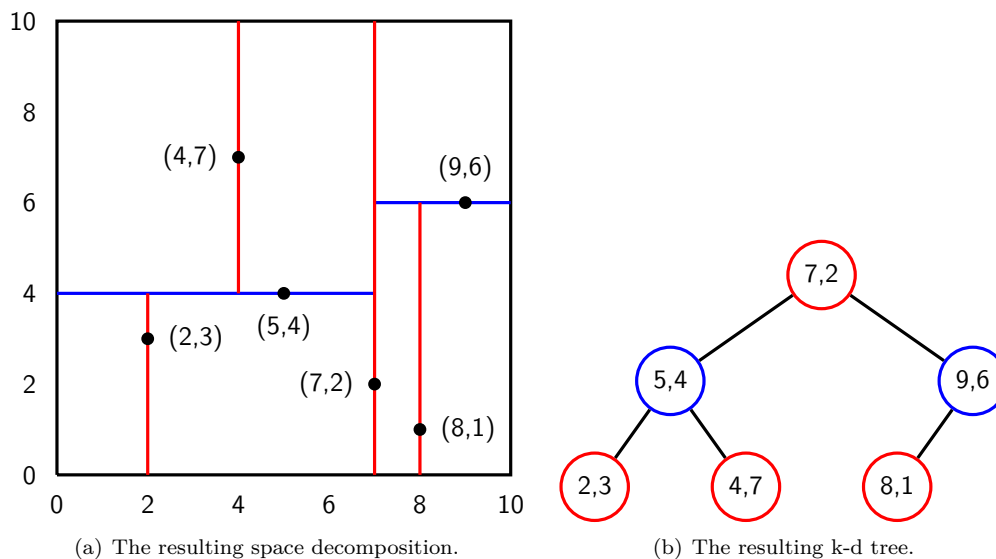


Figure 5.3: A simple example of the creation of a k-d tree from a set of points: $\{(2, 3), (5, 4), (9, 6), (4, 7), (8, 1), (7, 2)\}$.

voxel¹ in the space contains only one point and then use an algorithm distance-independent. This second approach makes the distance between any couple of adjacent points no more than $\sqrt{6} \cdot v$, where v indicates the length of the sides of one voxel. Then it helps in reducing the number of points to take into account for the clustering operation, making it faster.

5.3 Clusters evaluation

The objective of the clusters evaluation is to preserve only the clusters belonging to persons and to extract a series of features necessary for the subsequent tracking phase.

A clear advantage given by the use of the voxel grid filter is that the number of points of a cluster is not dependent on the distance, as it is for the original data generated by the sensor. Therefore, it is possible to immediately eliminate clusters with too many points because they cannot belong to people. Unfortunately, the opposite is not true. Clusters with a relative small number of points can belong to people partially occluded, thus they must be considered. After this initial skimming, we estimate for each cluster the following features (Figure 5.4).

Height It is the theoretical height of the cluster in meters. It is computed as the distance between the highest point of the cluster and the ground plane. Given the coefficients of the plane (a, b, c, d) and the point $H = (x_h, y_h, z_h)$, the height h is calculated as follows:

$$h = \frac{|ax_h + by_h + cz_h + d|}{\sqrt{a^2 + b^2 + c^2}}.$$

Real centroid It is the geometric center of the cluster's shape that is, informally, the arithmetic mean of all the points of a cluster. Supposing to have a cluster with k points the centroid

¹voxel = volumetric pixel.

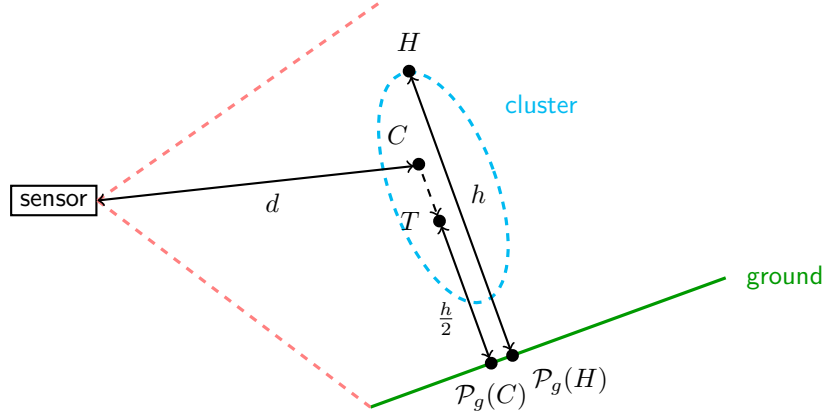


Figure 5.4: Geometrical view of some features.

$C = (x_c, y_c, z_c)$ is calculated as:

$$C = \frac{1}{k} \sum_{i=1}^k P_i.$$

Theoretical centroid It is an estimation of where the real center of a person should be even when partially occluded. It is calculated by projecting the *real centroid* on the ground, taking the vector that goes from the projection to the centroid and scaling it so that its magnitude become half of the height of the cluster.

$$T = \left(\frac{C - \mathcal{P}_{ground}(C)}{|C - \mathcal{P}_{ground}(C)|} + \mathcal{P}_{ground}(C) \right) \cdot \frac{h}{2}$$

Distance It is the euclidean distance d of the *real centroid* from the sensor.

Occlusion It indicates whether a cluster is occluded by any other cluster at minor distance from the sensor or it is partially out of the camera view.

Blob It is the texture corresponding to the cluster. It is obtained by projecting the 3D points back on the RGB image.

HOG confidence It consists in the confidence obtained by applying a HOG people detector [7] to the part of the RGB image corresponding to the cluster theoretical bounding box, that is the bounding box that should contain the whole person, from the head to the ground. This procedure allows to obtain a more reliable HOG confidence (respect to applying the HOG detector directly to the cluster bounding box) also when a person is occluded.

5.4 Implementation details

To implement the detection phase, we have created two different ROS nodes. The first, namely `ground_detector`, is launched at the beginning and deals only with the first ground plane estimation. The second node, the `detector`, performs all the remaining operations frame

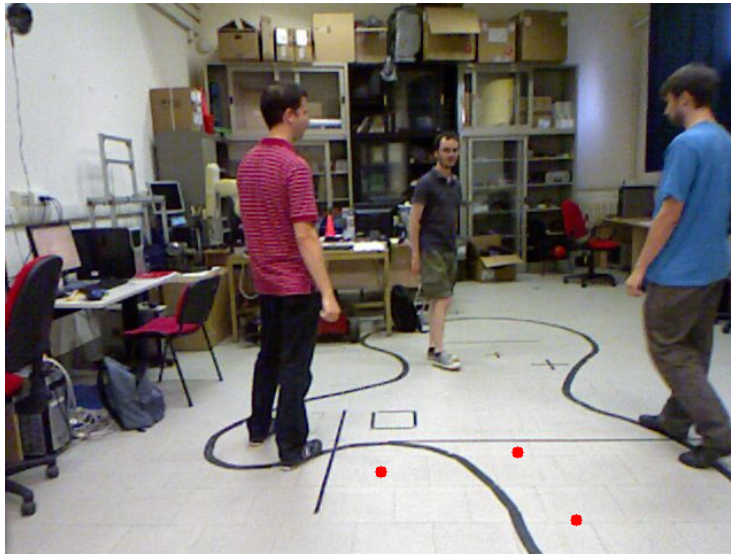


Figure 5.5: Ground detection window. The three points selected are drawn as red circles.

per frame. The down-sampling of the point cloud is instead performed by a node called `voxel_grid_filter`, node available in the PCL library build inside ROS. It also removes `nan` points² and points above a defined threshold along one of three dimensions.

5.4.1 Ground detector

To detect the ground we have to select three points belonging to it. Since it is not possible to directly use the point cloud, the user is presented a window showing the RGB image of the scene from which to select the three points (Figure 5.5) by clicking with the mouse. Then the corresponding points in the point cloud are used to calculate the coefficients of the equation, as described in section 5.1.

5.4.2 Detector

The `detector` starts by waiting for the coefficients of the floor equation to be available on the specific topic. Only then it can perform all the other operations.

Ground estimation/removal and clustering

The three phases dealing with the manipulation of the point cloud, that is ground estimation, ground removal and clustering, heavily use the PCL library. For the ground estimation and removal, we create a `SampleConsensusModelPlane` object that permits both to remove all the points within a defined distance from the plane (in our case the distance is equal to the size of the voxel) and to refine the equation using the same points. For the clustering phase, instead, we use the `EuclideanClusterExtraction` class. It divides the point cloud in clusters according to the distance tolerance given ($2 \times$ the size of the voxel) and remove all those clusters whose size (number of points) is outside the defined range (parameters `min_points` and `max_points` in Table 5.1).

²`nan` points are those points for which the position cannot be calculated due to error in the measurements.

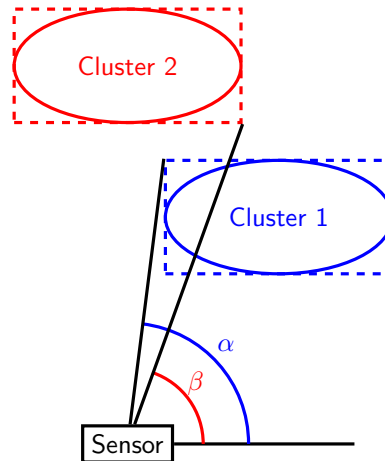


Figure 5.6: Occlusion evaluation example. Cluster 2 is occluded by Cluster 1 because $\alpha > \beta$.

Clusters evaluation

The evaluation phase starts by ordering the clusters by distance. Once ordered, they are filtered by height, that is, since we are dealing with people, we keep only clusters within a predefined height range, and by distance (parameters `min_height`, `max_height` and `max_distance` in Table 5.1). Clusters that are too far from the sensor are indeed seriously corrupted and could lead to false detections. Those clusters that do not meet these constraints are discarded. Then we calculate the real centroid but, instead of using the whole cluster, we consider only the head. We chose this approach because during occlusions the centroid calculated on the whole cluster tends to affect negatively the Kalman estimation, while the one based only on the head remains more stable. It is anyway possible to decide which one of the two solutions to adopt by setting the appropriate flag (parameter `head_centroid` in Table 5.1). Regarding the estimation of occlusions, the operation is performed in an approximate way. We consider only the angles formed by the bounding box of the clusters with respect to the sensor center as depicted in Figure 5.6. This choice has been driven by the difficulty to select the correct external points to calculate the right angles.

Once the features based only on the cluster have been extracted, we calculate those dealing with the RGB image. The first one is the blob. This operation needs to access to the original point cloud to have fairly good results (it depends on the calibration between RGB and depth sensors). The points of the cloud need indeed to be projected back to the image but those obtainable from the down-sampled cluster are too far one from the other and there is no efficient way to construct a blob starting from them.

For what concerns the people detector, we use Dollár's implementation of HOG³ and the procedure and parameters described by Dalal and Triggs [7] for creating and training a SVM classifier. Moreover, to meet the time constraints, the HOG feature is computed only on detections visible from the top to the bottom, even partially occluded. The other clusters are assigned a default confidence value of -100.0 .

At the end of the process, when all the clusters have been evaluated, we create a message (Listing 5.1 and 5.2) containing all the detections with their features and publish it on the output topic. As described in section 4.2 we publish all the clusters detected (that meet height and distance

³Contained in his Matlab toolbox <http://vision.ucsd.edu/~pdollar/toolbox>.

```
Header header
pcl_tracking/Detection3D[] detections
sensor_msgs/Image image
float64[] intrinsic_matrix
```

Listing 5.1: Detection3DArray message.

```
pcl_tracking/BoundingBox3D box_3D
pcl_tracking/BoundingBox2D box_2D

geometry_msgs/Vector3 centroid
geometry_msgs/Vector3 bottom
geometry_msgs/Vector3 top

float64 height
float64 confidence
float64 distance
bool occluded
sensor_msgs/Image blob_image
```

Listing 5.2: Detection3D message.

constraints) but this behaviour can be altered by setting the `publish_all` parameter (Table 5.1) to `false` and deciding the minimal HOG confidence for the detections (`min_confidence` parameter in Table 5.1).

5.5 Results

The results obtained by launching the `detector` node are depicted in the following figures. Figures 5.7 and 5.8 show respectively the original point cloud and the down-sampled one. From it we remove the floor (Figure 5.9) and then subdivide the scene in clusters (Figure 5.10). The RGB image (Figure 5.11) contains only the clusters which pass the height test and are not too far from the sensor. At the end, Figure 5.12 displays the results of the features evaluation. By looking at the confidence values obtained, we can see how they vary depending on the posture of the person classified.

Parameter name	Default value
<code>~people/min_height</code>	1.4
<code>~people/max_height</code>	2.3
<code>~people/min_confidence</code>	-1.5
<code>~people/publish_all</code>	true
<code>~people/head_centroid</code>	true
<code>~people/max_distance</code>	6.5
<code>~cluster/min_points</code>	30
<code>~cluster/max_points</code>	600

Table 5.1: Default parameters for the `detector` node considering the voxel size equal to 0.06 m.

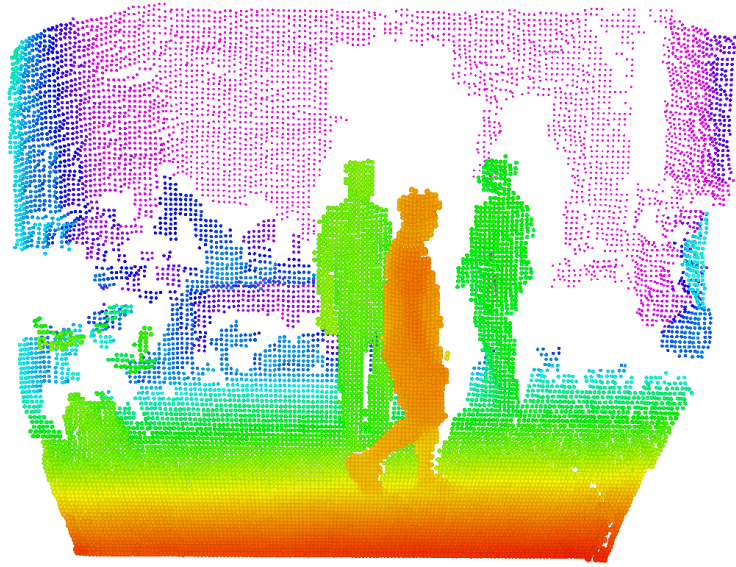


Figure 5.7: The original point cloud.

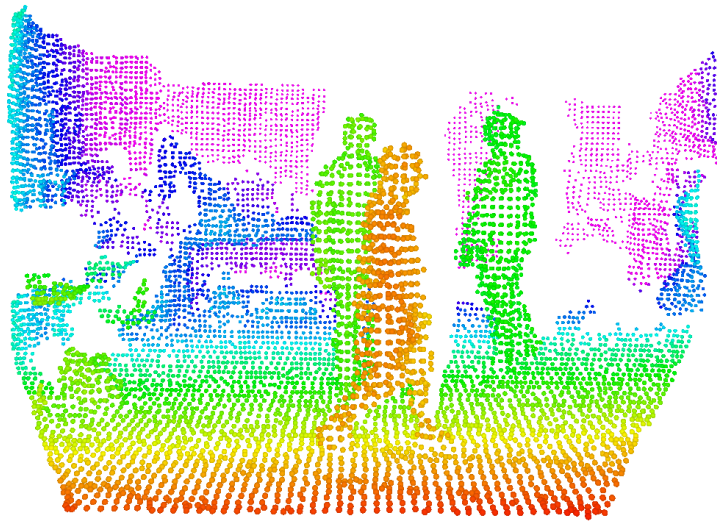


Figure 5.8: The down-sampled point cloud.

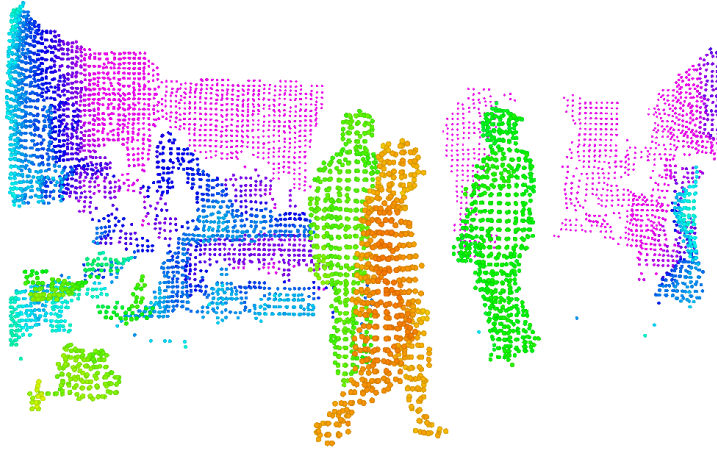


Figure 5.9: The point cloud after the ground removal operation.

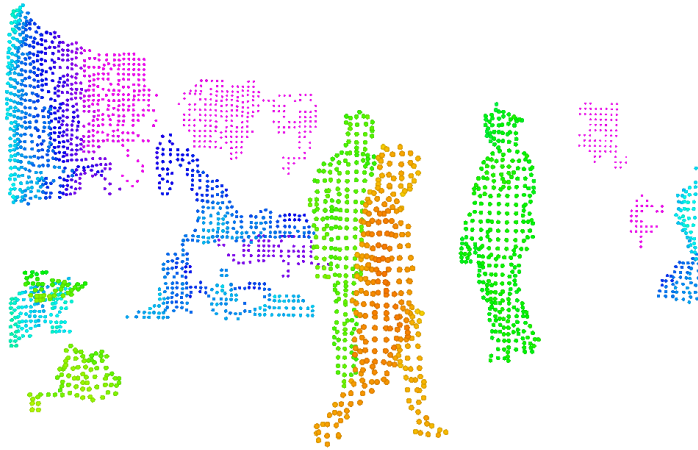


Figure 5.10: The point cloud after the clustering operation.

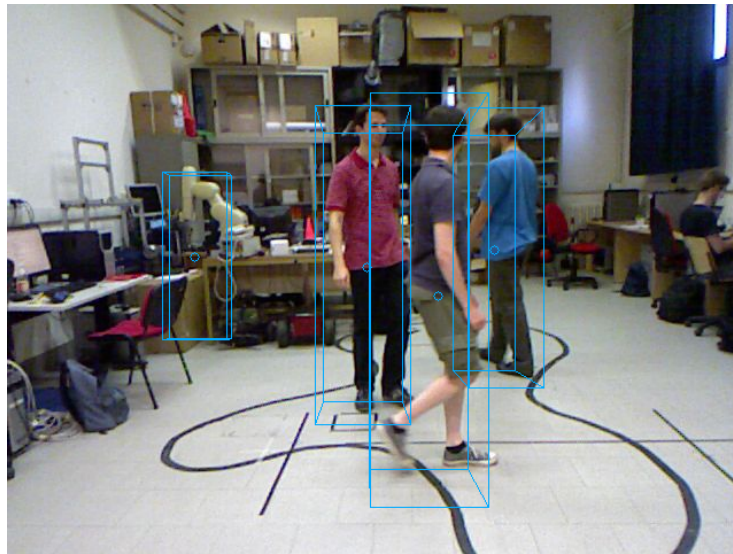


Figure 5.11: The detected clusters (those which pass both the height and the distance filters) viewed on the RGB image.





<pre>centroid: x: 0.347343655584 y: 0.0747241796717 z: 2.75198072494 height: 1.67026865482 confidence: -1.87891810231 distance: 2.67420935631 occluded: False</pre>	
<pre>centroid: x: 0.0224864542609 y: -0.0739659891284 z: 3.57691193076 height: 1.78267812729 confidence: 0.535209623189 distance: 3.60510587692 occluded: True</pre>	
<pre>centroid: x: 0.924414800343 y: -0.215643779412 z: 4.22668651489 height: 1.80970287323 confidence: -0.399896511625 distance: 4.22172641754 occluded: True</pre>	
<pre>centroid: x: -1.70397576028 y: -0.24008244779 z: 6.04538833363 height: 1.63850009441 confidence: -100.0 distance: 6.169485569 occluded: True</pre>	

Figure 5.12: The results of the features evaluation on the selected clusters.

Chapter 6

Tracking

The tracking process is designed following a tracking-by-detection approach: it takes the output given by the detection phase and tries to assign each person detected to the correct track relying on a data association method. The way this data association is performed is the core of the whole process and, as described in chapter 4 it is divided in two different steps. The first step consists in associating every detection to the track that is predicted to be the closest possible in the space. This step is usually enough when tracking one object alone in a scene without obstacles but become insufficient when dealing with multiple targets in a crowded environment. For this reason, we introduce a second step based on a classifier to associate detections that cannot be associated using their location in space to previously “lost” tracks.

The chapter is divided as follows. Section 6.1 delineates how people are intended to move to perform the first association based on the motion. Section 6.2 explains how an on-line classifier is realized and the way we use it for our aims. Section 6.3 is focused on the process of association based on the results given by the Kalman filters and the classifiers, while section 6.4 gives an overview of the policies of creation, update and removal of the tracks.

6.1 Motion model

There are a lot of ways to model human motion in tracking. However, a good way to manage full occlusions is to consider a constant velocity model, as described in [2]. In order to use the Kalman filter, we need to explicit the model in accordance with its framework. Starting from the internal state, we need to maintain the positions and velocities along the two axes x and y , that is

$$\mathbf{x}_k = (x \quad y \quad \dot{x} \quad \dot{y})^T. \quad (6.1)$$

At each frame we measure for each detection its (noisy) location in the space and use it to update the filter, so our observation vector is

$$\mathbf{z}_k = (x \quad y)^T. \quad (6.2)$$

Following this model, we assume that between two subsequent detections, the person undergoes a constant, normally distributed, acceleration with mean 0 and variance σ_a^2 in both directions. From the equations of motion we derive

$$\mathbf{x}_k = \mathbf{F} \cdot \mathbf{x}_{k-1} + \mathbf{G} \cdot \mathbf{a}_k \quad (6.3)$$

where

$$\mathbf{F} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{G} = \begin{pmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix} \quad \text{and} \quad \mathbf{a}_k = \begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix}. \quad (6.4)$$

From it we can hence construct the dynamic system model of the filter

$$\begin{cases} \mathbf{x}_k = \mathbf{F} \cdot \mathbf{x}_{k-1} + \mathbf{w}_k \\ \mathbf{z}_k = \mathbf{H} \cdot \mathbf{x}_k + \mathbf{v}_k \end{cases}. \quad (6.5)$$

The process noise \mathbf{w}_k is a zero mean multivariate normal distribution with covariance \mathbf{Q} ($\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$), where

$$\mathbf{Q} = \mathbf{G}^T \mathbf{G} \cdot \Sigma_{\mathbf{a}} = \begin{pmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix} \cdot \begin{pmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix}^T \cdot \begin{pmatrix} \sigma_{ax}^2 \\ \sigma_{ay}^2 \end{pmatrix} \quad (6.6)$$

The observation model is then the following.

$$\begin{cases} \mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k) \\ \mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \\ \mathbf{R}_k = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot (\sigma_v^2 + \sigma_d^2) \end{cases} \quad (6.7)$$

Regarding the covariance matrix of the noise, we express it as a sum of two different components, σ_v^2 and σ_d^2 .

The first term, σ_v^2 , models the quantization error introduced by the voxel filter and can be simply derived by considering the down-sampling error as uniformly distributed. Let this assumption be

$$\sigma_v^2 = \frac{\text{voxel_size}^2}{12}. \quad (6.8)$$

The second term, σ_d^2 , models instead the measurement error introduced by the depth sensor. For our particular sensor, this error is proportional to the distance squared¹ as depicted in Figure 6.1.

6.2 Online classifier

To overcome the limits imposed by the constant velocity model when dealing with full occlusions, we maintain for each track an on-line classifier based on **Adaboost** [11, 20], like the one introduced by Grabner in his PhD thesis [12].

Briefly, boosting is a widely used technique to improve the accuracy of learning algorithms. Given training samples \mathbf{x} with labels \mathbf{y} , a **strong classifier** $H(\mathbf{x})$ is computed as linear combination of a set of weighted hypotheses called **weak classifiers** $h(\mathbf{x})$ that usually perform just better than random classifiers. The main feature of the on-line version is that it permits to have an usable strong classifier after each update.

¹http://www.ros.org/wiki/openni_kinect/kinect_accuracy.

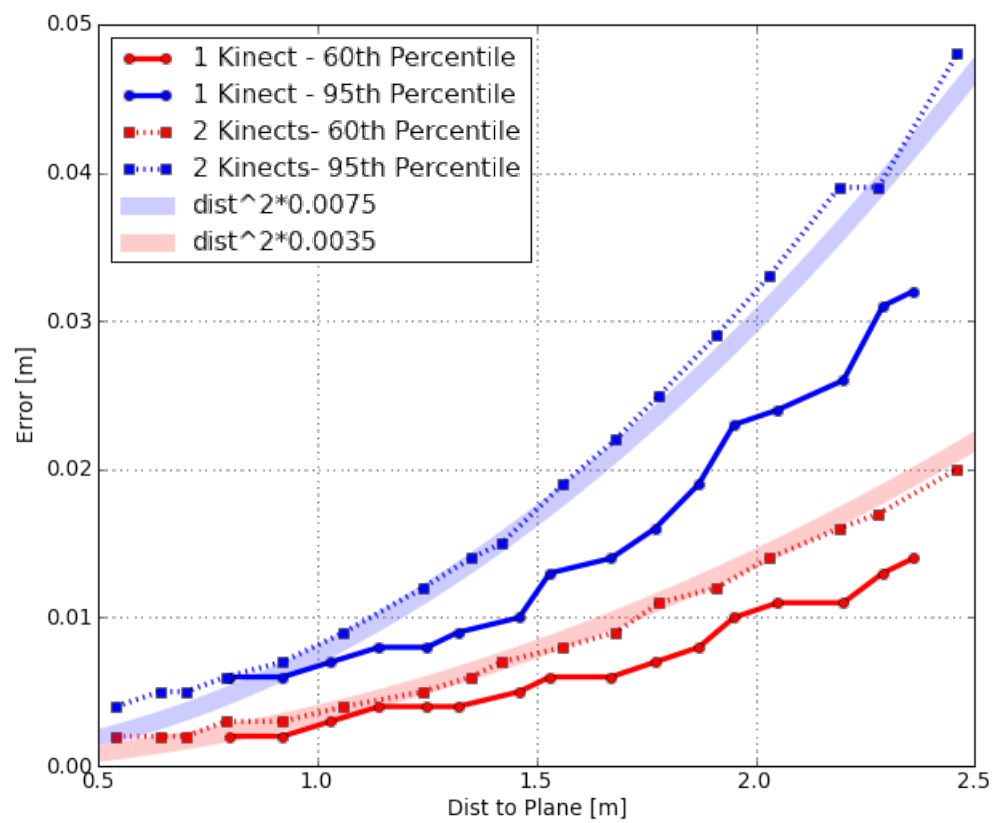


Figure 6.1: The error model of the Kinect. The graph shows it is proportional to the distance squared.

6.2.1 Original work

In [12], Grabner presented a supervised and on-line trained two-class classification algorithm that selects proper features in order to discriminate between these two classes. His main innovation with respect to previous works is the introduction of the so called **selectors** that select the best weak classifiers to form the strong classifier. The three main properties of his approach, that make it applicable to a large variety of computer vision applications, are indeed:

Generality Many computer vision problems can be formulated as binary classification problems, such as detectors, tracking, recognition, and background modeling.

On-line learning The benefit of on-line learning is twofold. It facilitates learning from large databases, and learning when the data is not completely available at the beginning. Hence, the classifier can change over time, i.e., adapt itself.

Features selection A subset of simple images features is selected and changed over time in order to solve the classification problem.

Summarizing, features selection in combination with on-line learning allows to adapt the model by changing features over time. For tracking, the task is narrowed down to differentiate between the current object appearance and the local background. An overview of the tracking approach as a classification problem is visible in Figure 6.2. At each frame, the tracked object is searched for in the neighborhood and its position is extracted relying on the confidence map. The classifier is then updated by taking as positive sample the central patch and as negative samples the four patches around.

6.2.2 Our approach

The approach introduced by Grabner is not directly applicable to our work. It is primarily intended to perform a tracking based only on RGB features and its main scope is to distinguish the target from the background, things that we can do with more reliable means. A step forward has recently been introduced by Luber [16] and consists in using an external detector to find the position of the target (using 3D information as in our case) and then using the classifier in conjunction with a Multi Hypothesis Tracker (MHT) to perform the data association. But also this approach is not what we need nor something we can apply to perform a long term tracking. The problem with these two methods is that in both cases the classifiers learn to distinguish their targets from the background, while we must distinguish each person from all the others in order to recover the track after long lasting occlusions, for example.

We then use another approach to reach our goals. Unlike [12] and [16], that use the target as positive sample and patches in the background as the negative ones, we take as negative samples the other existing tracks (or rather detections not associated to any track). But this is not the only modification we do, as a matter of fact this change forces us to use also other types of features. Instead of computing the features directly on the RGB (or depth) image, we calculate them on the color histogram of the target. We do this because features dependent from the location in the space don't work well when trying to distinguish people in frames distant in time, since the background can change a lot. What we need is to detect features that identify the target and distinguish it from the others. Thus, we calculate those features as follows:

1. We select the RGB image pixels belonging to the person by exploiting the blob mask given by the detector.
2. We compute the 3D color histogram of these points on the three RGB channels.

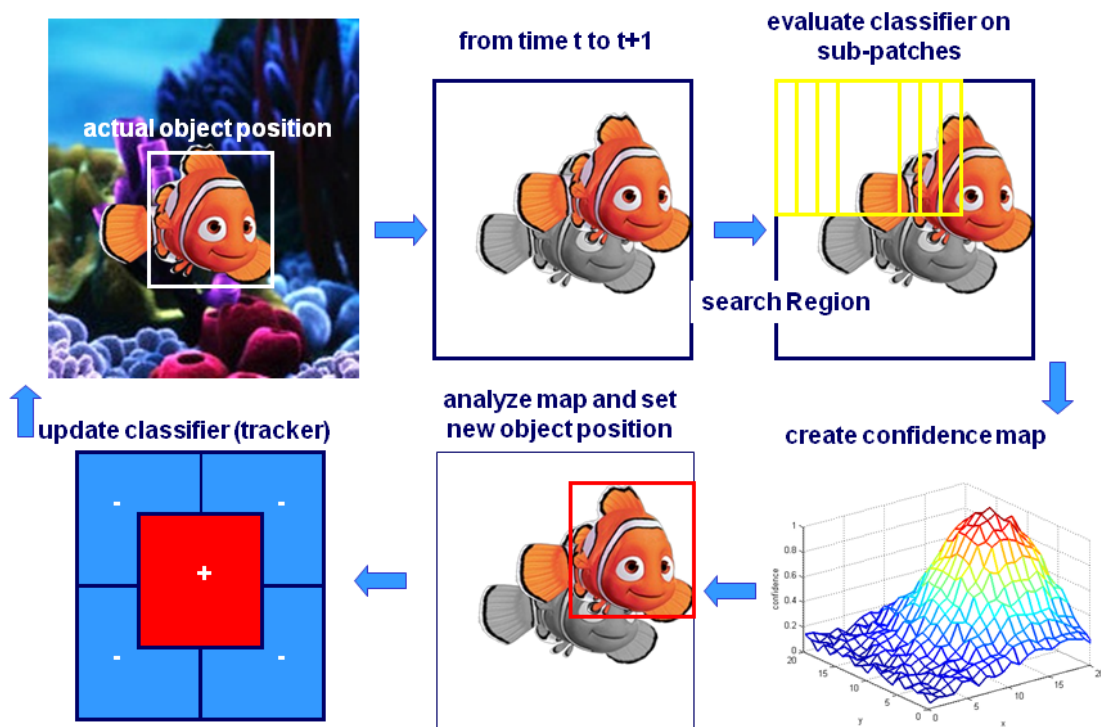


Figure 6.2: Tracking via on-line boosting approach.

3. We consider a set of randomized parallelepipeds (one for each weak classifier) inside the histogram. The feature value is then given by the sum of the histogram values that fall inside the given parallelepiped.

6.3 Data association

Once a set of detections is available, we check whether it corresponds to already existing tracks. The association process is divided into two subsequent steps: at first only the prediction given by the Kalman filter is considered, while, as a second step, we consider only the results given by the classifier. We perform the data association based on the Kalman filter prediction with the Global Nearest Neighbor (GNN) method, described in [15] and [2]. This method considers the residual vector between track i and detection j

$$\tilde{\mathbf{y}}_k(i, j) = \mathbf{z}_k(j) - \mathbf{H} \cdot \hat{\mathbf{x}}_{k|k-1}(i) \quad (6.9)$$

and its covariance $\mathbf{S}_k(i)$ (provided by the filter) to calculate the statistical distance, namely Mahalanobis distance, that is

$$d^2(i, j) = \tilde{\mathbf{y}}_k^T(i, j) \cdot \mathbf{S}_k^{-1}(i) \cdot \tilde{\mathbf{y}}_k(i, j). \quad (6.10)$$

A distance is valid if its value is less than a constant threshold (gate) G , derived from the χ_n^2 distribution table, where n stands for the dimension of the measurement vector, in our case 2. The desired association solution is the one that minimizes the summed total distance.

At this point the remaining detections and tracks are considered for the association based on the classifier. The approach is similar to the one described for the Kalman filter: the remaining tracks and detections are used to fill a table containing for each pair the result given by the track's classifier on the detection. In this case we maintain only values above a predetermined threshold and take as solution the one that maximize the summed values. If, after this step, there are some detections that are still unassociated, they are taken into account for the creation of new tracks.

6.4 Tracks management

The policies of creation/update/deletion of the tracks are really important to get good results from the whole tracking process. For this purpose, for each track we maintain a state as a combination of the following variables:

status Indicates whether the track is **new** or not.

validation Indicates whether the track has been promoted to human or not.

visibility Indicates if the track is completely **visible**, partially **occluded** or completely **not visible** (lost).

How we use these variables is summarized in the remaining of the section.

6.4.1 Initialization

A new track is created from an unassociated detection if the confidence value given by the HOG detector goes over a defined threshold for a fixed number of times while the track is considered **new**. If this happens, the track is promoted to **validated**, otherwise it is discarded.

6.4.2 Update

After the detection-track association, the Kalman filter is updated with the measurement of the theoretical centroid of the cluster. Then if the cluster is not occluded also the classifier is updated. This limitation is given by the fact that, when a person is occluded, some colors can result less visible or even missing, so updating the classifier would decrease the importance of these color features, thus distorting the results of successive classifications.

6.4.3 Recovery

After a full occlusion, a person could be wrongly assigned to a new track instead of the old one. This happens if neither the Kalman filter, nor the classifier correctly associate the new detection to the previous track. But, while the trajectory error of the Kalman filter cannot be corrected, the classifier can manage to recover from this mistake. In particular, during the first frames after its creation, the new track histograms are evaluated by the classifiers of the missing tracks and, whether the result is above a determined threshold, the new track is deleted and the old one recovered.

6.4.4 Removal

After a person becomes occluded or goes out of the scene, the correspondent track is marked as missing. A track is deleted and no more considered if it remains in that state for a certain number of consecutive frames or, as described above, if it is not validated before time runs out.

6.5 Implementation details

The **tracker** is realized as a standalone program that runs at a fixed frame rate. It subscribes to the topic where the detector node publishes its messages and publishes its own results on various topics.

6.5.1 Coordinates conversion

The first operation the tracker does on the data received is to convert their coordinates from the Kinect's system to the world one. This operation is done by using the **tf** library provided by ROS, it uses the frames tree in conjunction with the odometry given by the robot, to calculate the location of any given input point with respect to the initial position of the robot. Since the frame rate of the data published by the Kinect, and consequently the one of the tracker, is different from the publishing rate of the odometry, at each frame we take the last available transform.

6.5.2 Kalman filter

The Kalman filter previously described has been realized by using the **Bayes++** library², an open source, highly optimized, set of bayesian filtering classes. Among the different extensions to the basic Kalman filter available in that library, we decided to use the **Unscented** version. Analyzing various articles about vantages and disadvantages of each different implementation, we have realized the Unscented Kalman filter has prediction capabilities near those of a particle filter, but it is only slightly more computationally expensive than an Extended Kalman Filter

²<http://bayesclasses.sourceforge.net/Bayes++.html>

[2].

An important parameter of the filter, that needs to be appropriately tuned, is the acceleration variance (Equation 6.6). We have measured it empirically by choosing the value that best performed from the visual point of view, since analytical measures gave no useful information. Its value can be set by modifying the parameter `acceleration_variance` in the configuration file (the default value is visible in Table 6.1).

Data association

The data association process, following the GNN directives, consists in three steps:

1. Create the **distance matrix**. It is the matrix containing the Mahalanobis distance $d_{i,j}$ between each track i and detection j .
2. Create the **cost matrix**. It is derived from the distance matrix by substituting the distances above the gate value G with a high number M . The gate value is derived from the χ^2 with 2 degrees of freedom table by choosing the probability p so that $P(d_{i,j}^2 < G) < p$.
3. Solve the association problem with the Hungarian method, a combinatorial optimization algorithm which runs in polynomial time.

Specifically, to solve the assignment problem, we use the **Kuhn–Munkres** algorithm³ for which there exist a lot of efficient implementations. Regarding the value of the gate G , it can be set by setting the correspondent probability p (parameter `gate_distance_probability` in Table 6.1).

6.5.3 Adaboost classifier

The on-line classifier based on Adaboost has been developed from scratch following the guidelines presented in [13, 14]. It consists in a library focused only on the realization of the classifier and separated from the rest of the work, so to be reusable in the future. It permits to use the color features previously described as well as Haar-like features and it can be easily improved. Regarding instead how the classifier is used for the tracking purpose, as previously described, we maintain an on-line classifier for each person present in the scene (also for the ones not visible). Every strong classifier has a fixed number of selectors that choose the best features by selecting the weak classifiers among the available ones (parameters `selectors` and `weak_classifiers` in Table 6.1). The threshold above which a detection is considered to belong to a track is settable by changing the parameter `min_prediction` (Table 6.1).

6.5.4 Tracks management

As described in section 6.4, there are some policies that describe when a track is created, when it can be updated and when it can be deleted. Starting from the creation, we consider a detection as a new track when it is not associated to any previous track and the confidence value given by the HOG detector is over a defined threshold (parameter `min_confidence` in Table 6.1). At this point it is marked as new and remains in this state for a number of frames defined by parameter `frames_remain_new` (Table 6.1). Only during this period the track can be recovered back to an existent one by using the classifier.

The parameter `frames_before_fake` (Table 6.1) is instead used to set the period during which the track must be validated as a human, i.e. if the confidence values of the detections associated with the track go beyond the `min_confidence` (Table 6.1) parameter for a number of times equal

³<https://github.com/saebyn/munkres-cpp>

Parameter name	Default value
~kalman/gate_distance_probability	0.999
~kalman/acceleration_variance	4.0
~recovery/weak_classifiers	250
~recovery/selectors	50
~recovery/min_prediction	9.0
~target/min_confidence	-0.5
~target/frames_before_old	90
~target/frames_before_fake	60
~target/frames_remain_new	30
~target/detections_to_validate	3

Table 6.1: Default parameters for the `tracker` node.

```

uint8 VISIBLE = 0
uint8 OCCLUDED = 1
uint8 NOT_VISIBLE = 2

int32 id

float64 x
float64 y
float64 height

uint8 visibility
pcl_tracking/BoundingBox2D box_2D

```

Listing 6.1: Track message.

to `detections_to_validate` (Table 6.1), the track is validated. If this does not happen, the track is discarded.

The last parameter to consider is `frames_before_old` (Table 6.1). It tells the tracker how long to maintain a track that is not visible and try to recover it by using the classifier.

6.6 Results

The `tracker` node gives results in several ways. At first it publishes a message (Listing 6.1) at each frame for each maintained track: for every person it gives both location and height, as well as the visibility status. Then it outputs both an RGB view of the tracks in the scene (as presented in Figure 6.3) and the current locations in 3D (Figure 6.4).

In the RGB view each track is bounded by a parallelepiped whose lines' thickness indicate whether the person is completely visible (thick lines) or partially occluded (thin lines).

The 3D view instead, is a combination of two different outputs. The first one is the current location of each visible track signaled by a big sphere and the correspondent track number. The second output is instead a point cloud containing the history of the locations each track pass through.

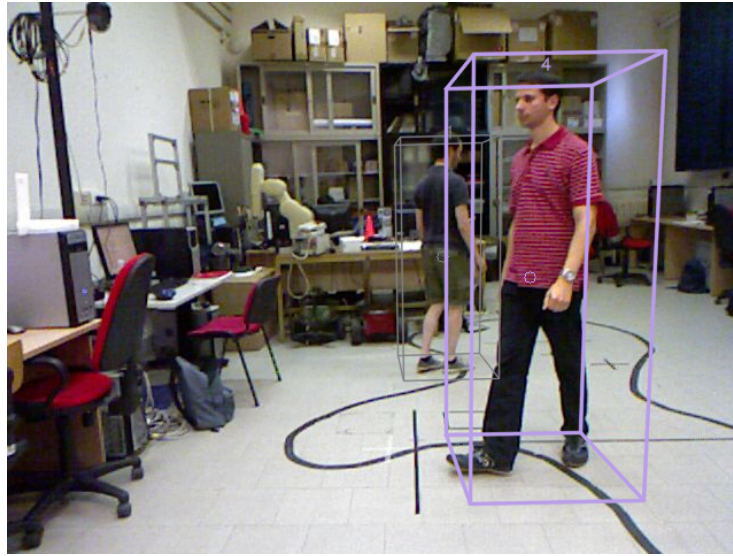


Figure 6.3: RGB output of the tracker node.

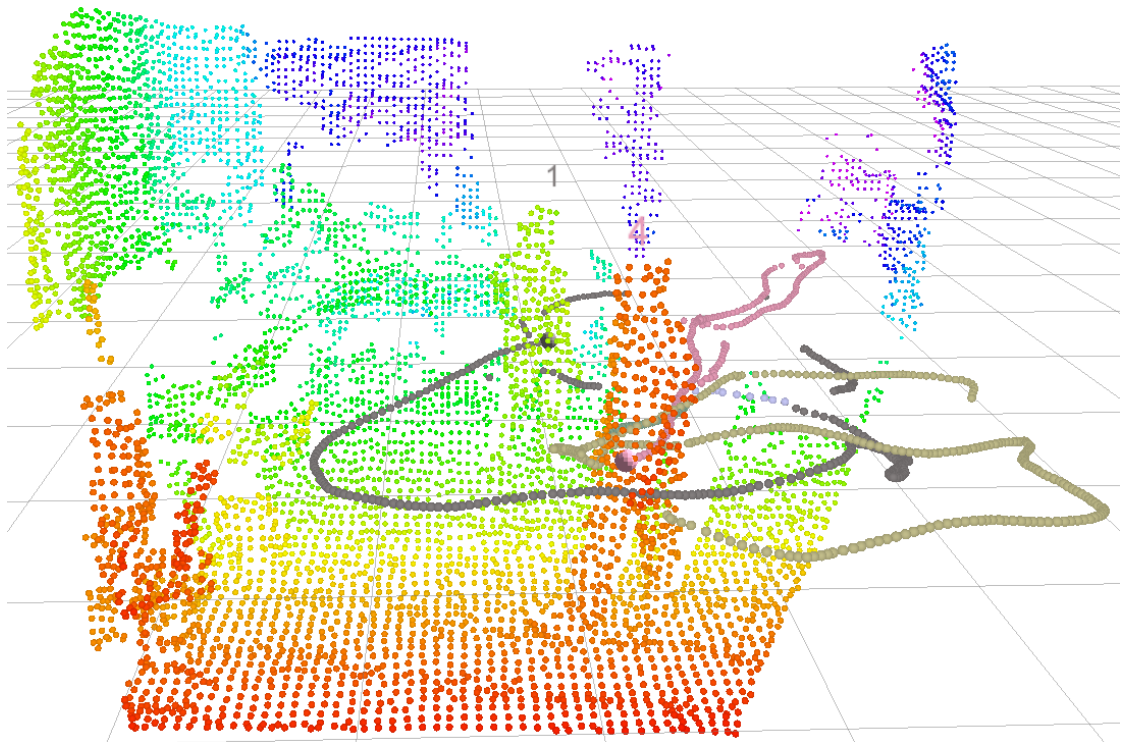


Figure 6.4: 3D output of the tracker node (combined with the down-sampled input point cloud).

Chapter 7

System evaluation

In this chapter we show the performance of our system on RGB-D video sequences collected in an indoor environment with the mobile robot shown in Figure 7.1.

We performed tests both while keeping our platform as static and while moving it on a predefined path. In both cases we acquired videos in three different scenarios of increasing difficulty:

1. No obstacle is present, people move with simple (linear) trajectories.
2. No obstacle is present, people move with complex trajectories and interact with each other.
3. Obstacles are present, people move with complex trajectories and interact with each other.

Every video sequence extends over about 750 frames, thus the total test set includes 4671 frames, 12272 instances of people and 26 tracks that have been manually annotated on the RGB image and that constitute the ground truth.

The chapter is subdivided as following. Section 7.1 demonstrates the effectiveness of the classifier to distinguish among different people. Section 7.2 analyzes the time necessary to perform the different operations and the frame rate reached. Section 7.3 is instead focused on the analytical results that can be extrapolated from the tracking process. In the end, section 7.4 demonstrates a real world application of the tracker.



Figure 7.1: The mobile platform we used for the experiments.

7.1 Classification effectiveness

As previously described, the classifier is used to recover a lost track whenever the Kalman filter fails. We can first evaluate qualitatively how the best features selected by the boosting



Figure 7.2: Situation from which the features are taken.

algorithm change over time. Starting from the situation depicted in Figure 7.2, we show for each person the best three features selected at the beginning of the tracking process and after about 15 seconds. Figure 7.3, 7.4 and 7.5 show the best features for track #1, track #2 and track #3 respectively. As we can see, the features that become the most important to distinguish a track from all the others are the ones based on the leading colors of the track.

However, we cannot stop to a qualitative measure of the features. A proof of the goodness of our approach is visible in Figure 7.6. In this graph we show the prediction values given by the classifier of track #1 when evaluating all the existing tracks. We can see that the results for track #1 are always positive while for the other two tracks (#2 and #3) they are always negative. We can also see its weakness to occlusions, even partial: as soon as track #1 becomes occluded (green line), the value predicted by the classifier starts to decrease and the person is no more recognized as belonging to the track even if it is still associated to it by using the Kalman filter.

As a final step to verify the importance of the classifier, we have evaluated the number of frames each distinct track remains in the scene, with respect to the time the classifier is maintained after the person becomes not visible (Figure 7.7). It shows that when the classifier is not used (i.e. the time is equal to 0 seconds) the tracker creates a lot of different tracks (one for each color) that remain in the scene quite shortly. Instead, when the classifier is maintained for 3 seconds (the default value for all the following tests) the number of tracks created is significantly lower. This indicates that some tracks previously not visible have been recovered, proving the effectiveness of the classifier. In the end, we tried to over use the classifier maintaining it for 30 seconds. As we can see, in each tests there are 3 or 4 predominant tracks. These are exactly the numbers of people visible in each video except for the last one, where the correct number is 3. Figure 7.8 shows the process of recovery taking place during a test. Track #2 goes out of the field of view of the camera and, when it comes back in, it is assigned to a new track (#4), but, in a few frames, it returns to its original ID (#2).

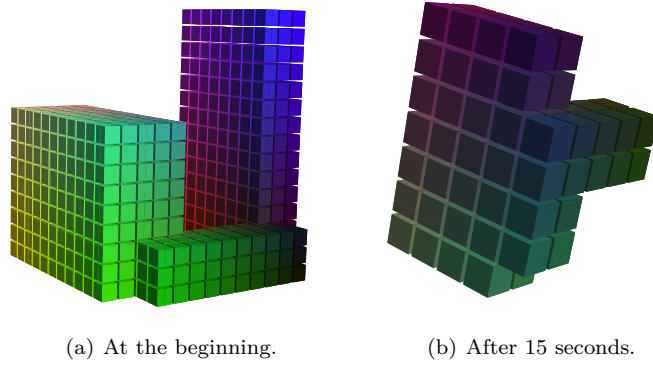


Figure 7.3: Best three features for track #1.

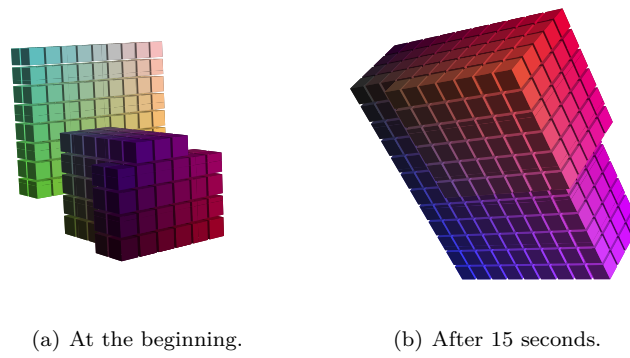


Figure 7.4: Best three features for track #2.

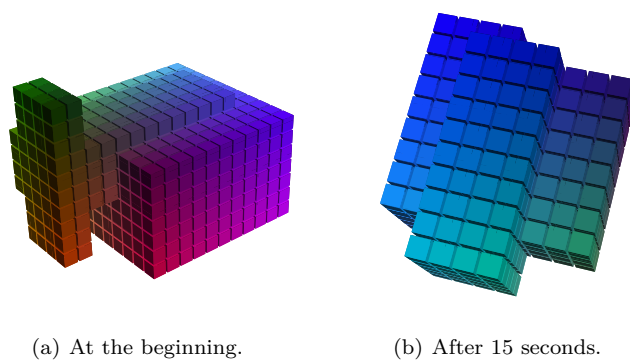


Figure 7.5: Best three features for track #3.

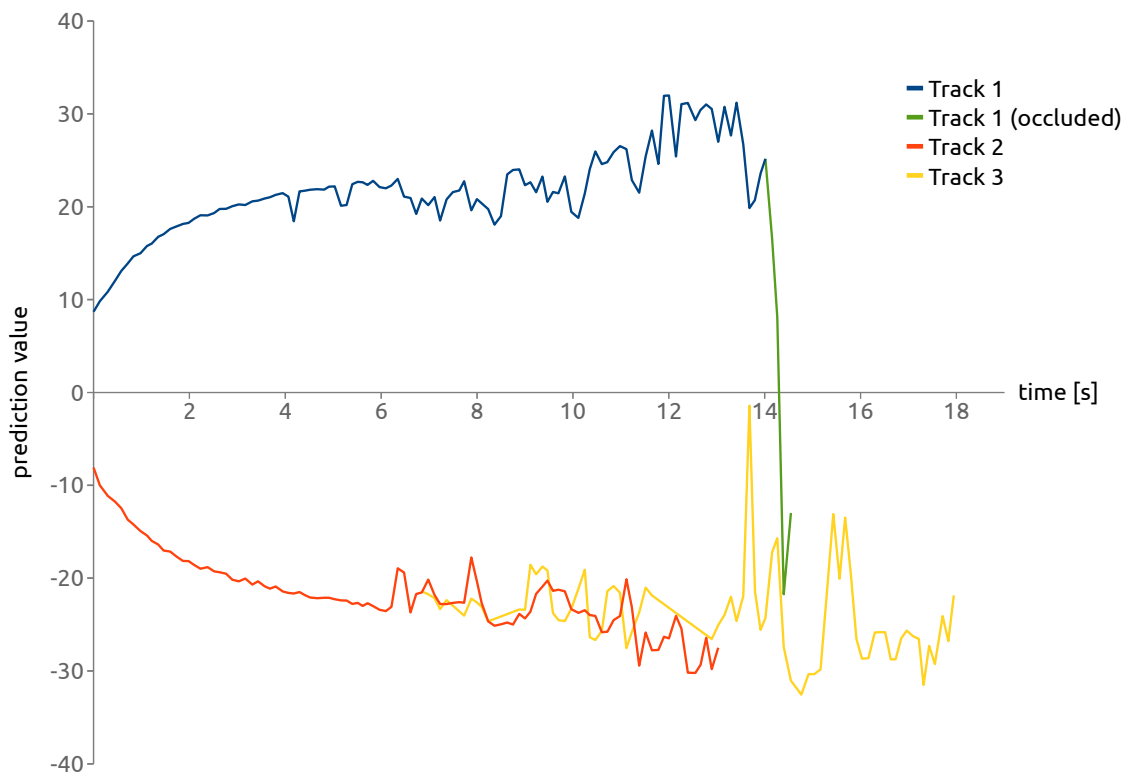


Figure 7.6: Prediction values given by the classifier of track #1.

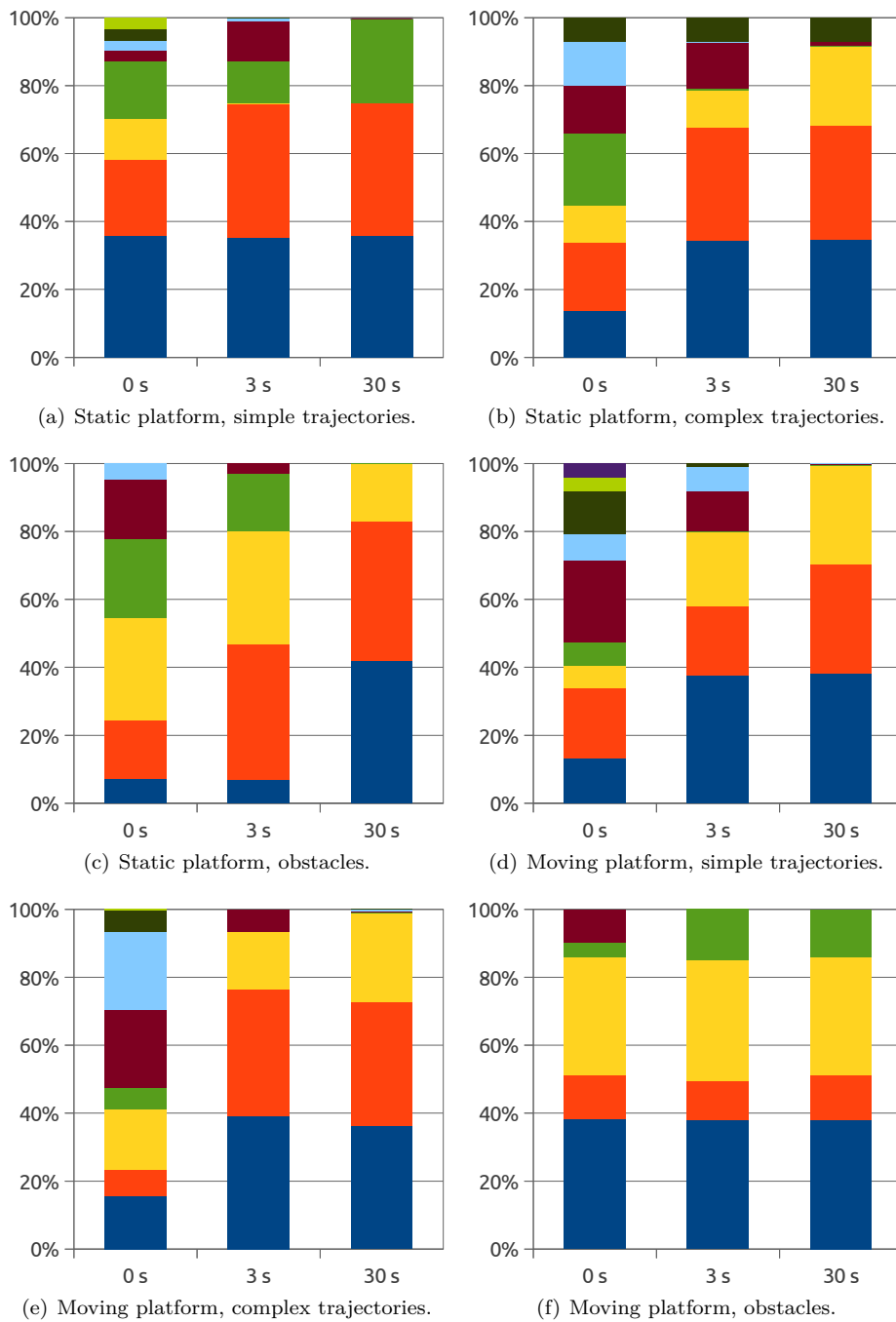


Figure 7.7: Number of frames each distinct track remains in the scene.



Figure 7.8: An example of track recovery. Track #2 becomes #4 after the full occlusion, but, in a few frames, it returns to its original ID.

	Kinect Data	Detector	Detector+Tracker
Simple/complex trajectories	29.971	26.268	26.215
With obstacles	29.967	25.243	25.227

Table 7.1: Average frame rates for our videos.

7.2 Time performances

The entire system is implemented in C++ within ROS, making use of highly optimized libraries for 2D computer vision, 3D point cloud processing and bayesian estimation.

Since one of our objectives was to operate in real-time, we performed some tests to evince the average frame rate of each step.

All of the tests for which we report the results have been made on the videos previously mentioned using a notebook with an Intel i5-520M 2.40GHz (dual core) processor and 4GB of memory (DDR 3).

An overview of the frame rates reached by the three main steps are reported in Table 7.1. They indicate the average rates reached by the Kinect data publication, the detection phase and our complete system (detection and tracking). We have divided the videos into two categories, the ones with obstacles and all the others, since the results were identical for each video belonging to the same group.

As we can notice, for the videos with obstacles our system is slightly less efficient than for the others. This happen because there are more clusters that have to be evaluated by the detector and every evaluation involves the calculus of the HOG confidence, operation that takes about 1.5 ms each. As it can be inferred from the data, the bottleneck of the system is the detector, while the tracking algorithm is very fast. Overall, the whole system can track multiple people in the scene at more than 25 frames per second.

An important parameter that influences the performances of the detection is the size of the voxel grid. In Figure 7.9 we show how the average time of the detection phase changes according to the dimensions of the voxels. In the tests we have limited the size range of the voxel grid from 0.04 meters to 0.15 meters: values lower that 0.04 meters become comparable with the distance error introduced by the Kinect and therefore the clustering operation becomes very difficult, while values above 0.15 meters reduce too much the number of points in the scene. From it we can see that both the time to read the data published by the Kinect and the evaluation of the clusters are approximately constant, while the time to remove the ground is negligible. The only phase that depends on the voxel size is the clustering. The average time needed for the whole detection ranges from 0.025 seconds to 0.029 seconds, both under the imposed threshold of 0.033 seconds (30 Hz). This could be seen as in contrast with the previous results about the average frame rate, but it is not. In fact the time needed is often above the threshold and this imply that some frames are not evaluated because they are overwritten by the newest ones.

7.3 Tracking evaluation

For the purpose of evaluating the tracking performance we adopted the **CLEAR MOT** metrics [3], that consists of two indexes: **MOTP** and **MOTA**. The MOTP indicator measures how well exact positions of people are estimated, while the MOTA index gives an idea of the amount of errors that are made by the tracking algorithm in terms of false negatives, false positives and mismatches. In particular, given that our ground truth does not consist in the metric locations of all people but in their places inside the image, we computed the MOTP index

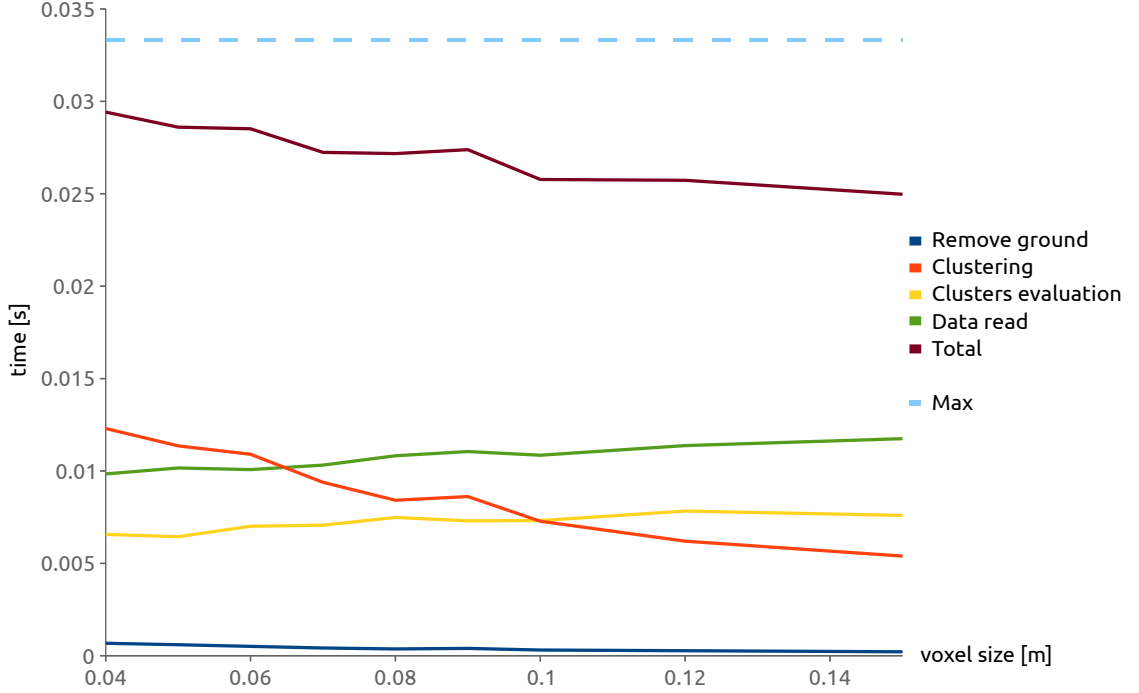


Figure 7.9: Average time for the main phases of the detection process.

as the average PASCAL index [10] (intersection over union of bounding boxes) of the associations between ground truth and tracker results by setting the validation threshold to 0.5.

Regarding the MOTA index, we compute it with the following formula

$$\text{MOTA} = 1 - \frac{\sum_t (\text{FN}_t + \text{FP}_t + \text{ID}_t^{sw})}{\sum_t g_t} \quad (7.1)$$

where FN_t is the number of ground truth people instances (for every frame) not found by the tracker, FP_t is the number of output tracks instances that do not have correspondences with the ground truth and ID_t^{sw} represents the number of times a track corresponding to the same person changes ID over time. In the end g_t is the number of instances for each track.

In Table 7.2 and 7.3 we report, for every test sequence, the MOTP and MOTA indexes, the percentage of false positives and false negatives and the number of ID switches. The average precision of our tracker (MOTP) is always around 80%, while its accuracy (MOTA) slightly decreases from 95% to 89% as we increase the scenario complexity. The minimum accuracy is reached for the video that we captured while moving our platform inside an environment with obstacles that can significantly occlude people. These occlusions mainly have the effect to delay tracks initialization because people hardly get a HOG confidence that is over the threshold we imposed for the track initialization, here -0.5. Thus the false negative percentage increases and the MOTA index decreases. Nevertheless, the track initialization based on the HOG confidence makes our algorithm to totally avoid false tracks. The percentage of false positives shown in the tables are due to bounding boxes generated by real persons that are not perfectly centered on the ground truth data, thus they do not pass the PASCAL test.

In general, our algorithm showed to be very robust even for tracking partially occluded people moving with complex trajectories. We track 78% of every ground truth track with a single ID.

	MOTP	MOTA	FP	FN	ID Sw.	Main Tr.	All Tr.
Simple traj.	79.0%	93.1%	3.8%	2.9%	2	76.7%	97.1%
Complex traj.	79.2%	96.2%	3.1%	0.7%	1	86.3%	99.3%
With obstacles	82.6%	89.6%	2.9%	7.5%	1	74.7%	92.5%

Table 7.2: Tracking evaluation results for static tests.

	MOTP	MOTA	FP	FN	ID Sw.	Main Tr.	All Tr.
Simple traj.	81.7%	92.4%	2.8%	4.6%	4	82.7%	95.4%
Complex traj.	83.7%	90.6%	4.7%	4.5%	3	78.8%	95.5%
With obstacles	84.1%	89.2%	5.3%	5.5%	1	69.2%	94.5%

Table 7.3: Tracking evaluation results for moving tests.

This number raises to 96% if we consider all the IDs that correspond to a certain ground truth person.

As we said in chapter 5, the method we use for selecting which clusters are passed to the data association algorithm is based on a simple check of clusters height from the ground plane. This approach proved to be very effective for people tracking. In fact, if only the clusters that have an HOG confidence above a threshold (e.g. -3) were used to update the tracks, the MOTA index would drop of about 50% (results visible in table 7.4). This is due to the fact that the HOG people detector score decreases very quickly in case of occlusion. Furthermore, the on-line person-specific classifier correctly managed most of track recoveries after a full person occlusion. In particular, for the most complicated scenario we tested (moving platform, obstacles and complex trajectories), it allowed to track 20% more of ground truth tracks without ID switches (Table 7.5).

The last set of tests performed concerns the confidence threshold beyond which the tracker creates a new track. We let it vary from -3 to 0.5 and collect the results reported in Table 7.6 and visible in Figure 7.10. Summarizing they say that a confidence threshold too low (< -1.5) brings the tracker to create a lot of tracks even not belonging to people. A high threshold instead, makes it difficult to initialize new tracks. They also suggest the correct threshold is a value close to -1, the one that minimizes the sum of false positives and false negatives.

7.4 Real-world application: people follower

As a further test for proving the robustness and the real time capabilities of our tracking method we asked the robot to follow a particular person along a narrow corridor with many lighting changes and in a hall where many people are present. In Figure 7.11 and 7.12 the output of the tracking algorithm is visualized for some frames collected during this experiment.

	MOTP	MOTA	FP	FN
Height	84.1%	89.1%	5.3%	5.5%
HOG confidence	81.3%	38.8%	31.3%	25.5%

Table 7.4: Tracking evaluation results for two methods of selecting clusters to be passed to the tracker. (Test video: moving platform, obstacles)

	MOTP	MOTA	FP	FN
Classifier	84.1%	89.1%	5.3%	5.5%
No classifier	82.1%	72.8%	19.6%	7.1%

Table 7.5: Tracking evaluation results with/without the classifier. (Test video: moving platform, obstacles)

Confidence	MOTP	MOTA	FP	FN	ID Sw.	Main Tr.	All Tr.
0.50	83.3%	82.7%	4.2%	13.0%	1	67.5%	87.0%
0.00	84.2%	88.0%	4.2%	7.8%	0	65.1%	92.2%
-0.50	83.9%	90.1%	4.2%	5.6%	3	80.2%	94.4%
-1.00	83.9%	91.4%	4.4%	4.0%	4	79.9%	96.0%
-1.50	84.0%	90.7%	5.7%	3.4%	5	78.5%	96.6%
-1.75	83.7%	79.3%	17.1%	3.2%	6	74.2%	96.8%
-2.00	83.7%	75.4%	21.3%	3.0%	8	58.3%	97.0%
-2.50	83.8%	70.4%	26.5%	2.8%	8	54.0%	97.2%
-3.00	83.7%	70.4%	26.6%	2.7%	8	55.1%	97.3%

Table 7.6: Evaluation of the tracking performance by varying the confidence threshold used to initialize a track. (Test video: moving platform, complex trajectories, no obstacles)

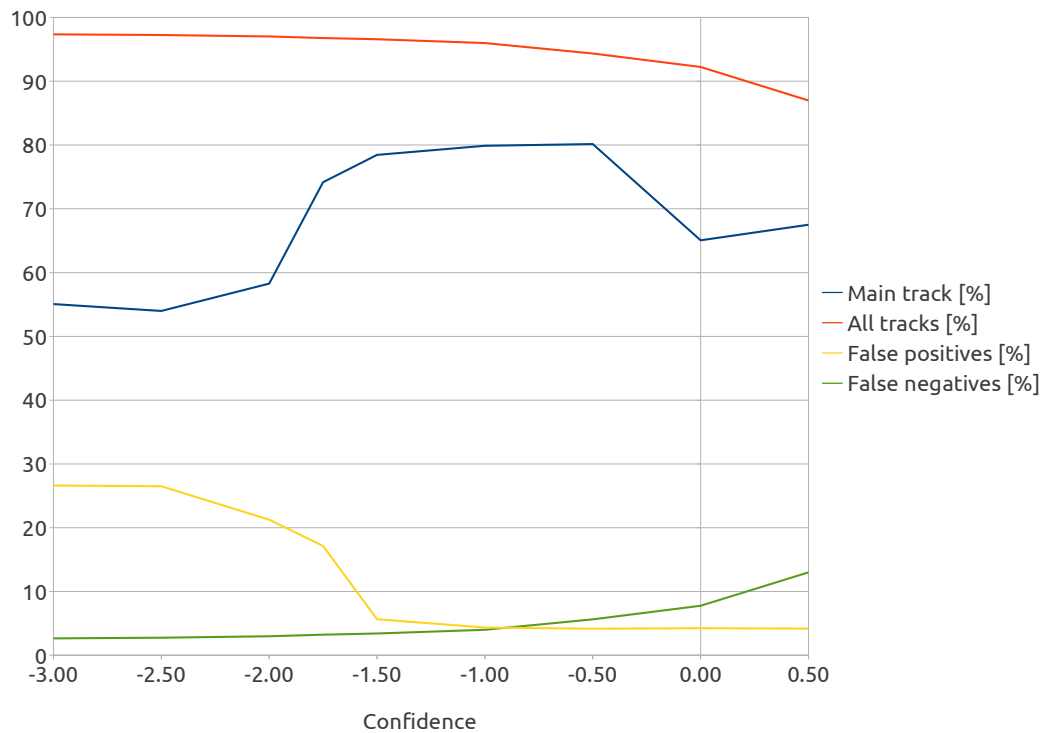


Figure 7.10: The graph representing the evolution of four indicators as a function of the confidence threshold used to initialize a track.

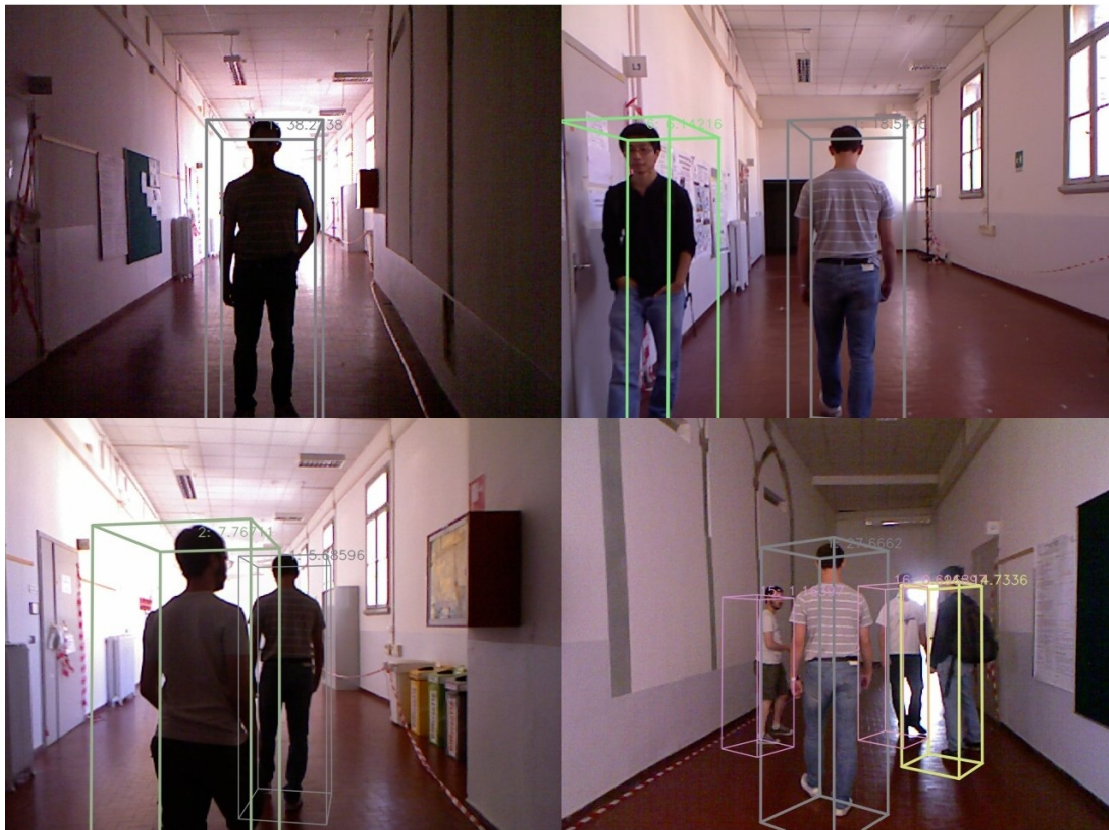


Figure 7.11: People following test: examples of correctly tracked frames when other people are present in the scene.

In Figure 7.11, the tracking (and following) robustness is shown when other people walk next to the followed person or between him and the robot. In Figure 7.12 we can see that the person is correctly tracked and followed by the platform along the corridor, while the lighting conditions considerably change.



Figure 7.12: People following test: examples of tracked frames while a person is robustly followed along a narrow corridor with many lighting changes.

Chapter 8

Conclusions

8.1 Results analysis

In this work we have presented a very fast algorithm for multi-people tracking from mobile platforms equipped with a RGB-D sensor. As we said, one of the key features of the whole system is velocity. It derives both from the use of a voxel filter to down-sample the point cloud generated by the Kinect sensor and from the subdivision of the scene in clusters. The filtering process is necessary to make the point cloud tractable in real time. The division in clusters, instead, is used to limit the number of times the HOG descriptor is calculated on the RGB image, operation that takes about 1.5 milliseconds to be performed. The use of the point cloud has another big advantage. It permits to have a detection even if the person is strongly occluded (an example is visible in Figure 8.1).

Even the results of the tracking process are really good. The robust track initialization obtained by checking the confidence value given by the HOG people detector makes the percentage of false positives very low. Then, the joint use of a Kalman filter and an on-line learnt classifier for each person let us associate the detections to the correct tracks in most cases, even after full occlusions. All of these statements are proved by a series of tests performed in scenarios of increasing complexity and evaluated with popular metrics such as the CLEAR MOT ones.

The algorithm we proposed has also some limitations, mainly given by the Kinect sensor. Two of the most significant ones are the limited field of view and the poor depth estimation over eight meters of distance. The Kinect demonstrated also to suffer the presence of sunlight so that it failed to detect people even less than three meters far from it, when tested in a strongly enlightened hall. Regarding the HOG people detector, it performs very well only when dealing with people completely visible while it is quite useless in presence of partial occlusions. Another key problem affecting it is the time needed to evaluate a single image (128×64 pixels): about 1.5 milliseconds. This fact forced us to set limits to the height a cluster can have in order to be evaluated and then exclude for example children from the detections. In the end, even the classifier proved to be very effective to recover tracks after full occlusions, it needs the scene a constant light. In fact, when there some changes in the brightness of the environment, its effectiveness decreases. This is caused by the type of features we use to classify the tracks, they are not very robust to lighting changes.



Figure 8.1: An example of a person detected even if only his head was visible.

8.2 Future developments

To overcome the limitations imposed by the narrow field of view of the Kinect, we have prepared the system to support more than one sensor but no one has tested it. Thus, the first improvement to the system is to make two or more Kinects work together. This leads to two different problems. The first one is the necessity to correctly calibrate each sensor with the others. The second one, instead, is that two sensors imply to have two detectors running at the same time and this could decrease the frame rate reached.

A new approach that should help in reducing the time spent to elaborate point clouds is the use of ROS **Nodelets** instead of nodes. Nodelets are designed to provide a way to run multiple algorithms on a single machine, in a single process, without incurring copy costs when passing messages intraprocess. A future work is to rewrite the code transforming each node into a nodelet and test the gain in terms of time.

Another improvement that promises to be really effective is the use of algorithms implemented on GPUs. The latest version of the PCL has started to move in this direction but the algorithms nowadays implemented are few. In the future the use of such optimized algorithms could lead to an abatement of the time needed to manipulate point clouds and images with great benefits to the whole system.

As said above, the type of features used by the classifiers are not really robust to brightness changes. Thus, as a future work, there is the necessity to look for other types of features that can help in solving this problem.

Bibliography

- [1] Max Bajracharya, Baback Moghaddam, Andrew Howard, Shane Brennan, and L H Matthies. Results from a real-time stereo-based pedestrian detection system on a moving vehicle. In *Workshop on People Detection and Tracking IEEE ICRA*, 2009.
- [2] Nicola Bellotto and Huosheng Hu. Computationally efficient solutions for tracking people with a mobile robot: an experimental evaluation of bayesian filters. *Autonomous Robots*, 28:425–438, May 2010.
- [3] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *J. Image Video Process.*, 2008:1:1–1:10, January 2008.
- [4] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O’Reilly, Cambridge, MA, 2008.
- [5] Michael D. Breitenstein, Fabian Reichlin, Bastian Leibe, Esther Koller-Meier, and Luc Van Gool. Robust tracking-by-detection using a detector confidence particle filter. In *IEEE International Conference on Computer Vision*, October 2009.
- [6] A. Carballo, A. Ohya, and S. Yuta. People detection using range and intensity data from multi-layered laser range finders. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 5849 –5854, 2010.
- [7] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition*, volume 1, pages 886–893, June 2005.
- [8] A. Ess, B. Leibe, K. Schindler, and L. Van Gool. A mobile vision system for robust multi-person tracking. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1 –8, 2008.
- [9] A. Ess, B. Leibe, K. Schindler, and L. Van Gool. Moving obstacle detection in highly dynamic scenes. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation, ICRA’09*, pages 4451–4458, Piscataway, NJ, USA, 2009. IEEE Press.
- [10] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88:303–338, June 2010.
- [11] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag.

- [12] Helmut Grabner. *On-line Boosting and Vision*. PhD thesis, Graz University of Technology, Graz, AT, August 2008.
- [13] Helmut Grabner and Horst Bischof. On-line boosting and vision. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, pages 260–267, Washington, DC, USA, 2006. IEEE Computer Society.
- [14] Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In Mike J. Chantler, Robert B. Fisher, and Emanuele Trucco, editors, *BMVC*, pages 47–56. British Machine Vision Association, 2006.
- [15] Pavlina Konstantinova, Alexander Udvarev, and Tzvetan Semerdjiev. A study of a target tracking algorithm using global nearest neighbor approach. In *Proceedings of the 4th international conference conference on Computer systems and technologies: e-Learning*, pages 290–295, New York, NY, USA, 2003. ACM.
- [16] Matthias Luber, Luciano Spinello, and Kai O. Arras. People tracking in rgb-d data with on-line boosted target models. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 2011.
- [17] Christian Martin, Erik Schaffernicht, Andrea Scheidig, and Horst-Michael Gross. Multi-modal sensor fusion using a probabilistic aggregation scheme for people detection and tracking. *Robotics and Autonomous Systems*, 54(9):721–728, 2006.
- [18] Oscar Mozos, Ryo Kurazume, and Tsutomu Hasegawa. Multi-part people detection using 2d range data. *International Journal of Social Robotics*, 2:31–40, 2010.
- [19] Luis E. Navarro-Serment, Christoph Mertz, and Martial Hebert. Pedestrian detection and tracking using three-dimensional ladar data. In *FSR*, pages 103–112, 2009.
- [20] Nikunj C. Oza and Stuart Russell. Online bagging and boosting. In *In Artificial Intelligence and Statistics 2001*, pages 105–112. Morgan Kaufmann, 2001.
- [21] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.
- [22] J. Satake and J. Miura. Robust stereo-based person detection and tracking for a person following robot. In *Workshop on People Detection and Tracking IEEE ICRA*, 2009.
- [23] Luciano Spinello and Kai O. Arras. People detection in rgb-d data. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 2011.
- [24] Luciano Spinello, Kai O. Arras, Rudolph Triebel, and Roland Siegwart. A layered approach to people detection in 3d range data. In *Proc. 24th AAAI Conference on Artificial Intelligence, PGAI Track (AAAI'10)*, Atlanta, USA, 2010.
- [25] Luciano Spinello, Matthias Luber, and Kai O. Arras. Tracking people in 3d using a bottom-up top-down people detector. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'11)*, Shanghai, China, 2011.
- [26] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. volume 1, pages I–511–I–518. IEEE Comput. Soc, 2001.