POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

PEOPLE DETECTION AND TRACKING FROM A SMALL-FOOTPRINT MOBILE GROUND ROBOT USING AN RGB-D SENSOR

Relatore: Prof. Vincenzo Caglioti

Correlatore: Dr. Alessandro Giusti
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Lugano

Tesi di Laurea di:
Armando Pesenti Gritti, matricola 782916
Oscar Tarabini, matricola 783393

Sessione di Laurea Aprile 2014
Anno Accademico 2012-2013

Make everything as simple as possible, but not simpler.
Albert Einstein

Abstract

Small-footprint mobile ground robots are becoming increasingly popular as reference platform for robotic researches. In fact, they are relatively cheap, easily transportable thanks to their small size and little weight, power efficient and most of them are also provided with some basic moving and sensing equipment (along with low-level software functionalities) that represents a convenient starting point for implementing and testing new algorithms. Moreover, this typology of robots can be found in several commercial appliances that are affecting the everyday life of a growing number of families in tasks such as vacuum cleaning, lawn mowing and children entertainment.

Such small-sized robots are by necessity equipped with sensors which lie close to the ground. Reliably detecting and tracking people from this viewpoint is a challenging problem, whose solution is a key requirement for many applications involving sharing of common spaces and close human-robot interaction.

We present a robust solution for cluttered and unstructured indoor environments, using an inexpensive RGB-D sensor such as the Microsoft Kinect or Asus Xtion. Our machine-learning based system is able to detect and track multiple people even in difficult scenarios, where occlusions are frequent and several complex objects could cause potential false detections.

We have also created evaluation datasets to quantitatively analyze the performance of the system and to allow an effective comparison with both existing state-of-the-art and future works.

A real-time, open-source, ROS-enabled implementation is available and has been successully deployed on two different robot platforms: *Turtlebot*, a small wheeled commercial robot supplied to us by the *Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)*, and *StarlETH*, a small quadruped robot developed at the *Swiss Federal Institute of Technology Zurich (ETH)*.

Sommario

Nell'ambito della ricerca robotica si registra un interesse crescente verso robot mobili di piccole dimensioni ed ingombro limitato, i quali vengono spesso scelti come piattaforme di riferimento su cui sperimentare nuove idee e funzionalità. La loro praticità, il loro costo contenuto, la possibilità di poterli trasportare facilmente, il loro peso ridotto ed il conseguente minor fabbisogno energetico, la dotazione, per la maggior parte di essi, di un sistema basilare di attuazione e di percezione, li rendono un ottimo punto di partenza per esplorare la fattibilità ed analizzare le prestazioni di nuovi algoritmi.

Inoltre, è possibile ritrovare questa tipologia di robot anche in vari prodotti commerciali sempre più diffusi nelle nostre case, ad esempio quelli rivolti a lavori domestici che possono essere facilmente automatizzati, come la pulizia del pavimento o il taglio dell'erba del giardino, nonchè all'intrattenimento dei bambini.

I sensori di cui questi robot sono forniti devono essere inevitabilmente posizionati vicino al pavimento. Riuscire ad individuare e seguire correttamente il movimento delle persone presenti nell'ambiente, avendo a disposizione solamente le informazioni percepite da un punto di vista così basso, si rivela un problema molto complesso. Una sua soluzione è però necessaria in quanto rappresenta un requisito di fondamentale importanza per poter favorire lo sviluppo di tutte quelle applicazioni che prevedono un'interazione ravvicinata tra il robot e le persone, in particolare quando questi devono condividere in modo efficiente degli spazi comuni.

Il nostro lavoro rappresenta una soluzione al problema appena citato, applicabile ad ambienti interni, non strutturati e caratterizzati dalla presenza di più persone e di vari oggetti. L'input del sistema è fornito da un sensore RGB-D, come Microsoft Kinect o Asus Xtion, e l'approccio basato su *machine-learning* permette di identificare le persone e distinguerle da altri ostacoli, anche negli scenari in cui le occlusioni sono frequenti e molti oggetti, visibili solo in parte, potrebbero dar luogo ad un'errata classificazione.

Al fine di analizzare quantitativamente le prestazioni del sistema e permettere un confronto sia con lavori esistenti, sia con quelli futuri, abbiamo realizzato e messo a disposizione alcuni *dataset* comuni per la valutazione.

L'implementazione finale del sistema, contenente gli algoritmi sviluppati e descritti in questa tesi, si caratterizza per essere *real-time*, *open-source* e per la possibilità di essere facilmente integrata in robot basati sul *middleware* ROS. Grazie a quest'ultima funzionalità, in particolare, è stato facilmente possibile verificare le prestazioni complessive anche su due robot reali: *Turtlebot*, un piccolo robot commerciale dotato di ruote messo a nostra disposizione dall'*Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)*, e *StarlETH*, un piccolo robot quadrupede sviluppato presso l'università *Swiss Federal Institute of Technology Zurich (ETH)*.

Acknowledgments

Foremost, we would like to express our sincere gratitude to our supervisor Alessandro Giusti for his continuous and prompt support, for his patience in clarifying all our doubts, for his motivation and enthusiasm also in helping us prepare the additional publications related to our project and for all his insightful comments.

Besides our supervisor, we would like to thank Jérôme Guzzi and the *Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)*, for their advices, their encouragement and for providing us the Turtlebot robot as a reference platform for our thesis project.

Our sincere thanks also goes to Péter Fankhauser and the *Swiss Federal Institute of Technology Zurich (ETH)*, for offering us the great opportunity to test our algorithm on the legged robot StarlETH.

Last but not the least, we would like to thank our families for their support and their patient availability in being precious "actors" in the innumerable videos we needed to record throughout the thesis development.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Many emerging robotics applications require that mobile robots and humans share common spaces and interact with each other. One of the main issues to be solved to enable these scenarios is reliable detection and tracking of nearby humans. Only after this issue is solved, robots can manage tasks such as the following.

- Safely and efficiently navigate an environment shared with pedestrians [25], following predictable paths that stay out of the pedestrians' way (a form of indirect interaction).
- Adjust their position, following *social rules* that take into account personal spaces [50], in order to facilitate direct interaction and cooperation [37]; an example is a robot moving at an appropriate distance in front of an human with which an interaction is desired or ongoing [53].
- Carry out application-specific tasks such as: monitoring crowd movements; letting robots learn the human trajectories [48]; following a specific person in a cluttered environment to provide assistance or predict its intentions [32]; standing in a line with humans [43].

RGB-D sensors, such as the Microsoft Kinect [3] or Asus Xtion [1], are an ideal and widely-adopted tool for human perception: in this work, we deal with the problem of *detecting* and *tracking* one or multiple people when the sensor is mounted on the robot itself. More specifically, we focus on the issues arising from configurations in which the sensor viewpoint is close to the ground: this is the case, for example, of the widely popular Turtlebot [7] and Kobuki [2] robots; however, this is in general true for any small-footprint ground robot, such as autonomous vacuum cleaners, small platforms for surveillance, monitoring, item delivery or children entertainment.

Figure 1.1: Images from a low-lying viewpoint, highlighting the challenging conditions for people detection and tracking.

RGB-D sensors were designed for perceiving people, and many techniques have been published for solving problems much more demanding than detection and tracking, with impressive results [45, 21]. Unfortunately, existing algorithms are not suitable for the task at hand, because in our case one or more key assumptions, such as the visibility of the upper body of the subject in the frame, are not met (we review related literature in Chapter 2). Detecting and tracking people from a low-lying viewpoint is a fundamentally different and mostly unexplored problem with unique challenges (see Figure 1.1):

- occlusions (by other subjects, other robots, or even short objects such as tables and chairs) happen frequently and are severe;

- when the subject is close to the robot – i.e. just when accurate perception is of paramount importance – most of the subject falls outside of the sensor's field of view; the only visible feature is part of the person's legs, which exhibit a complex, highly-variable appearance with irregular motion and no obvious markers for detection;

- typical indoor environments are cluttered with distractors (such as table legs and chairs) which are not easily distinguishable from human legs;
- the robot's own motion disallows to adopt straightforward techniques for segmenting moving objects.

Our **main contribution** is an efficient technique which robustly solves people detection and tracking in this challenging scenario, under the assumption that the floor is planar – which is reasonable for most indoor environments drivable by a ground robot. More specifically, our system detects each person visible in the sensor field of view and returns its position relative to the robot and its approximate orientation; moreover, each subject is also tracked through time as long as it stays visible (note that we do not deal with the problem of person re-identification after a track is lost for a long time).

In our approach, leg-like objects protruding from the floor are initially segmented, then classified as either human legs or distractors by means of a statistical classifier learned from a large, 26000-instance training dataset acquired in 15 different real-world environments; the resulting probabilistic information is processed in a two-level tracking framework, which associates data in space and time to detect and track people. We illustrate our algorithm in detail using real-world datasets, and showcase results in various complex, realistic scenarios (Chapter 8), representing environments disjoint from those used during training. Quantitative results show significant performance improvements over alternative approaches, highlighting the contributions of the learned classifier and tracking framework.

Extensive demonstration videos and a real-time, open-source, ROS-enabled implementation are available as supplementary material at `http://bit.ly/perceivingpeople`.

A preliminary version of our system was demonstrated to the public in the HRI 2014 video track [24] (9th ACM/IEEE International Conference on Human-Robot Interaction, Bielefeld University, Germany).

Chapter 2

Related Work

Due to the importance of people detection and tracking for practical applications, many solutions have been proposed in the robotics literature.

2.1 2D Laser Range Data

Early works mainly exploited 2D laser range data on a horizontal plane: depending on the sensor height, different horizontal sections of the human body are considered, the most common being the waist [23] or the legs [8, 38, 49]. Range measurements are first spatially clustered on the horizontal plane, then each cluster is classified using either machine learning [8] or model fitting algorithms [33]. In the former, several features are extracted from the cluster points and given in input to a trained classifier; in the latter, simpler geometric considerations about the spatial relations of the cluster points are exploited to detect people.

Approaches based on 2D laser data are efficient, but operate on perceptions limited to one (or few) planar sections of the scene; then, depending on the sensor height, a human and a different object could yield very similar observations and therefore be impossible to discriminate [42]. These sensors typically provide a wide field of view, often reaching a 360 degree covering, but their low angular resolution (relative to the one of an RGB-D sensor) limits the spatial density of collected measures. All these limitations yield a reduced accuracy compared to methods based on more powerful sensors, as we verify in Chapter 8. Moreover, the cheapest 2D laser scanners cost more than 4 times as much as an RGB-D sensor and inexpensive robots such as the Turtlebot [7] and Kobuki [2] are not equipped with one in their default configuration.

On the other hand, it is interesting to note that several laser-based ap-

proaches [8, 38, 49] share with our system the challenging requirements that arise from adopting a low-lying sensor position, such as dealing with the problem of pairing legs belonging to the same person [9].

2.2 RGB Data

There exist several works that deal with the detection and tracking of people in RGB images acquired with a monocular camera. Some of their practical applications concern the analysis of surveillance videos and sport sequences, often exploiting advanced tracking solutions and achieving remarkable results also in crowded scenes [54] or even with an uncalibrated moving camera [16]. Regardless of the fact they usually assume that the entire person is inside the sensor field of view [47, 54, 16] or that the sensor is still [41] (assumptions not holding in our context), most of them detect and track people positions and velocities only in the image plane, without inferring 3D information.

In [28] it is proposed a method to connect the 2D person shape to the 3D person pose by relying on prior knowledge about human motion learned from training data, yet without facing the problem of placing the person in the 3D scene.

The connection to the 3D scene could be partially realized by assuming a prior knowledge of the environment [44]. In such structured environments, in fact, it is usually possible to identify some clues (e.g. orthogonal or parallel lines, known distances, etc.) that allow to back-project the pixels into the original 3D points. This, however, is not a realistic assumption to take when developing a general application for mobile robots.

A common approach to reconstruct the 3D point cloud from RGB data is to combine multiple images of the scene recorded from different cameras, which is usually referred to as stereo camera vision [31, 36]. The same information could alternatively be collected by considering multiple consecutive frames captured by a single moving camera and then applying the algorithms for stereo camera vision [19]. In both cases, however, besides the difficulty to robustly obtain dense *depth* maps (e.g. due to variable lighting conditions and difficulties in feature detection and matching), an extra significant computational cost is inevitably added.

This is the main reason why RGB-D sensors are a common, practical and efficient alternative to obtain a dense 3D point cloud of the environment, while also keeping the color information typical of an RGB camera.

2.3 RGB-D Data

Despite being cheaper than 2D laser scan technology, RGB-D sensors provide aligned color and depth images of the environment in their field of view, allowing to reconstruct a 3D point cloud of the observed space. Furthermore, one may easily merge 2D RGB image processing tecniques with 3D point cloud analysis, which is a key advantage over a monocular video stream and allows to avoid the additional computational cost required by stereo vision setup. Since they use structured infrared lighting, RGB-D sensors are not suitable for outdoor applications; in these cases, 3D point clouds of the environment in front of the robot can be instead acquired by means of 3D laser scanners [51] (or, taking into account the previously mentioned limitations, stereo cameras [31]). Roboticists use RGB-D sensors for disparate indoor perception tasks, ranging from indoor environment mapping [27] to human body pose and gesture recognition [45].

A fundamental issue to be solved when perceiving people is the *detection* problem, that is, determining whether a person is in the field of view, and its approximate position. Most algorithms dealing with people perception from RGB-D data focus instead on higher-level tasks, such as pose recovery, action or gesture recognition. In these scenarios, a reasonable assumption is that *at least the subject's upper body is visible*: then, the detection problem is solved through simpler techniques such as: looking for obvious person markers (e.g. the head [57]); filtering point clusters matching the expected approximate dimensions of a person (most importantly, the height [12]); or using statistical classifiers trained on the whole shape of the person [52]. However, in order to ensure that the subject's upper body is visible, the viewpoint can not lie too close to the subject; moreover, in order to avoid occlusions, expecially in crowded or cluttered environments, the sensor should stay in an elevated position, at least at the level of the subjects' chest or eyes. The mentioned algorithms, thus, cannot be directly used on small-footprint ground robots, which must by necessity be short, with sensors lying close to the ground, and which may get very close to the subjects. In this case, the detection problem becomes significantly more complicated, due to the uncommon viewpoint, from which the lower part of the human body is the most prominent feature and severe occlusions frequently occur in cluttered scenarios.

Moreover, several RGB-D based methods not developed explicitly for robotics applications assume that the sensor is still, and segment moving objects using background subtraction [5], which is not a suitable approach if the sensor is mounted on a mobile robot, since the whole scene moves w.r.t.

the reference frame of the sensor. There exist works dealing with foreground segmentation with moving sensor, such as in [35], which could be integrated but only at the cost of significantly increasing the computational complexity of the whole system.

A relevant related work [26] faces the problem of people detection in presence of occlusions, by splitting the 3D point cloud in layers according to the height from the floor, then finding clusters in each layer and classifying each cluster as a human segment or not: different clusters classified to be part of a human are finally connected according to their relative distance in order to reconstruct the visible part of the person. However, only occlusions affecting the lower part of the subject are considered (conversely, in our scenario the legs are often the only visible part, which raises a number of challenging issues detailed in the following chapters), and each frame is independently processed: instead, our method adopts tracking at different levels of the pipeline, which yields a significant advantage in overall accuracy, which we quantify in Chapter 8.

To the best of our knowledge, our approach is the first to demonstrate practical and robust detection and tracking of people from an RGB-D sensor mounted on a mobile, low-lying viewpoint.

Chapter 3

System Overview

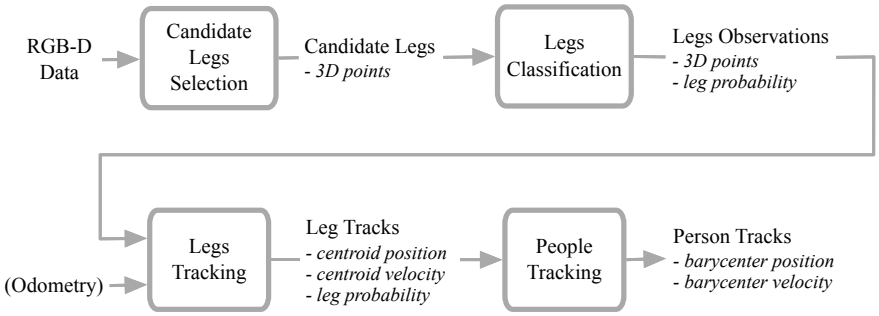


Figure 3.1: System overview

Legs are the most prominent (and, for close subjects, the only) feature which is visible from our considered point of view. Hence, our approach is based on the detection of legs in the 3D point cloud. However, legs have a very variable appearance, and many objects with a similar shape are present in cluttered environments: therefore, our system first *detects candidate legs* (Chapter 4) using simple rules that yield high sensitivity (i.e. rarely misses an actual leg) but suffer from low specificity (i.e. detects many spurious objects); then, a statistical classifier previously learned from large training datasets is applied to candidates in order to discriminate actual legs from other objects (Chapter 5). Individual leg observations are tracked along time (Chapter 6) and their leg probabilities continuously updated by integrating classifier outputs in time: at this stage, people can finally be detected and tracked using leg positions and likelihoods to generate people barycenter observations for a Kalman filter. An overall view of the system is presented in Figure 3.1.

Chapter 4

Data Preprocessing and Selection of Candidate Legs

Starting from the RGB-D data acquired from the sensor, the 3D point cloud corresponding to the viewed scene is reconstructed and clusters of points emerging from the floor are selected as candidate legs.

4.1 RGB-D Data

An RGB-D sensor consists of an RGB camera, an infrared camera and an infrared emitter which together allow to compute depth information aligned with the RGB data. This is realized through a structured light approach: the infrared emitter projects a fixed light pattern, whose deformation when striking the surfaces on its way, captured by the infrared camera, is used to calculate the depth information of the objects in the scene. An RGB-D sensor thus enriches the color information for each pixel found in a common *RGB image*, with a *Depth image*, which contains for each pixel the distance between the corresponding projected 3D point and the image plane. Knowing the intrinsic calibration parameters of the infrared camera f_x, f_y, u_0 and v_0 (respectively, the focal length in terms of pixel width and height and the pixel coordinates of the principal point), it is possible to reconstruct the original 3D coordinates x, y, z in the sensor reference system $\mathfrak{R}_{sensor}$ (see Figure 4.1) of a 3D point projected into the pixel with coordinates u, v and depth value d:

$$x = -\frac{(u - u_0)}{f_x} z \qquad y = -\frac{(v - v_0)}{f_y} z \qquad z = d \tag{4.1}$$

For the most common RGB-D sensors, which are based on PrimeSense [4]

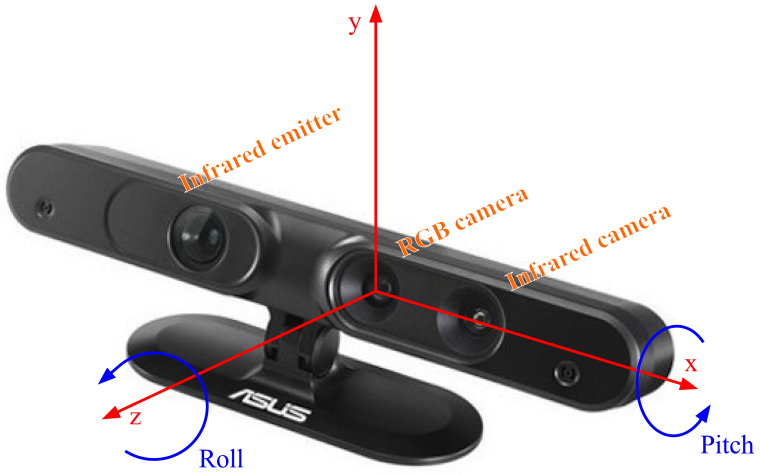


Figure 4.1: Sensor reference system $\mathfrak{R}_{sensor}$. The RGB-D device aligns RGB and Depth images in such a way that the principal point of the infrared camera coincides with the one of the RGB camera.

technology (e.g. Microsoft Kinect and Asus Xtion), the intrinsic parameters, when using 640×480 resolution, can be assumed with good approximation to be: $f_x = 525.0$, $f_y = 525.0$, $u_0 = 319.5$ and $v_0 = 239.5$.

It is quite common to find some pixels in the depth image with $d = 0$. This situation may occur when the object is too close to the sensor or when the infrared light is not properly reflected by an object (e.g. due to its material properties or shading by foreground objects). Ignoring these pixels, which contain erroneus measurements, the point cloud is reconstructed and results to contain approximately 250000 3D points (for 640×480 resolution videos).

4.2 Point Cloud Downsampling

Initially, point cloud data is preprocessed by discarding points beyond the declared range of the sensor (which are affected by large amounts of noise [34]): for both Microsoft Kinect and Asus Xtion it is 350 cm.

Then, the point cloud is downsampled using a voxel grid approach [6]. The voxel grid is a set of 1 cm side 3D boxes in space, built over the input point cloud data. All the points falling in the same voxel are approximated with their centroid. This approach is a bit slower than approximating them with the center of the voxel, but it allows to represent the underlying surface more accurately.

This technique reduces by more than 80% the amount of data to be

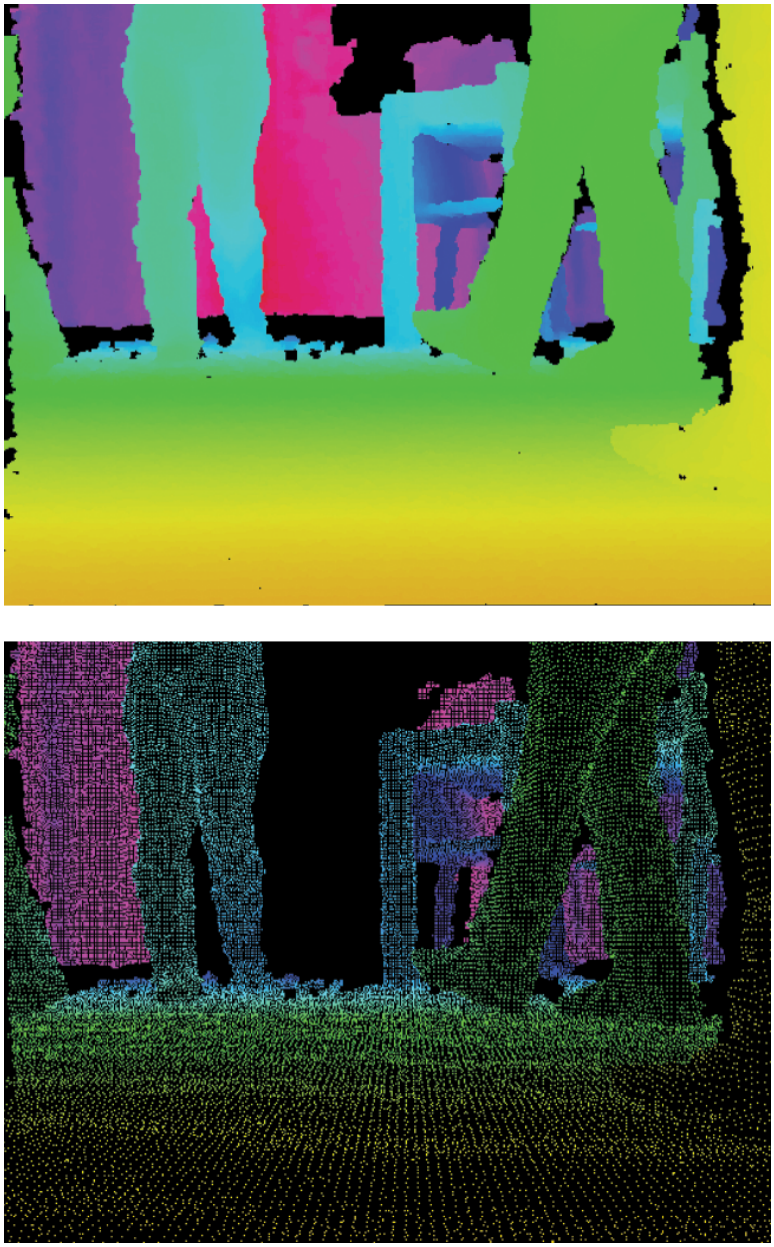


Figure 4.2: Original (top) and downsampled (bottom) point cloud. The various colors represent different distances from the sensor.

processed and yields a more homogeneous point density at different distances from the sensor (Figure 4.2). Moreover, it mitigates the differences among point clouds extracted from videos with various resolutions, creating more similar input data for the successive steps of the algorithm.

4.3 Floor Plane Detection

Because the sensor position and rotation may be only approximately known, we initially detect the floor plane in the 3D point cloud using a RANSAC-based [22] technique, which has been shown to be significantly faster than alternative approaches such as the 3D Hough transform [14].

First of all, only the points with a negative y-coordinate in $\mathfrak{R}_{sensor}$ (see Figure 4.1) are considered in this process, since most of the floor points are very likely to fall into the half-space below the xz plane in $\mathfrak{R}_{sensor}$ (this is the case at least when the sensor is not used upside down, in which situation it would just be sufficient to properly rotate the input RGB and Depth images of an angle of 180 degrees before computing the point cloud).

We randomly choose three non-colinear points P_1, P_2 and P_3 among them and compute the fitting plane $ax + by + cz + d = 0$. Being $\mathbf{n} = (a, b, c)$ a vector orthogonal to the plane, we compute it as the cross product of two vectors lying on the target plane:

$$\mathbf{n} = (P_2 - P_1) \times (P_3 - P_1) \tag{4.2}$$

The last unknown parameter d is then found by constraining the plane to pass through the point $P_1 = (x_1, y_1, z_1)$:

$$d = -ax_1 - by_1 - cz_1 \tag{4.3}$$

Once the four plane parameters have been determined, we count how many of the other points belong to the same plane (within an accepted maximum distance of 2 mm). The entire process is iteratively repeated N times.

The plane fitting the highest number of points or the first plane that exceeds a sufficient number of fitted points is chosen as floor plane. This basic implementation works well when the surface of the floor is a significant area in the image (Figure 4.3).

In order to make the floor recognition effective also in more complex situations (e.g. where the floor is just a restricted area with respect to the other planar surfaces in the image) we apply an optimisation: by limiting the possible inclination of the sensor (this parameter can be provided manually or through an auxiliary sensor), we can discard some plane orientation *a priori*, because they could not be the floor, but only other surfaces, like walls (Figure 4.4).

Finally, to ensure that the vector $\mathbf{n}$ points upwards with respect to the floor, the sign of parameters a, b, c and d is possibly changed in such a way that the origin of $\mathfrak{R}_{sensor}$ (and consequently all the points above the floor) lies in the upper half-space $ax + by + cz + d > 0$.

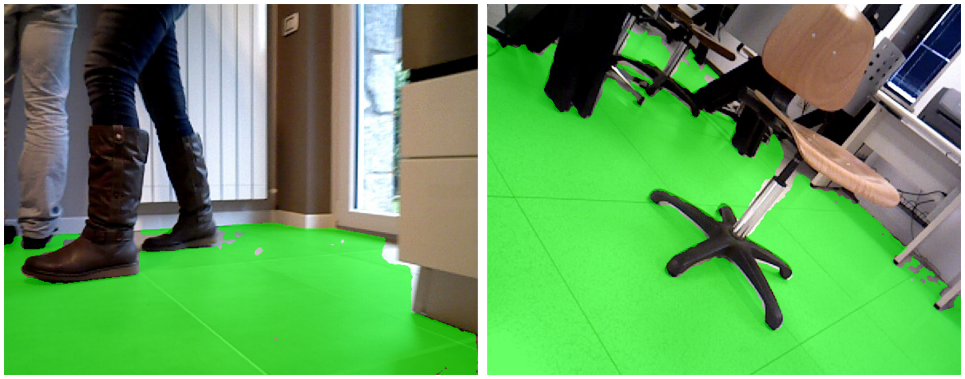


Figure 4.3: Floor detections with sensor in different poses. The points belonging to the detected floor plane are highlighted in green.

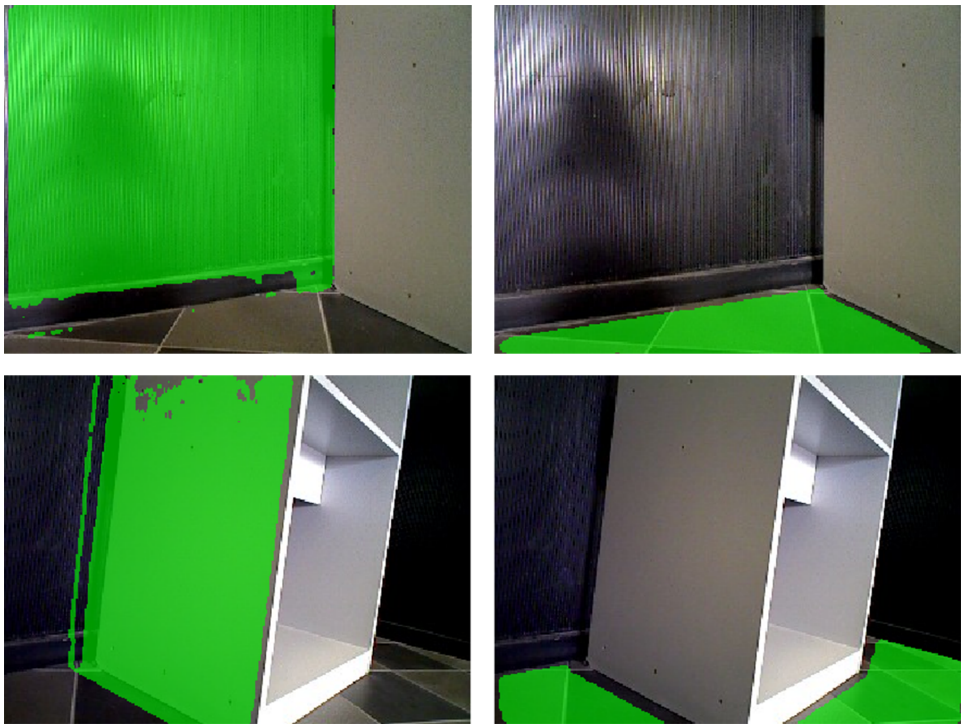


Figure 4.4: Floor detection optimization. On the left, the results obtained with the basic implementation of the algorithm, which makes no restriction on the floor plane orientation. On the right, the improvements obtained with the optimized version, which is able to discard other planar surfaces also when the floor plane occupies a small portion of the image (the maximum inclination of the camera was set to 30 degrees w.r.t. the floor plane).

4.4 Point Cloud Transformation

Using the equation of the estimated floor plane, the whole point cloud (that up to now is expressed with respect to the sensor reference system $\mathfrak{R}_{sensor}$

shown in Figure 4.1) can be rigidly transformed in such a way that the floor lies onto the plane $z = 0$, the origin coincides with the orthogonal projection of $\mathfrak{R}_{sensor}$ origin onto the floor plane, the z-axis points upwards, the y-axis is directed along the projection of the optical axis onto the floor plane and the x-axis is determined to complete a right-handed reference frame, obtaining what we call the world reference system $\mathfrak{R}_{world}$ (Figure 4.5).

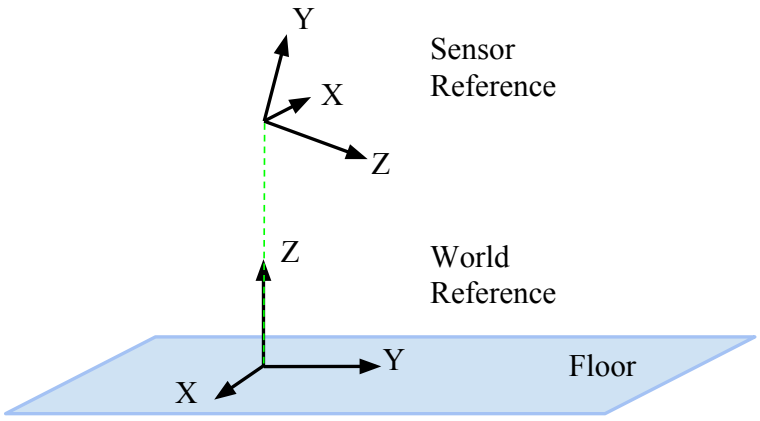


Figure 4.5: Frame of reference change. The point cloud, originally expressed w.r.t. the sensor reference system $\mathfrak{R}_{sensor}$, is transformed to be expressed w.r.t the world reference system $\mathfrak{R}_{world}$.

Such a transformation can be achieved by applying, in order, the following three basic rotations and one final translation.

- Rotation R_1 along z-axis of the sensor till the x-axis is parallel to the floor plane. This transformation is intended to compensate a possible roll of the sensor (see Figure 4.1), whose value is an angle between $-90°$ and $+90°$ (otherwise the sensor is being used upside down, but as said before, in such a situation it is sufficient to properly rotate the input RGB and Depth images of an angle of 180°).
- Rotation R_2 still along the z-axis of the sensor of 180° to align the versus of the new x-axis to the x-axis in $\mathfrak{R}_{world}$.
- Rotation R_3 along the new x-axis to align the z-axis to the normal vector $\mathbf{n}$ of the floor plane, also considering that the sensor could have a non-zero pitch angle (see Figure 4.1).
- Translation $\mathbf{t}_z = (0, 0, h)$ along the new z-axis (which now is aligned to z-axis in $\mathfrak{R}_{world}$) of an amount h equal to the height of the sensor over the floor, which can be computed as the distance between the origin of $\mathfrak{R}_{sensor}$ and the floor plane.

Given a point $\mathbf{p}$ in $\mathfrak{R}_{sensor}$, the corresponding point $\mathbf{p}'$ in $\mathfrak{R}_{world}$ has coordinates:

$$\mathbf{p}' = R_3 R_2 R_1 \mathbf{p} + \mathbf{t}_z \tag{4.4}$$

This change of reference system allows a more convenient representation of the point cloud and greatly simplifies the processing in the successive steps. For instance, the height of the points above the floor is a feature used in various phases of the algorithm and in $\mathfrak{R}_{world}$ it is simply the z-coordinate.

4.5 Candidates Selection

Candidates are identified as point clusters emerging from the floor (Figure 4.6(a)): in particular, we initially limit the analysis to 3D points within the first 20 cm above the floor and apply a density-based 3D clustering technique.

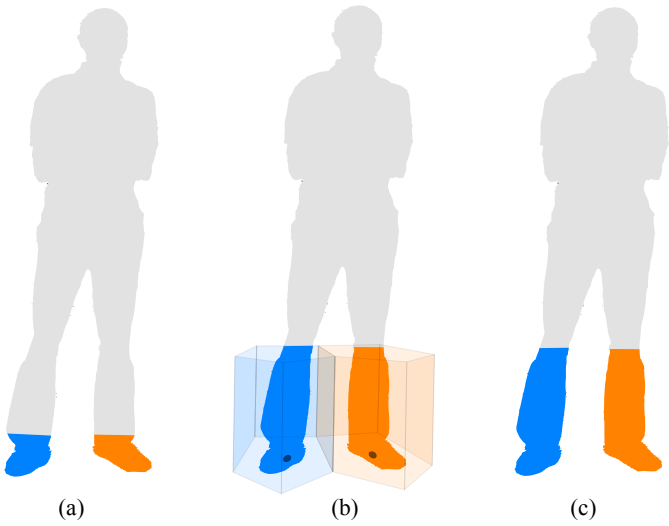


Figure 4.6: Point clusters within the first 20 cm above the floor are identified as candidate feet (a). The centroids of the clusters are projected onto the floor plane and are used as generators of a Voronoi diagram. Each candidate foot is then expanded to a candidate leg, by constraining its expansion up to the knee level and only to the points whose vertical projections fall within the corresponding Voronoi region (b). The resulting clusters are kept as candidate legs (c).

This family of clustering algorithms defines clusters as areas of higher density than the remainder of the data set. The point cloud section we select to apply this search is generally characterized by few sparse objects in large areas: these clear variations in point density perfectly meet the

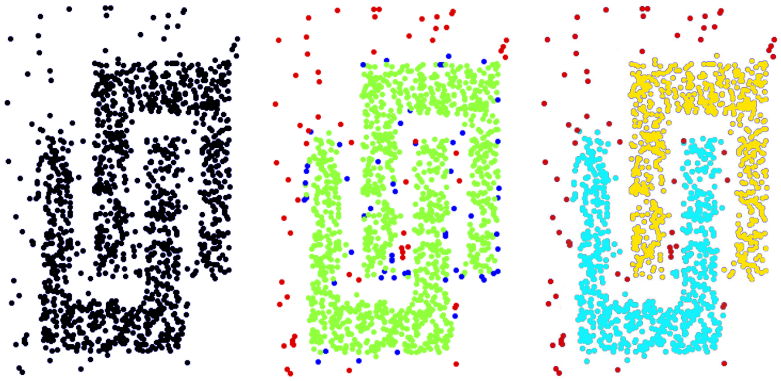
Algorithm 1 DBSCAN

```
function DBSCAN(pointCloud, radius, nNeigh)
    C ← 0
    for each unvisited point P in pointCloud do
        mark P as visited
        neighbors ← regionQuery(P, radius)
        if #neighbors == 1 then
            mark P as OUTLIER
        else if #neighbors > nNeigh then
            mark P as CORE POINT
            C ← C + 1
            expandCluster(P, neighbors, C, radius, nNeigh)
function expandCluster(P, neighbors, C, radius, nNeigh)
    add P to cluster C
    for each point P' in neighbors do
        if P' is not visited then
            mark P' as visited
            add P' to cluster C
            neighbors' ← regionQuery(P', radius)
            if #neighbors' > nNeigh then
                mark P as CORE POINT
                neighbors ← neighbors ∪ neighbors'
            else
                mark P as BORDER POINT
function regionQuery(P, radius)
    return all points within P's radius-neighborhood (including P)
```

requirements of DBSCAN [20], which is described in Algorithm 1 along with an explanatory picture in Figure 4.7.

The method requires two parameters: a radius delimiting the neighborhood extent and the minimum number of neighbors required to form a cluster. Density-based clustering methods greatly benefit from a relatively homogeneous density in the input point cloud: in our case, this is ensured by the voxel-grid downsampling preprocessing step presented in Section 4.2. Indeed this simplifies the setup of the parameters, which we empirically tuned to be a minimum of 5 neighbors in a 4 cm radius, having verified they perform well in very different situations.

In practice, this approach may occasionally lead to cluster together two feet, expecially when the person is standing still (conversely, while a person



(a) (b) (c)

Figure 4.7: DBSCAN algorithm applied on 2D input points plotted in Figure (a). Figure (b) highlights the detection of core (green), border (blue) and outlier (red) points. Figure (c) shows the two identified clusters, together with the outlier points (red) that are discarded.

is walking its feet are usually well-spaced and would not be merged): we accept this possibility, and let the subsequent machine learning and tracking approaches handle this case.

The dimensions of the bounding box of each cluster horizontal projection are used as a simple criterion to pre-filter candidates which are obviously too large or too small to represent either an actual single foot or a joint couple of feet.

4.6 Expansion of Candidates to Candidate Legs

The resulting 3D point clusters are limited to a maximum height of 20 cm, and therefore contain little geometric information for discriminating actual feet from other objects; attempting to exploit such data alone would incur in the same issues observed with 2D laser-scans, as many objects would appear similar to feet. Therefore, we now expand candidates up to knee-height (50 cm), thus incorporating useful information about the characterizing features of a person's leg, like the ankle shape and the almost cylindrical surface of the limb. Further expansion would lead to issues with subjects close to the sensor, where anything above the knee would fall outside of the sensor's field of view.

Each cluster is expanded with a density-based method inspired to DBSCAN, described in Algorithm 2. The neighborhood radius and the number of neighbors parameters have the same meaning and values as presented in

Algorithm 2 CANDIDATE EXPANSION

function EXPANDCANDIDATE($candidatePoints$, $radius$, $nNeigh$)
  $neighbors \leftarrow candidatePoints$
  for each point P in $neighbors$ **do**
    if P is not visited **then**
      mark P as visited
      $neighbors' \leftarrow$ VORONOIREGIONQUERY(P, $radius$)
      if $\#neighbors' > nNeigh$ **then**
        $candidatePoints \leftarrow candidatePoints \cup neighbors'$
        $neighbors \leftarrow neighbors \cup neighbors'$
function VORONOIREGIONQUERY(P, $radius$)
  return all points within P's $radius$-neighborhood (including P)
    AND in the Voronoi region of the candidate

Section 4.5.

In order to avoid adding too many unrelated points to a candidate leg, which would inevitably occur in case of very close clusters and may lead to an erroneous classification, the expansion of each candidate foot is spatially bounded. In particular, we consider the projection on the floor plane of the centroids of all candidates and for each we compute the 2D Voronoi region, which is the set of points closer to that centroid than to any other (see Figure 4.8). For a given candidate, the expansion only considers 3D points whose projection on the floor falls within the respective Voronoi region (Figure 4.6(b)).

Figure 4.9 shows candidate feet and their expansion on real sensor data.

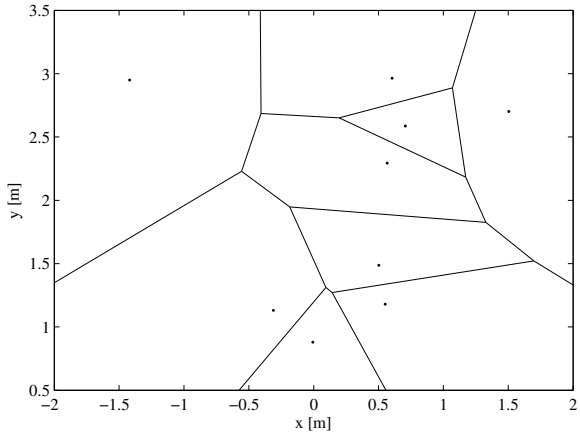


Figure 4.8: Example of a 2D Voronoi Diagram generated by the candidate feet centroids projected on floor (black points).

(a)

(b)

Figure 4.9: Candidate feet clusters (a) and their expansion (b) into candidate legs.

Chapter 5

Candidate Legs Classification

The candidate legs detected by the previous phase of the algorithm may be either real human legs or any other object present in the environment. In order to assign to each candidate a probability of being a true leg, a supervised machine learning approach is adopted. For each candidate 3D shape, a *rectified depth image* is computed and used to build a feature vector for classification by means of a statistical SVM classifier [30] (Figure 5.1).

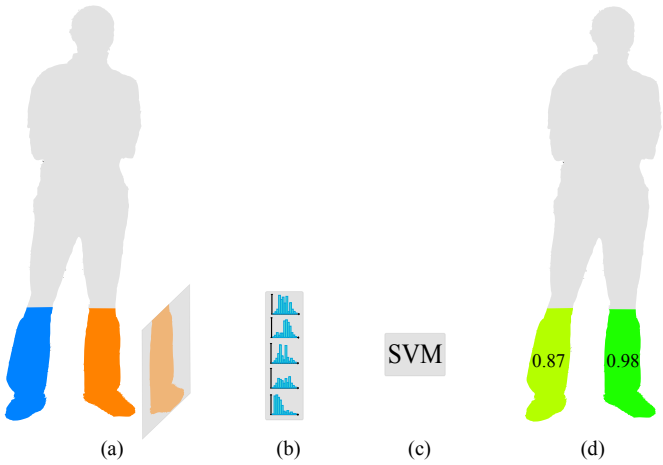


Figure 5.1: The Rectified Depth Image is computed for each candidate leg (a) and from it the Histogram of Oriented Gradients feature vector is extracted (b). The latter is classified (c) by a trained SVM and the corresponding leg probability is obtained (d).

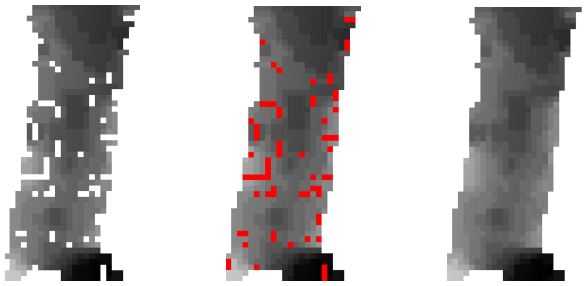
5.1 Rectified Depth Image Computation

The rectified depth image for a given candidate is computed by an orthographic projection of its 3D points onto a vertical plane (Figure 5.1(a)

and 5.3). Note that, unlike the depth image directly acquired by the sensor, the rectified depth image is not affected by the sensor pitch and roll (which may not be exactly horizontal), nor by perspective distortions (which would make farther objects appear smaller); therefore, it is expected to yield a standardized representation of the candidate appearance, robust to variations in the sensor position on the robot.

Depth values are computed on a 40×50 cm grid with cell-size of 1 cm. In order to maximize the number of visible points of the candidate, thus minimizing the number of discretization collisions, we select the vertical projection plane to be the one tangent to the candidate surface and orthogonal to the line connecting the 2D centroid of the candidate (projected onto the floor plane) to the origin of the world reference system (see Figure 4.5).

A mild morphological filtering is then applied to regularize the rectified depth image, filling the cells that miss a depth information due to a lower local density of points. The application of *dilate* and *erode* operators allows to locate such cells, which are set to a value equal to the average of their neighbors non-null depths (Figure 5.2).



(a) (b) (c)

Figure 5.2: Effects of morphological filtering applied to the rectified depth image (shown with grayscale representation). The benefit of this filtering is particularly evident for candidates located far from the sensor, which have generally a lower point density that leads to miss depth information for several internal cells (a). These "holes" (red pixels) are identified through the application of dilate *and* erode *operators (b) and each one of them is filled with the avarage of its neighbors non-null depth values.*

5.2 Feature Computation

We summarize the rectified depth image for a candidate by means of the Histogram of Oriented Gradients [18] descriptor (HOG): this allows us to robustly characterize local 3D shape and appearance of an object by representing local depth changes in simple histograms (Figure 5.1(b)). A similar

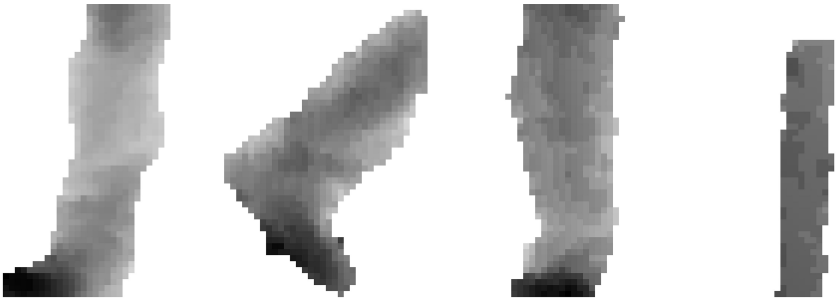


Figure 5.3: Grayscale representation of the rectified depth image for four leg candidates, the last of which is a non-leg object.

descriptor is the Histogram of Depths, which was also adopted for people detection in RGB-D data [52].

Initially, the gradient vector is computed for each cell by means of a convolution of the rectified depth image with the following derivative filtering kernels:

$$dx = [-1, 0, 1] \quad dy = [-1, 0, 1]^T \tag{5.1}$$

Then, the image is divided in a 4×5 grid of square windows with an edge of 10 cells and for each window a 9-bin histogram summarizing local depth changes is computed. Each histogram bin accounts for an interval of gradient orientations and has an height equal to the sum of the magnitudes of the gradient vectors having directions in that range.

When computing the HOG descriptor on an RGB image, it is a good practise to normalize more local histograms together, considering a larger region of the image, called block. This normalization results in better invariance to changes in illumination or shadowing. Since depth images are not affected by such problems, we normalize each window on its own. At this point, in each histogram, the sum of all the bins heigths equals to one.

Another difference with respect to a classic HOG implementation for RGB images, is that while the latter is tested to perform better when considering "unsigned"(0°-180°) gradients [18], we verified "signed"(0°-360°) gradients to achieve slightly better results in our case, since also the information about the versus of the gradient helps to properly characterize the object surface in the space.

As result of all the previous choices, a 180-dimensional feature vector is extracted from the rectified depth image, yielding a good trade-off between classification accuracy and feature dimensions (in Section 8.3.1 we report quantitive comparisons).

5.3 SVM Classification

Support Vector Machine (SVM [30]) is a supervised learning model commonly used for binary classification. Given some training data $\mathcal{D}$, a set of n samples in the form:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^m, \quad y_i \in \{-1, 1\}\}_{i=1}^{n} \tag{5.2}$$

where y_i represents the class of the m-dimensional point $\mathbf{x}_i$, the SVM classifier searches for the hyperplane that best separates the two classes of points. Such a hyperplane, called *maximum margin hyperplane*, is the one with the largest distance to the nearest training data point of any class (Figure 5.4), since, typically, the larger the margin, the lower is the generalization error of the classifier.

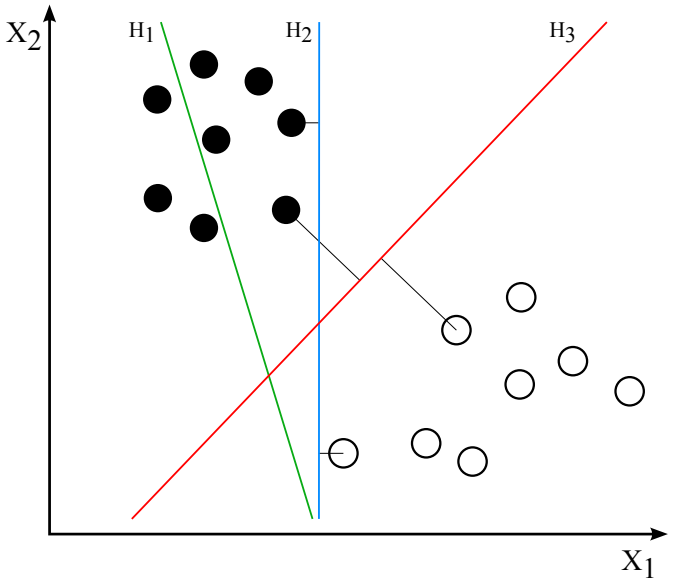


Figure 5.4: SVM classifier on 2D space. H_1 does not separate the classes. H_2 does, but only with a small margin. H_3 separates them with the maximum margin and therefore is choosen as the maximum margin hyperplane.

Any hyperplane can be written as the set of points $\mathbf{x}$ satisfying the equation:

$$\mathbf{w} \cdot \mathbf{x} - b = 0 \tag{5.3}$$

and by geometric considerations it can be proved that the margin to maximize is inversely proportional to $\|\mathbf{w}\|$. Thus, it can be formulated as a quadratic programming optimization problem:

$$\arg\min_{(\mathbf{w},b)} \frac{1}{2}\|\mathbf{w}\|^2 \tag{5.4}$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1. \qquad \forall i = 1, \ldots, n \tag{5.5}$$

If there exists no hyperplane that can split correctly the train data $\mathcal{D}$, the *Soft Margin* method will choose a hyperplane that splits the samples as cleanly as possible. In order to do so the method introduces non-negative slack variables ξ_i, which measure the degree of misclassification of the data x_i. The optimization problem becomes:

$$\arg\min_{\mathbf{w},\xi,b} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \right\} \tag{5.6}$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \qquad \forall i = 1, \ldots, n \tag{5.7}$$

The output of a binary SVM classifier, as described so far, is the class $y \in \{-1, 1\}$ which the input point $\mathbf{x} \in \mathbb{R}^m$ belongs to. In many contests, it is more useful to have the probabilities p_k of belonging to a certain class $k \in \{-1, 1\}$, rather than the discrete class label. It can be noticed that a good indicator of the classification confidence of the point $\mathbf{x}$ is the decision value $\hat{f}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} - b$: the sign of $\hat{f}$ indicates the predicted class, while the higher the absolute value $|\hat{f}|$ the more reliable is the classification.

A common method to extend the SVM to a *statistical* SVM [39] is by applying a logistic regression on the decision values computed from the train data. More specifically, the goal is to fit a logistic function on the data $\hat{\mathcal{D}}$, obtained from the training data $\mathcal{D}$:

$$\hat{\mathcal{D}} = \{(\hat{f}(\mathbf{x}_i), \hat{y}_i) \;\mid\; (\mathbf{x}_i, y_i) \in \mathcal{D}, \;\; \hat{y}_i = max(0, y_i)\}_{i=1}^{n} \tag{5.8}$$

The probability p_1 of a point $\mathbf{x}$ to belong to the class $k = 1$ is then approximated by the value:

$$p_1 = \frac{1}{1 + e^{A\hat{f}(\mathbf{x})+B}} \tag{5.9}$$

where A and B are estimated by minimizing the negative log likelihood of data $\hat{\mathcal{D}}$. The probability p_{-1} of a point $\mathbf{x}$ to belong to the class $k = -1$ is trivially given by $p_{-1} = 1 - p_1$.

In many real applications, a linear model (hyperplane) is too weak to obtain a good separation between the training points belonging to the different classes. SVM can be extendend to create non-linear classifiers by applying the *kernel trick*. The resulting algorithm is formally similar, except that every dot product is replaced by a non-linear kernel function. This allows the

algorithm to fit the maximum margin hyperplane in a *transformed* feature space. The transformation may be non-linear and the *transformed* space high dimensional; therefore, even though the classifier is a hyperplane in the *transformed* feature space, it may be non-linear in the *original* input space (Figure 5.5).

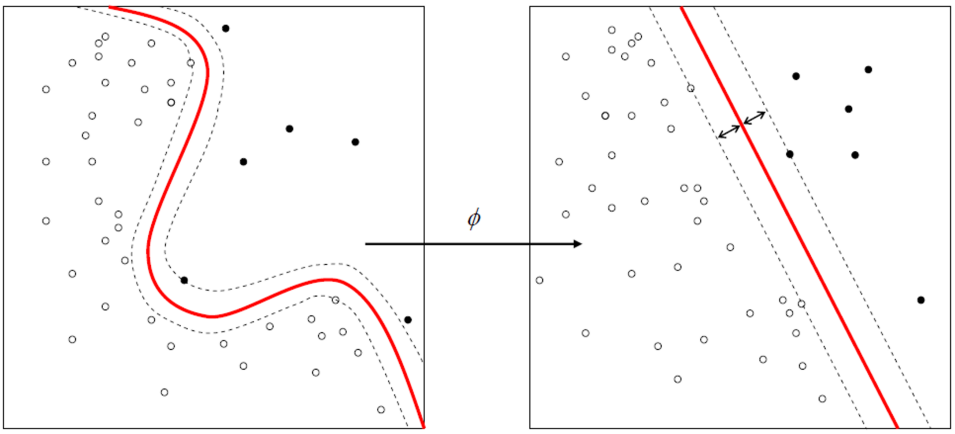


Figure 5.5: Illustrative example of non-linear SVM classification. The original *feature space (on the left) is mapped by means of the transformation ϕ to the* transformed *feature space (on the right), where a linear separation of the two classes is possible. In practice, the* kernel trick *allows to find the maximum margin hyperplane in the* transformed *feature space without having to explicitly define ϕ.*

A common choice is to use the Gaussian radial basis function (RBF) as kernel k:

$$k(\mathbf{p}, \mathbf{q}) = \exp(-\gamma \|\mathbf{p} - \mathbf{q}\|^2) \qquad \gamma > 0 \qquad \mathbf{p}, \mathbf{q} \in \mathbb{R}^m \tag{5.10}$$

For the purpose of our work, we select a statistical, soft margin, non-linear SVM with an RBF kernel.

The effectiveness of such SVM depends on the selection of the kernel parameter γ and the soft margin parameter C. The best combination is selected through a grid search as the one which optimizes the 5-fold cross validation accuracy. The final model, which is used for testing and classifying new data, is then trained on the whole training set, using the selected parameters.

The resulting trained SVM receives in input the feature vector described in Section 5.2 and returns as output the corresponding probability of being a human leg (Figure 5.1(c,d)).

In Chapter 8 we will also show some comparative results between our classifier and other binary classifiers, highlighting the reasons that led us to our particular choice.

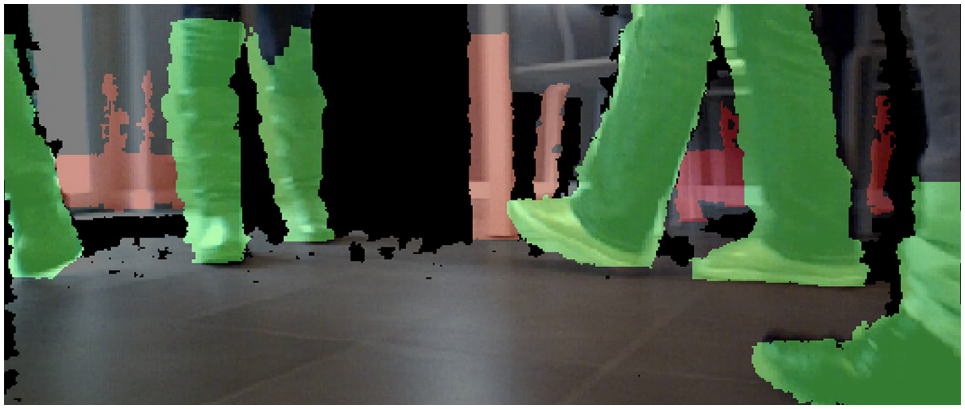


Figure 5.6: Visualization of the results of candidate classification: green and red correspond respectively to high and low leg probability.

After the classification phase, a set of *leg observations*, consisting of the leg candidates 3D points together with their leg probability, is available for further processing. In Figure 5.6 we show the leg observations in the point cloud and in Figure 5.7 we provide a concise representation of their 2D positions on the floor plane.

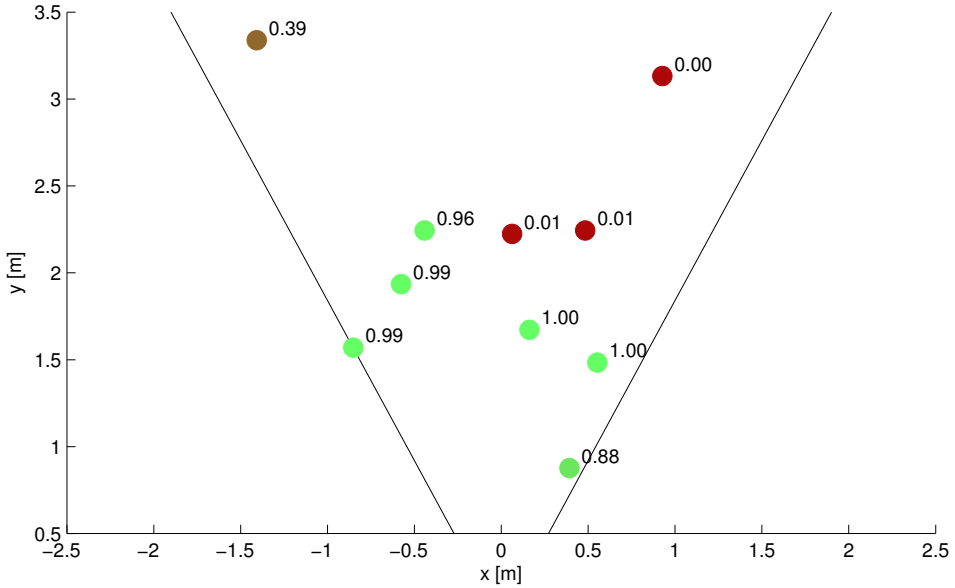


Figure 5.7: Representation of legs observations on the floor plane. The leg observation centroids (circles) are plotted onto the plane $z = 0$ of the world reference system $\mathfrak{R}_{world}$, which corresponds to the floor plane and on which it is highlighted the horizontal field of view of the sensor. The colors range from red to green depending on the leg probability (reported next to the circle) assigned by the SVM classifier.

Chapter 6

Tracking

People are detected and tracked from leg observations computed for each frame by means of a two-step approach: in the first step, individual leg observations are used to robustly detect and track legs in time, obtaining a set of *leg tracks*; in the second step, leg tracks are used as observations in a filter which yields a set of *person tracks*, solving challenging data association issues.

6.1 Dynamic Model

Our tracking algorithm is built as an extension of the Kalman filter, that is a filtering technique that uses a series of noisy measurements observed over time and produces estimates of unknown variables that tend to be more precise than those based on single measurements alone.

The Kalman filter method is based on linear dynamic systems discretized in the time domain. The state of the system is represented as a n-dimensional vector of real numbers. At each discrete time increment, a linear operator is applied to the state to generate the new state, with some noise mixed in. Then, another linear operator mixed with more noise generates the q observable outputs from the system state.

For the purpose of our work, the linear dynamic system can be assumed to be time invariant and not affected by external control. The model with which the state $\mathbf{x}_k \in \mathbb{R}^n$ at time k is evolved from the state $\mathbf{x}_{k-1} \in \mathbb{R}^n$ at time $k-1$ is:

$$\mathbf{x}_k = F\mathbf{x}_{k-1} + \mathbf{w}_{k-1} \tag{6.1}$$

where $F \in \mathbb{R}^{n \times n}$ is the *state transition model* and $\mathbf{w}_{k-1} \in \mathbb{R}^n$ is the process noise which is assumed to be indipendent and identically distributed Gaussian white noise, with covariance $Q \in \mathbb{R}^{n \times n}$ (i.i.d. $\mathbf{w}_{k-1} \sim \mathcal{N}(0, Q)$).

At time k an observation (or measurement) $\mathbf{z}_k \in \mathbb{R}^q$ of the state $\mathbf{x}_k \in \mathbb{R}^n$ is made according to:

$$\mathbf{z}_k = H\mathbf{x}_k + \mathbf{v}_k \tag{6.2}$$

where $H \in \mathbb{R}^{q \times n}$ is the *observation model* which maps the state space into the observed space and $\mathbf{v}_k \in \mathbb{R}^q$ is the observation noise which is assumed to be indipendent and identically distributed Gaussian white noise, with covariance $R \in \mathbb{R}^{q \times q}$ (i.i.d. $\mathbf{v}_k \sim \mathcal{N}(0, R)$).

In order to better clarify the concepts just introduced, we present a simple example for modelling the 1D motion of a point with a linear dynamic system (extracted from [11, pp. 272–274]) . We choose this example because an extension of this is implemented in our tracking algorithm.

The state $\mathbf{x}_k$ of the moving point is composed by its 1D position x_k and its velocity $\dot{x}_k$. The assumption is that the point undergoes a small acceleration which remains constant during each time step interval (of length ΔT). The resulting *nearly constant velocity* state transition model can then be formulated as:

$$\mathbf{x}_k = F\mathbf{x}_{k-1} + \mathbf{\Gamma} a_{k-1} \tag{6.3}$$

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix} \quad F = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} \quad \mathbf{\Gamma} = \begin{bmatrix} \frac{1}{2}\Delta T^2 \\ \Delta T \end{bmatrix} \tag{6.4}$$

$$Q = \mathrm{E}[\mathbf{\Gamma} a_{k-1} a_{k-1} \mathbf{\Gamma}^T] = \mathbf{\Gamma}\sigma_a^2\mathbf{\Gamma}^T = \begin{bmatrix} \frac{1}{4}\Delta T^4 & \frac{1}{2}\Delta T^3 \\ \frac{1}{2}\Delta T^3 & \Delta T^2 \end{bmatrix} \sigma_a^2 \tag{6.5}$$

having decomposed the process noise $\mathbf{w}_{k-1} = \mathbf{\Gamma} a_{k-1}$, where a_{k-1} is a zero-mean Gaussian white noise (i.i.d. $a_{k-1} \sim \mathcal{N}(0, \sigma_a^2)$) representing small accelerations and $\mathbf{\Gamma}$ is a vector which transfers the acceleration a_{k-1} to the consequent changes in position and velocity during the current time step.

Assuming now that at each time step a noisy measurement of the true position x_k of the point is available (while we can not know its true velocity $\dot{x}_k$), the Equation 6.2 is defined by:

$$\mathbf{z}_k \in \mathbb{R} \qquad H = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad v_{k-1} \sim \mathcal{N}(0, \sigma_z^2) \tag{6.6}$$

6.2 Kalman Filter

The Kalman filter [56] is a recursive estimator that aims to provide an estimate $\hat{\mathbf{x}}_{k|k}$ of the system state at time k given observations up to time k (included) and a covariance matrix $P_{k|k}$ that measures the estimate accuracy.

Typically, the filtering is implemented with two alternating phases: *predict* and *update*.

The *predict* phase uses the state estimate $\hat{\mathbf{x}}_{k-1|k-1}$ from the previous time step to produce a prediction of the state $\hat{\mathbf{x}}_{k|k-1}$ at the current time step, also computing the corresponding prediction covariance $P_{k|k-1}$:

$$\hat{\mathbf{x}}_{k|k-1} = F\hat{\mathbf{x}}_{k-1|k-1} \tag{6.7}$$

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q \tag{6.8}$$

In the *update* phase, the current prediction $\hat{\mathbf{x}}_{k|k-1}$ is combined with the current observation $\mathbf{z}_k \in \mathbb{R}^q$ to refine the state estimate $\hat{\mathbf{x}}_{k|k}$ and the corresponding estimate covariance $P_{k|k}$:

$$\hat{\mathbf{z}}_{k|k-1} = H\hat{\mathbf{x}}_{k|k-1} \tag{6.9}$$

$$\widetilde{\mathbf{y}}_k = \mathbf{z}_k - \hat{\mathbf{z}}_{k|k-1} \tag{6.10}$$

$$S_k = HP_{k|k-1}H^T + R \tag{6.11}$$

$$K_k = P_{k|k-1}H^T S_k^{-1} \tag{6.12}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + K_k\widetilde{\mathbf{y}}_k \tag{6.13}$$

$$P_{k|k} = P_{k|k-1} - K_k H P_{k|k-1} \tag{6.14}$$

$\widetilde{\mathbf{y}}_k$ is called *innovation* and provides a measure of how much the state prediction $\hat{\mathbf{x}}_{k|k-1}$ differs from the system state $\mathbf{x}_k$, by comparing the observation $\mathbf{z}_k$ from the system with the predicted observation $\hat{\mathbf{z}}_{k|k-1}$. K_k is called *Kalman gain* and it is proved to minimize the mean square error $\mathrm{E}[\|\mathbf{x}_k - \hat{\mathbf{x}}_{k|k}\|^2]$.

6.3 Probabilistic Data Association Filter

In many real applications, at a certain time step k several observations may be available (i.e. there is not a single $\mathbf{z}_k \in \mathbb{R}^q$, but there is a set $\mathcal{M}_k$ of measures $\mathbf{z}_k^i \in \mathbb{R}^q$) and there might be uncertainty about which one is originated by the system being tracked. The basic Kalman filter does not deal with this situation, but there exists a well-known extension that tackles this data association problem: the Probabilistic Data Association Filter (PDAF [10]). The two main aspects it introduces are the *measurement validation* and the *combined innovation*. The *measurement validation* aims to discard the observations $\mathbf{z}_k^i$ that differ too much from the predicted observation $\hat{\mathbf{z}}_{k|k-1}$. In order to do so, it selects a subset $\mathcal{V}_k \subseteq \mathcal{M}_k$ of observations which fall inside an elliptical region (also called gate) centered in the predicted observation:

$$\mathcal{V}_k(\gamma) = \{\mathbf{z}_k^i \in \mathcal{M}_k \mid (\mathbf{z}_k^i - \hat{\mathbf{z}}_{k|k-1})^T S_k^{-1}(\mathbf{z}_k^i - \hat{\mathbf{z}}_{k|k-1}) \leq \gamma\} \tag{6.15}$$

where γ is the parameter that determines the region extent. The $m_k = |\mathcal{V}_k(\gamma)|$ validated observations are then used for the computation of the *combined innovation* $\widetilde{\boldsymbol{\nu}}_k$, which replaces the standard Kalman *innovation* $\widetilde{\mathbf{y}}_k$ with an average of the *innovations* $\widetilde{\mathbf{y}}_k^i$ of each observation:

$$\widetilde{\boldsymbol{\nu}}_k = \sum_{i=1}^{m_k} \beta_k^i \widetilde{\mathbf{y}}_k^i \tag{6.16}$$

where the weights β_k^i are computed as:

$$\beta_k^i = \begin{cases} \frac{\mathcal{L}_k^i}{1 - P_D P_G + \sum_{j=1}^{m_k} \mathcal{L}_k^j} & i = 1, ..., m_k \\ \\ \frac{1 - P_D P_G}{1 - P_D P_G + \sum_{j=1}^{m_k} \mathcal{L}_k^j} & i = 0 \end{cases} \tag{6.17}$$

P_D is the target detection probability (that accounts for the possibility that none of the m_k observations is generated by the target), P_G is the probability that the target measure has been inserted amongst the m_k measures and

$$\mathcal{L}_k^i := \frac{\mathcal{N}[\mathbf{z}_k^i; \hat{\mathbf{z}}_{k|k-1}, S_k] P_D}{\lambda} \tag{6.18}$$

The value of λ represents an estimate of the spatial density of observations and can be obtained as the ratio between the number of validated observations m_k and the volume of the gate region. The notation $\mathcal{N}[\mathbf{z}_k^i; \hat{\mathbf{z}}_{k|k-1}, S_k]$ indicates the value of a multivariate Gaussian probability density function with mean equal to $\hat{\mathbf{z}}_{k|k-1}$ and covariance S_k in correspondence of $\mathbf{z}_k^i$:

$$\mathcal{N}[\mathbf{z}_k^i; \hat{\mathbf{z}}_{k|k-1}, S_k] = \frac{1}{\sqrt{(2\pi)^q det(S)}} e^{-\frac{1}{2}(\mathbf{z}_k^i - \hat{\mathbf{z}}_{k|k-1})^T S_k^{-1} (\mathbf{z}_k^i - \hat{\mathbf{z}}_{k|k-1})} \tag{6.19}$$

Considering all these extensions to the stardard Kalman Filter, in the PDAF recursive estimator the Equations 6.7, 6.8, 6.9, 6.10, 6.11, 6.12 are still valid, while the Equations 6.13, 6.14 change as follows. The state estimate $\hat{\mathbf{x}}_{k|k}$ considers the *combined innovation* of validated observations:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + K_k \widetilde{\boldsymbol{\nu}}_k \tag{6.20}$$

For the update of the state estimate covariance $P_{k|k}$, besides considering a contribution from all the validated observations, it is also taken into account the possibility that none of them has been generated by the target, with probability β_k^0 (defined in Equation 6.17):

$$P_{k|k}^c = P_{k|k-1} - K_k S_k K_k^T \tag{6.21}$$

$$\widetilde{P}_k = K_k \left(\sum_{i=1}^{m_k} \beta_k^i \widetilde{\mathbf{y}}_k^i \widetilde{\mathbf{y}}_k^{iT} - \widetilde{\boldsymbol{\nu}}_k \widetilde{\boldsymbol{\nu}}_k^T \right) K_k^T \tag{6.22}$$

$$P_{k|k} = \beta_k^0 P_{k|k-1} + (1 - \beta_k^0) P_{k|k}^c + \widetilde{P}_k \tag{6.23}$$

6.4 Legs Tracking

Tracking leg observations across frames is not trivial, because temporary occlusions of the legs (by foreground objects, the other leg, or other people) are very frequent.

The system state $\mathbf{x}_k$ for each leg observation consists of the 2D position (x_k, y_k) and velocity $(\dot{x}_k, \dot{y}_k)$ of the leg centroid projected onto the floor plane, as well as its probability p_k of being a leg. The *update step* is performed by associating each leg observation to the closest leg track and using the cluster centroid and the leg probability returned by the classifier as measurements. A *nearly constant velocity* model (introduced in Section 6.1) is assumed for each leg track, considering random gaussian noise to account for a variable acceleration (in practice, legs have a very irregular movement pattern). With reference to the Equations 6.1, 6.3, 6.4, 6.5, the resulting *state transition model* is composed in the following way:

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ \dot{x}_k \\ y_k \\ \dot{y}_k \\ p_k \end{bmatrix} \qquad F = \begin{bmatrix} 1 & \Delta T & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta T & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.24}$$

$$Q = \begin{bmatrix} \frac{1}{4}\Delta T^4 & \frac{1}{2}\Delta T^3 & 0 & 0 & 0 \\ \frac{1}{2}\Delta T^3 & \Delta T^2 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4}\Delta T^4 & \frac{1}{2}\Delta T^3 & 0 \\ 0 & 0 & \frac{1}{2}\Delta T^3 & \Delta T^2 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \sigma_a^2 \tag{6.25}$$

Note that for position and velocity the system is a trivial 2D extension of the 1D model presented in Section 6.1 (considering independent the positions and velocities along the two coordinates), whereas the leg probability is introduced as a further state variable which is not affected by noise during state transition.

With reference to Equation 6.2, the *observation model* is defined as:

$$\mathbf{z}_k \in \mathbb{R}^3 \qquad H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} \sigma_z^2 & 0 & 0 \\ 0 & \sigma_z^2 & 0 \\ 0 & 0 & \sigma_p^2 \end{bmatrix} \tag{6.26}$$

where σ_z^2 and σ_p^2 correspond to the error variances affecting the measurements of, respectively, the position (both x and y) and the leg probability of a leg observation.

Tracking multiple objects in cluttered environments is known to be a very challenging task. In fact, a set of measurements $\mathbf{z}_k^i$ is available at time step k and this makes it difficult to associate a measure to the correct target. Moreover the targets may often be very close to each other and interfere. Several sophisticated algorithms (most notably JPDAF [10], GMPHD [55], MHT [13], and multi-target particle filters [46]) have been proposed to deal with these complex scenarios. We have observed satisfactory results with a simpler and more efficient approach, which accounts for some particular features of the problem at hand.

All leg observations are used as measurements in our filtering, including those which were classified to have a low leg probability. In fact, partially occluded legs are frequently assigned a low leg probability by the classifier but in this way it's possible to limit the classification errors which may arise from the highly variable appearances of occluded objects, both for true legs and other obstacles. The integration in time of leg observations, indeed, allows to maintain in the leg tracks a more correct leg probability also when the 3D candidates shape, due to possible partial occlusions, does not appear in its neat form to be properly classified.

Because the robot may be moving or rotating while sensing, tracking must be performed using a fixed reference frame, not the robot's own: in particular, leg observations are converted from the moving world reference frame to a fixed world reference frame by exploiting the robot's odometry information. In this way the person's velocity is correctly estimated regardless of the robot's own motion.

In order to associate leg observations to leg tracks, we perform a variant of nearest neighbor data association. In particular, we project on floor the foot points of the leg observation and compute their minimum euclidean distance to the leg track centroid. When this minimum distance is under a small threshold (5 cm), the leg observation is directly associated to that leg track and a classic Kalman Filter update step is performed. If a leg observation can be directly associated to two distinct leg tracks, revealing a consistent possibility that two close legs previously tracked separately have been merged by the clustering phase into a unique leg observation, then both leg tracks are updated, each with a measure corresponding to an estimate of its new centroid.

Whenever a direct association is possible, leg tracks are also marked as *visible*: this indicates that they are in the sensor field of view, not occluded by foreground objects, and that the corresponding leg observations are available and may be used in the subsequent people tracking phase as further information.

Algorithm 3 LEGS TRACKING

function LEGSTRACKING(*legObservations*)
  for each t in $LegTracks$ **do** ▷ Predict phase
    apply Kalman predict phase to t
  for each *observation* in *legObservations* **do** ▷ Direct associations
    $footProj \leftarrow$ {foot points of *observation* projected on floor}
    $t1,\ t2 \leftarrow$ the two tracks in $LegTracks$ closest to $footProj$
    if DIST($t1$, $footProj$)$< 5cm$ and DIST($t2$, $footProj$)$< 5cm$ **then**
      $observation1,\ obervation2 \leftarrow$ split merged *observation*
      apply Kalman update to $t1$ with $observation1$
      apply Kalman update to $t2$ with $observation2$
      mark $t1$ and $t2$ as updated
      mark $t1$ and $t2$ as visible
      mark *observation* as associated
    else if DIST($t1$, $footProj$)$< 5cm$ **then**
      apply Kalman update to $t1$ with *observation*
      mark $t1$ as updated
      mark $t1$ as visible
      mark *observation* as associated
  $legObservations \leftarrow \{obs \in legObservations \mid obs$ not associated $\}$
  for each t not updated in $LegTracks$ **do** ▷ PDAF update
    $validatedObs \leftarrow \{obs \in legObservations \mid$ DIST(t, obs)$< 15cm\}$
    mark each *observation* in $validatedObs$ as validated
    apply PDAF update to t with $validatedObs$
  $legObservations \leftarrow \{obs \in legObservations \mid obs$ not validated $\}$
  for each *obs* in *legObservations* **do** ▷ New track creation
    create a new track t for *obs*
    add t to $LegTracks$

On the other hand, leg tracks not directly associated to a leg observation are updated through a PDAF approach (see Section 6.3), accounting for all observations within 15 cm from the track centroid.

Finally, in the case a leg observation is far (more than 15 cm) from any existing track, a new track is created.

A high level description of how the legs tracking is performed using the legs observations available at each frame is presented in Algorithm 3.

Note that only leg tracks with a leg probability larger than a threshold τ are used for the subsequent level of tracking. τ represents an important parameter of the system, whose impact is evaluated in Section 8.

In Figure 6.1 we provide an explanatory representation of the output of the legs tracking algorithm.

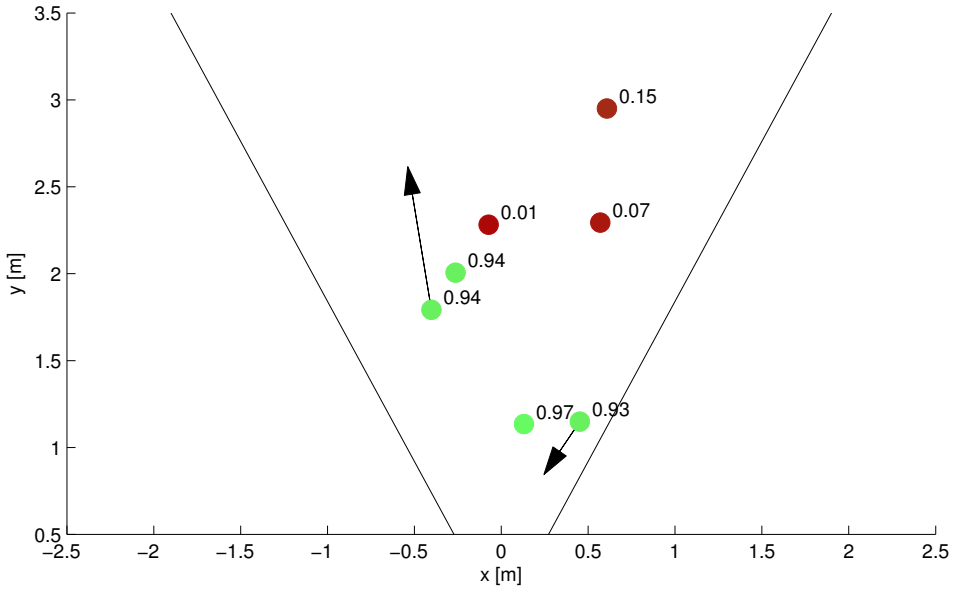


Figure 6.1: Representation of legs tracking output. The leg track centroids (circles) are plotted onto the plane $z = 0$ of the world reference system $\mathfrak{R}_{world}$, which corresponds to the floor plane and on which it is highlighted the horizontal field of view of the sensor. The colors range from red to green depending on the track $leg\ probability$, which is also reported next to the circle. Moreover, the arrows, when present, indicate the velocities of the corresponding leg track.

6.5 Legs Pairing

In order to achieve a more accurate tracking of people in the scene (which will be detailed in Section 6.6), it's necessary to identify, at each time step, the pairs of leg tracks which very likely belong to the same person. This problem can be formalized with a graph model.

Consider an undirected weighted graph $G = (V, E)$, where the set of vertices V comprises all the legs and the set of edges E associate legs which are likely to belong to the same person. The weights assigned to each edge in E represent a *distance* between the connected legs: the smaller the weight value (and so the shorter the *distance*), the higher is the probability that the legs should be paired. The choice of the function defining the *distance* metric is very critical for legs pairing and different features (e.g. colors, shapes, relative positions, etc.) could be considered for measuring how "close" two

legs are.

The procedure to build such a graph is the following.

- Add a vertex in V for each leg.
- Define a *distance* function $\Delta : V \times V \to \mathbb{R}^+$, which given a pair of legs computes their distance in terms of belonging to the same person.
- Determine a threshold δ, representing the highest *distance* between two legs to consider the possibility of pairing them.
- Add an edge in E for each pairs of vertices $v1, v2 \in V,\ v1 \neq v2$ such that $\Delta(v1, v2) \leq \delta$, and $\Delta(v1, v2)$ is the weight assigned to that edge.

In the ideal case (with a perfect distance funcion Δ), $\Delta(v1, v2) = 0$ if and only if $v1$ and $v2$ represent the two legs of the same person. In such situation, selecting a threshold $\delta = 0$ allows to perfectly discriminate right from wrong legs pairings.

Examples of possible resulting legs pairing graphs are shown in Figures 6.2 and 6.3

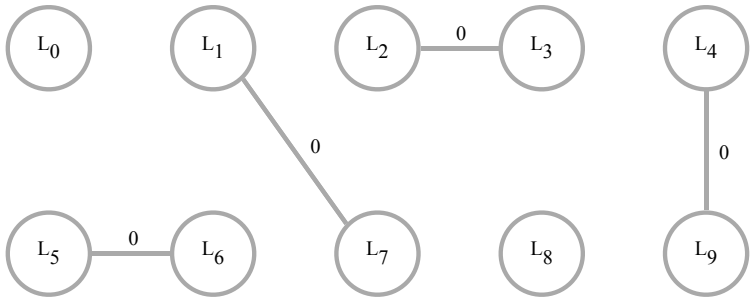


Figure 6.2: Example of a legs pairing graph in an ideal case. The ideal Δ function is able to perfectly recognize only the actual pairs of legs belonging to the same person. Two people show only one of their legs (respectively L_0 and L_8) and they are correctly recognized as isolated vertex in the graph. The optimal solution of the legs pairing ploblem, thus, consists in all the edges of the pairing graph.

In the obtained graph we may identify different connected components $G_i = (V_i, E_i)$ with order $|V_i| = n_i$. In the ideal case, with a perfect distance function Δ, each subgraph G_i has order $n_i = 1$ or $n_i = 2$: $n_i = 1$ corresponds to the case in which only one leg of a person is present and should not be paired to any other leg; $n_i = 2$ corresponds to the case in which two legs of the same person are present and should be correctly paired (see Figure 6.2).

This ideal scenario might not always be the case in real situations (see Figure 6.3), due to possible errors in legs detection and mostly due to the

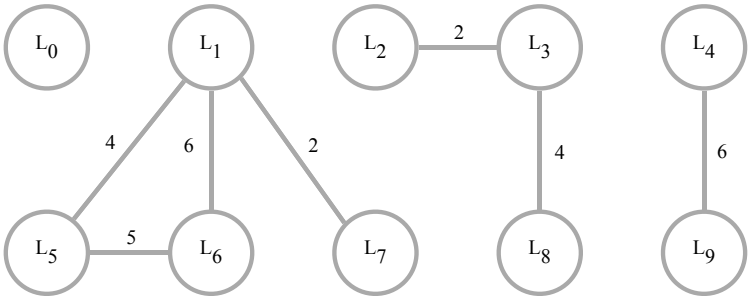


Figure 6.3: Example of a legs pairing graph in a real case. Given that all the legs are correctly detected, the real Δ function is not always able to recognize only the actual pairs of legs and more than two legs (also actual single legs) may be detected as "close" to each other, for any threshold δ that doesn't miss any real pair of legs. This results in different connected components with order greater than 2 and finding an optimal solution for them is not always straightforward.

difficulty in defining a good distance function Δ. When the order of a component G_i is greater than 2 ($n_i > 2$), a feasible solution of the *legs pairing* problem for that component consists in a subset of edges $A_i \subset E_i$ such that $|A_i| = \lfloor \frac{n_i}{2} \rfloor$ and each vertex in V_i is the endpoint of at most one edge in A_i. An optimal solution of the problem is a feasible solution that minimizes the sum of the weights of the selected edges in A_i and can be discovered by means of a *branch and bound* search algorithm in the space of feasible solutions. Figure 6.4 shows an optimal solution for a non-ideal legs pairing graph.

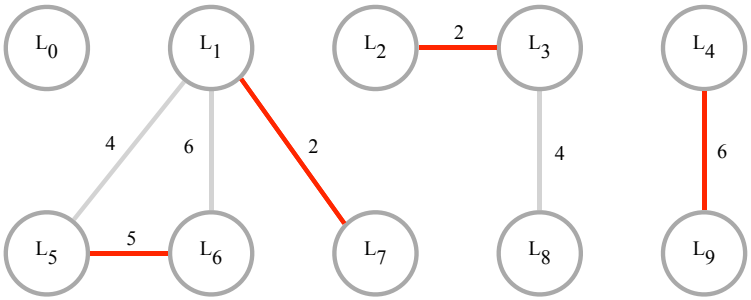


Figure 6.4: An optimal solution for a legs pairing graph in a real case. The optimal solution (highlighted in red) combines the optimal solutions of each connected component, which can be found by means of a branch and bound search algorithm in the space of their feasible solutions.

In our system, we implemented a Δ function which exploits the property that the two legs of the same person merge at the hip level. Therefore, the

3D point cluster of each leg observation is expanded upwards, up to the hip level (using the same density based approach described in Section 4.6). Then, given two leg observations L_1 and L_2 with their corresponding sets of expanded 3D points E_1 and E_2, we define the Δ function as:

$$\Delta(L_1, L_2) := min\{\|P_1 - P_2\| \ \mid \ P_1 \in E_1 \ and \ P_2 \in E_2\} \tag{6.27}$$

Our Δ function is characterized by a high specificity and a low sensitivity: in fact, it allows to have a high precision on detected pairs of legs, particularly when selecting a small threshold δ, but could miss many leg pairs, expecially for subjects close to the sensor (whose hip area falls outside the sensor field of view). We demonstrate these results in Section 8.3.2. Therefore we cannot rely only on the legs pairs detected by this just presented procedure, rather we use it to simplify and improve the people tracking phase, as it will be described in detail in the next section.

6.6 People Tracking

The second tracking step uses leg tracks as observations and returns a set of person tracks, which is the final output of the system.

In particular: the *state variables* tracked for each person consist of the 2D position of its barycenter and its velocity; a *nearly constant velocity* model (introduced in Section 6.1) is assumed as motion model; the *measurements* used for updating the state are derived from leg tracks as described below. The linear dynamic model can be easily derived from the one presented in Equations 6.24, 6.25 and 6.26, by discarding all the tracking aspects related to the leg probability p_k.

In crowded scenarios, correctly associating each leg track to the correct person track is not straightforward: we adopt the following geometric approach.

- First, considering only the leg tracks marked as *visible* by the previous level of tracking (i.e., the ones for which a leg observation has been directly associated), we search for pairs of them which most likely belong to the same person by finding a solution to the *legs pairing* problem described in Section 6.5, using our presented Δ function. Since exploring all the space of the feasible solutions to find the optimal one can be computationally expensive, we adopt a greedy approch to find a suboptimal solution that will be shown in Section 8.3.2 to be quite close to the optimum in the general case. If a couple of clusters is selected as a legs pair, we check whether their barycenter can be directly

associated to an existing person track using a nearest-neighbor policy (within a maximum distance of 8 cm). If this is verified, a classic Kalman state update is applied on the person track, and the two leg tracks are removed from further processing.

- For the remaining leg tracks, we adopt a probabilistic approach to the legs pairing problem. We create a list of all possible pairings consisting of all leg track pairs whose mutual distance is less than 80 cm (note that the same leg track may appear in more than one of such pairs). The centroid of each pair is computed and added to a list of *potential person barycenters*. Additionally, the centroid of each individual leg is also included in the same list, in order to account for the frequent case in which a single leg is visible for a person. For each person track yet to be updated, a PDAF approach is performed by accounting for all *potential person barycenters* within a 40 cm radius from the barycenter of the person track. In this way we can exploit the state information tracked over time for each person to compute the likelihood (by means of the PDAF weights β_k^i) that each *potential person barycenter* corresponds to the actual person barycenter, indirectly solving the legs pairing problem.

A critical aspect turns out to be the creation of a new person track. Our system adopts a conservative strategy, creating a new person only when two leg tracks have high classification probabilities, the upward expansions of the associated clusters nearly merge (i.e. they are detected as a legs pair by our legs pairing algorithm with a strict threshold δ), and their barycenter is far enough from the barycenter of existing person tracks. Consequently, single leg tracks far from any known person track are assumed to belong to a yet untracked person whose second leg has not been observed yet.

A high level description of how the people tracking is performed using the leg tracks (and leg observations) available at each frame is presented in Algorithm 4.

Due to the highly variable appearances of legs, the computation of the leg centroids and consequently the person barycenter (used as measures in the filtering procedures) may result inaccurate. In particular, this problem affects the estimated velocity of the person barycenter and makes it quite unstable and noisy over time. In order to solve this issue and provide a more useful information, our system reports a moving average of the Kalman-based estimates of person barycenter velocity over a 1 second temporal window.

Algorithm 4 PEOPLE TRACKING

function PEOPLETRACKING($allLegTracks$, $legObservations$)
  $legTracks \leftarrow \{l \in allLegTracks \mid$ the leg probability of $l > \tau\}$
  $legPairs \leftarrow \{(l1,\ l2) \mid l1, l2 \in legTracks, l1 \neq l2,$ DIST$(l1, l2) < 80cm\}$
  for each t in $PeopleTracks$ **do** ▷ Predict phase
    apply Kalman predict phase to t
  for each pair $(l1,\ l2) \in legPairs$ **do** ▷ Direct associations
    if $l1$ or $l2$ is not visible **then**
      continue to next pair $(l1,\ l2)$
    $obs1 \leftarrow$leg observation in $legObservations$ directly associated to $l1$
    $obs2 \leftarrow$leg observation in $legObservations$ directly associated to $l2$
    EXPANDTOHIP($obs1$)
    EXPANDTOHIP($obs2$)
    $\delta \leftarrow 5cm$
    if $\Delta(obs1, obs2) \leq \delta$ **then**
      mark $(l1, l2)$ as joining legs
      $barycenter \leftarrow$ BARYCENTER$(l1,\ l2)$
      $t \leftarrow$ track in $PeopleTracks$ closest to $barycenter$
      if DIST$(t,\ barycenter) < 8cm$ **then**
        apply Kalman update to t with $barycenter$
        mark t as updated
        remove all pairs in $legPairs$ containing either $l1$ or $l2$
        remove $l1$ and $l2$ from $legTracks$
  $possibleBarys \leftarrow \{$ BARYCENTER$(l1,\ l2) \mid (l1, l2) \in legPairs\}$
  $possibleBarys \leftarrow possibleBarys \cup \{$centroids of $legTracks\}$
  for each t not updated in $PeopleTracks$ **do** ▷ PDAF update
    $validatedBarys \leftarrow \{b \in possibleBarys \mid$ DIST$(t,\ b) < 40cm\}$
    mark each barycenter in $validatedBarys$ as validated
    apply PDAF update to t with $validatedBarys$
    mark t as updated
  $legPairs \leftarrow legPairs \setminus \{(l1, l2) \mid$ BARYCENTER$(l1,\ l2)$ was validated$\}$
  for each $(l1,\ l2) \in legPairs$ **do** ▷ New track creation
    if $(l1,\ l2)$ was marked as joining legs **then**
      create a new track t for BARYCENTER$(l1,\ l2)$
      add t to $PeopleTracks$
      remove all pairs in $legPairs$ containing either $l1$ or $l2$
      remove $l1$ and $l2$ from $legTracks$
  remaining $l \in legTracks$ are single legs of yet untracked people

The final output of our system is then a list of people present in the environment, each one characterized by:

- *ID*: a unique identification number of the person;
- *barycenter position*: the 2D coordinates of the estimated person barycenter on the floor plane, expressed in the world reference system relative to the sensor;
- *barycenter velocity*: the 2D velocity of the person barycenter on the floor plane, expressed in the absolute world reference system (when odometry is available, otherwise in the reference system relative to the sensor);
- *legs positions, velocities and visibilities*: the 2D positions and velocities of legs (if any) associated to the person and their *visibility*, that indicates for each leg whether it was in the sensor field of view at the current frame.

A concise explanatory representation of the output of the people tracking algorithm is shown in Figure 6.5

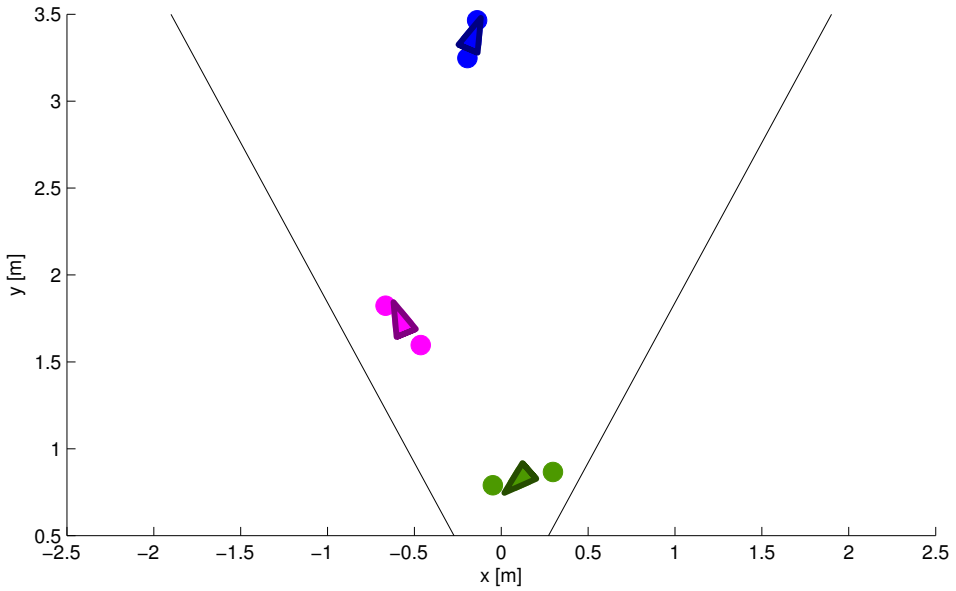


Figure 6.5: Representation of people tracking output. The legs centroids (circles) and the people barycenters (triangles) are plotted onto the plane $z = 0$ of the world reference system $\mathfrak{R}_{world}$, which corresponds to the floor plane and on which it is highlighted the horizontal field of view of the sensor. Different colors are used for different people and triangles headings consider the corresponding person moving direction.

Chapter 7

System Implementation and Dataset Acquisition

7.1 Implementation

The prototype system is implemented in MATLAB, with the most computationally expensive tasks written as *MEX* functions able to exploit multi-core CPUs thanks to OpenMP support. Various ROS-Matlab bridges exist [40] which allow to use our prototype within ROS: in particular, it appears as a node listening for RGB-D data and robot odometry, and publishing the positions and velocities of tracked people. Alternatively, the prototype can use the OpenNI library for accessing to live or recorded sensor data.

An overview of the system input/output interface is presented in Figure 7.1.

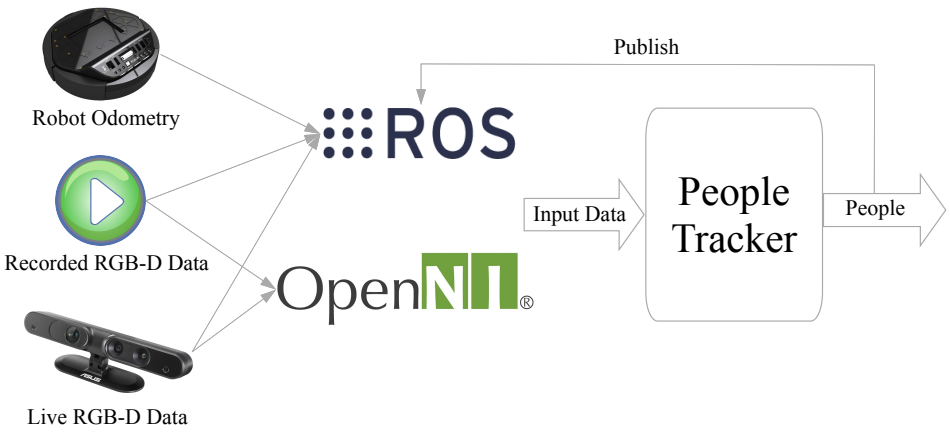


Figure 7.1: Overview of the system input/output interface.

We now present in more details our People Tracker system. Figure 7.2 shows a data flow diagram highlighting the various implemented modules, the data structures on which they operate and the relations intercurring among them.

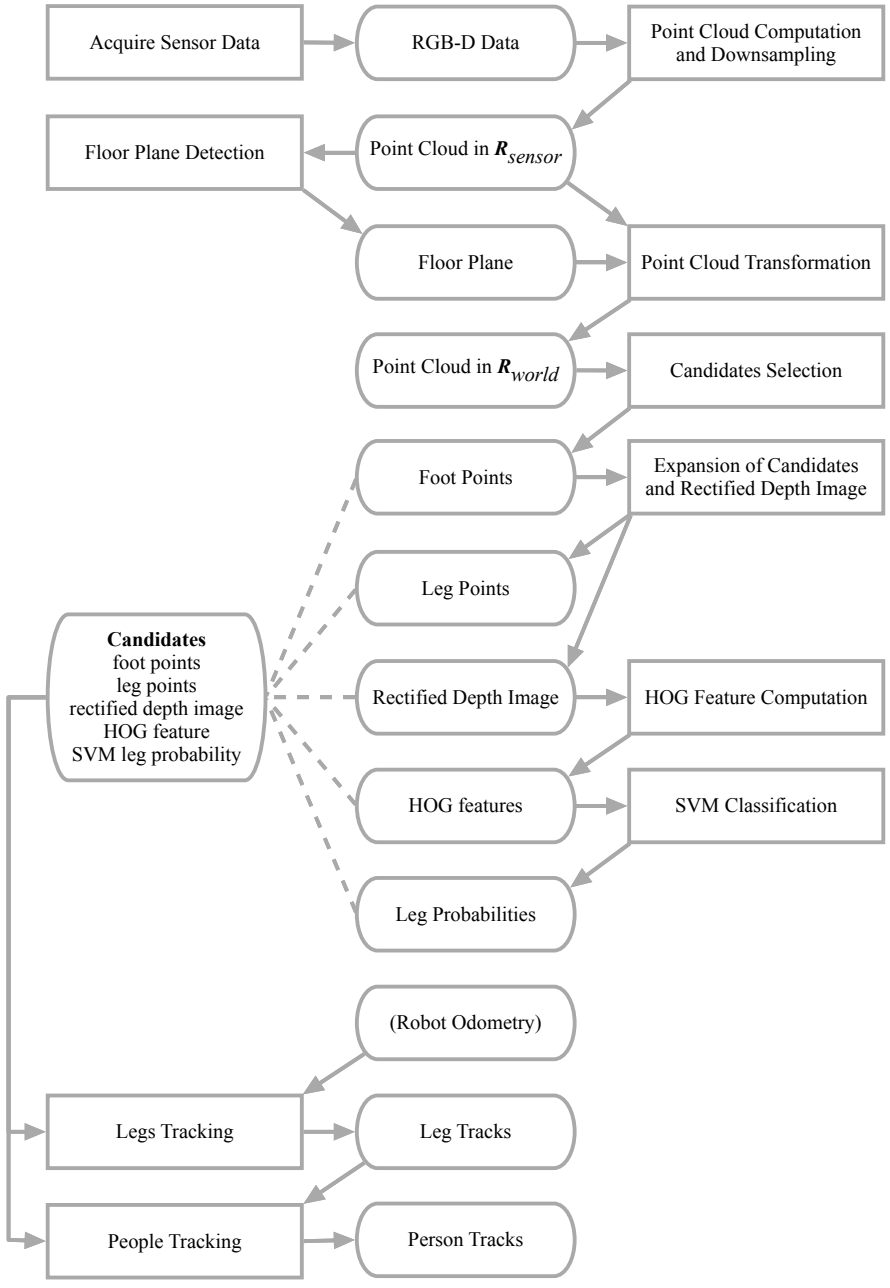


Figure 7.2: Data flow diagram of the implemented system. Rectangles correspond to modules, while rounded rectangles correspond to data structures.

The *Acquire Sensor Data* module integrates both OpenNI library and ROS-Matlab bridge to read RGB-D data from the two possible input interfaces.

The *SVM classification* module is entirely based on the open-source, freely available C library *LIBSVM* [17].

The release of our system implementation is available on-line at `bit.ly/perceivingpeople`.

7.2 Dataset Acquisition and Classifier Training

The *training dataset* for the SVM classifier is acquired as follows. We recorded two set of videos: the first set was shot in 15 different cluttered indoor environments without any person appearing in the frame; the second set was shot by placing the sensor in several wide indoor halls, recording only passing people. In the first set of videos, all detected leg candidates have been labeled with ground truth class *non-leg*; in the second set, all leg candidates have ground truth class *leg*. The resulting training dataset contains 26000 candidates, of which 35% are legs.

Moreover, *testing datasets* have been created for evaluating the overall performance of the system. We recorded three 30 seconds RGB-D videos in different and completely new environments (i.e., not included in the training dataset) and manually labelled each single frame, by marking every visible leg and indicating which person it belonged to. The testing scenarios are the following:

- **S-Easy**: the sensor is still and two people randomly walk in an obstacle-free environment;
- **S-Medium**: the sensor is still and three people walk around a small table with few other obstacles in the scene;
- **S-Difficult**: the sensor is on a mobile platform in a quite cluttered environment and a total of three people walk around a small table.

The three testing datasets with associated ground truth are also available on-line at `bit.ly/perceivingpeople`, to promote quantitative comparisons with future systems.

7.3 Computational Costs

Depending on the scenario complexity, the system processes 10 to 30 fps on a low-end Intel Core i5-2415M laptop. When using a top-end laptop based

Module (Ref. Section)	S-Easy		S-Medium		S-Difficult	
	ms	%	*ms*	%	*ms*	%
1. *Acquire Sensor Data* (4.1)	13.8	31.7	12.1	17.5	11.7	11.4
2. *Point Cloud Comput.* (4.1) *and Downsampling* (4.2)	3.6	8.3	10.5	15.2	11.7	11.4
3. Floor Plane Detection (4.3)	0.3	0.6	0.3	0.4	0.4	0.4
4. Point Cloud Transform. (4.4)	1.3	3.0	2.4	3.5	3.4	3.3
5. *Candidates Selection* (4.5)	4.2	9.6	11.4	16.4	15.8	15.4
6. *Expansion Candidates* (4.6) *and Rectified Depth Image* (5.1)	5.7	13.2	10.2	14.7	17.1	16.7
7. *HOG Feature Comput.* (5.2)	0.2	0.4	0.3	0.4	0.4	0.4
8. *SVM Classification* (5.3)	5.7	13.2	10.0	14.4	20.3	19.8
9. Legs Tracking (6.4)	3.2	7.4	5.8	8.3	12.4	12.1
10. People Tracking (6.6)	5.4	12.5	6.3	9.1	9.1	8.9

Table 7.1: Computational cost of system modules. The mean processing time required for each module, together with its ratio with respect to the total time, is reported for the three testing scenarios. Such times refer to the average time spent at each frame, when running on an Intel Core i5-2415M laptop. Beside each module name it is reported the reference to the Section where the implemented algorithms are described. Moreover, the module names are italicized if they are implemented as MEX functions.

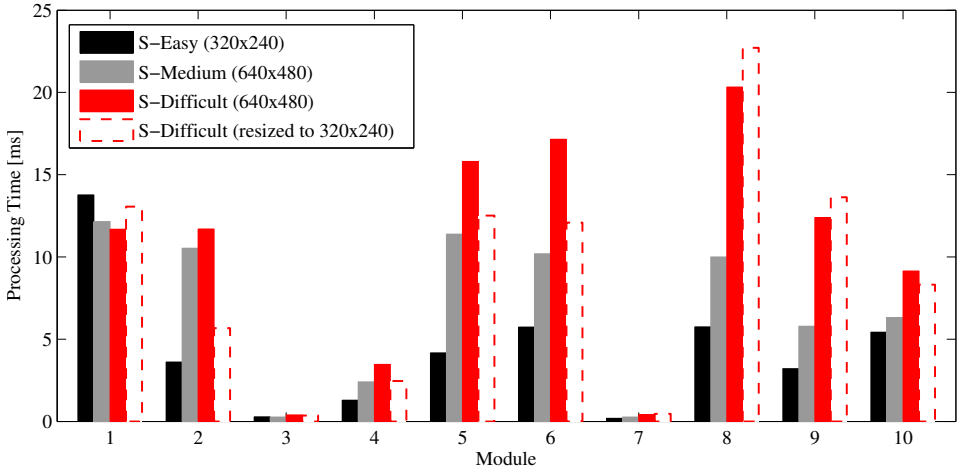


Figure 7.3: Bar Graph showing computational cost of system modules. The numbers on the horizontal axis refer to the modules in the same order they are listed in Table 7.1.

on an Intel Core i7-4930MX processor, the minimum framerate observed in very complex scenarios raises to 25 fps, whereas for most of the time the full frame rate is achieved with less than 40% CPU load. As we measure in Section 8.4, the system performance does not significantly degrade even with as few as 10 fps.

We further analyze the computational cost of each implemented module (presented in Figure 7.2). Table 7.1, along with Figure 7.3, report the average processing times registered for each module when running on the low-end hardware configuration.

As it can be observed, the complexity of the scenario and the consequent high number of candidate legs appearing in the sensor field of view greatly affects all the modules starting from the *Candidate Selection*.

Another factor that could influence the computational time of the system is the sensor resolution. We compare the processing times registered on the *S-Difficult* scenario in both its original resolution (640×480) and in a resized version (320×240). Thanks to the downsampling phase, the effect of the higher resolution is mostly limited to the module *Point Cloud Computation and Downsampling*.

Chapter 8

Experimental Results

8.1 Performance Measures

We tested the performance of our system using the frame-based metrics of *precision (p)*, *recall (r)* – also known as *sensitivity*, *F1 score (f1)* and *specificity (s)* defined as:

$$p = \frac{TP}{TP + FP} \qquad r = \frac{TP}{TP + FN} \qquad f1 = \frac{2pr}{p + r} \qquad s = \frac{TN}{TN + FP} \quad (8.1)$$

where TP is the number of True Positives, FP is the number of False Positives, FN is the number of False Negatives and TN is the number of True Negatives.

We separately evaluated legs detection, legs pairing and people detection.

- **Legs Detection**
  This evaluation is intended to measure the performance in correctly identifying legs in each frame, regardless of the person which they belong to. In this case, a True Positive is a correctly detected leg, a True Negative is any object correctly not reported as leg, a False Positive is a leg detection returned by our system which can not be associated to any leg in the ground truth of the frame, and a False Negative is a leg in the ground truth of the frame for which no detection has been reported.

- **Legs Pairing**
  This evaluation is intended to quantify the performance of different heuristics (i.e., different *distance* functions Δ and solution searching strategies) in correctly associating legs belonging to the same person (see Section 6.5). A True Positive is counted for each correctly identified pair of legs. A False Positive is any reported pair of legs not

belonging to the same person. A False Negative is any pair of legs in the ground truth not detected.

- **People Detection**
  This evaluation is intended to express an overall performance measure of our system, which only accounts for whether people visible in the frame are in fact detected. True Positives are counted for each person in the ground truth for which a person track has been returned within a 25 cm radius. False Negatives are counted for each person in the ground truth for which no person track is found within the same limit. False Positives are counted for each detected person track that cannot be associated to any person in the ground truth.

The three *testing datasets* used for the evaluation of our system have been presented in Section 7.2.

8.2 Alternative Methods For Legs Detection

In order to evaluate how our system compares with alternative methods, we implemented and tested on the same datasets three different approaches to leg detection.

- **Inscribed Angles Variance (IAV [33])**: a 2D laser-based method that classifies a cluster to be a leg based on geometrical relations. In particular, each cluster is first checked to have a quite circular shape by exploiting a geometric property: every point of a circular arc has congruent angles (angles with equal values) with respect to the arc extreme points. The detection of circles is thus achieved by calculating the average and standard deviation of the inscribed angles. Positive detection of circles occurs with standard deviation values below 8.6° and average values between 90° and 135°. If this is the case, the cluster is further checked to match common leg dimensions, by computing its center and its diameter, and verifying that the latter has a value between 10 cm and 25 cm. Only when these conditions are satisfied, the cluster is classified as leg.

- **Supervised Learning on 2D Range Data [8]**: a 2D laser-based method which trains a machine learning classifier on 13 cluster features. Such features range from the number of points in the cluster to the measures of linearity and mean curvature of the shape they describe, as well as the distances from preceding and succeding clusters in the same frame (thus the clusters must first be ordered, e.g.

counter-clockwise). For the detailed list of features, please refer to [8]. We compare results obtained with both Adaboost and SVM classifiers.

- **Histogram of Local Surface Normals (HLSN [26])**: an alternative classification feature extracted from 3D points. In particular, the method consists of estimating the normal vector to the candidate surface in each 3D point (by considering its local neighborhood). The computed normals are then discretized over a 30-bin histogram and given in input to a trained SVM classifier.

For 2D laser-based methods, we synthesize input data by computing a horizontal section of our 3D leg candidates at 30 cm above the floor plane (Figure 8.1).

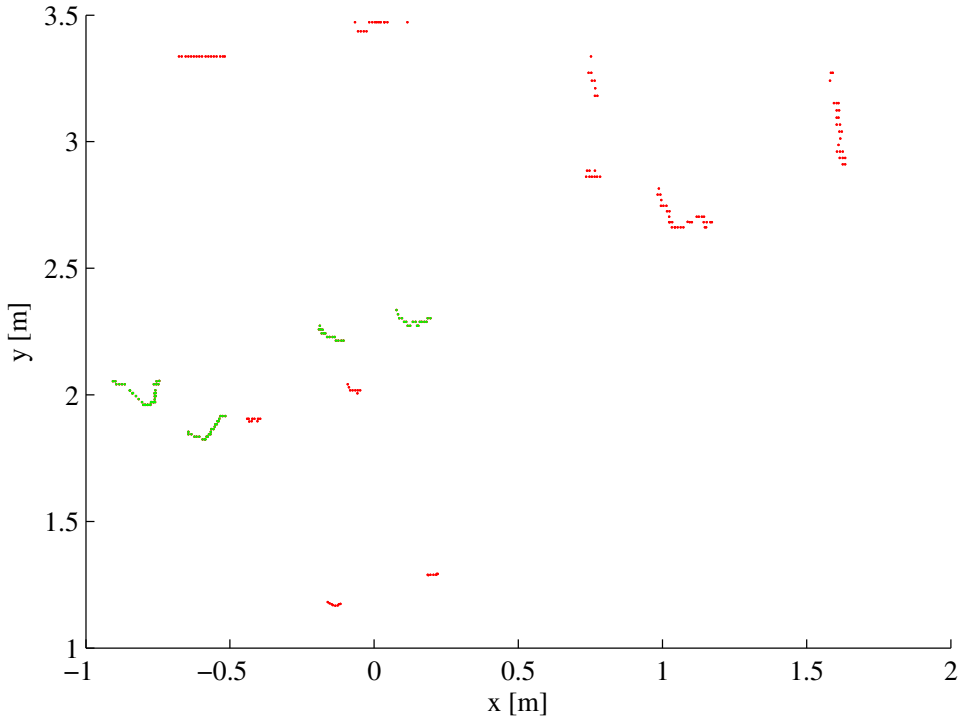


Figure 8.1: Examples of laser leg candidates obtained as a horizontal section of the 3D leg candidates. Specifically, they are obtained through the intersection of 3D candidates with the plane $z = 0.3$ [m] of the reference system $\Re_{world}$. Points belonging to true human legs are plotted in green.

8.3 Quantitative Results

Considering the performance measures and the alternative methods for legs detection presented in Sections 8.1 and 8.2, we evaluate our system at different stages.

8.3.1 Legs Detection

Candidates Selection Performance

In this section we show the performance of the *Candidates Selection* process described in Chapter 4. Each candidate available after this process is considered as a leg detection.

We evaluate it on the three testing scenarios and report the results in Table 8.1.

	Legs Detection		
Scenario	*p*	*r*	*f1*
S-Easy	0.95	0.91	0.93
S-Medium	0.64	0.88	0.74
S-Difficult	0.19	0.89	0.32

Table 8.1: Legs detection after candidates selection process.

The most important metric at this stage is the recall r, since we want the system to select the 3D candidates without loosing any of them corresponding to actual legs (high sensitivity). It can be noticed that its value is around 90% in all the scenarios, taking also into consideration that a part of the missed 10% of actual legs have in fact been merged in a single cluster with another leg and counted only for one True Positive. It will be shown in Section 8.3.1 how the tracking partially solves this problem by splitting the merged clusters and consequently raising the recall value.

Regarding the precision p, it can be observed how it strongly depends on the complexity of the scenarios and proves the necessity, in the general case, of a classifier to discriminate between actual human legs and other objects. Moreover, in the following sections it will be highlighted how it is difficult to reach high level of precision without worsening too much the recall value and that a tradeoff is inevitable.

Candidates Classification Performance

A good classifier for the purpose of our work should clearly discriminate between leg and non-leg classes and, at the same time, keep the computational cost as low as possible, a necessary condition for real time applications.

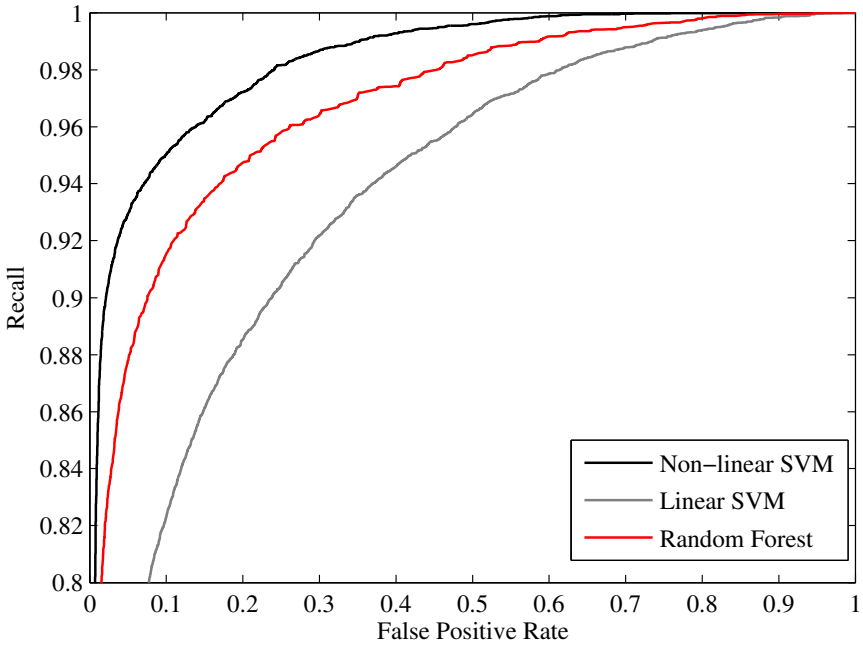
A common technique used to evaluate the goodness of a *statistical* binary classifier is to analyze the Receiver Operating Characteristic (ROC curve). It is a graphical plot showing the values of false positive rate $(1 - s)$ and recall (r) while varying the discrimination threshold. The ideal ROC curve has all its points coinciding with the $(0, 1)$ point: in fact, this means that the corresponding classifier is able to perfectly discriminate between the two classes, always assigning a probability equal to 1 to positive samples and a probability equal to 0 to negative samples. In real cases such a situation never occurs and the more the actual ROC curve approches the ideal ROC curve, the better the classification performance is. Specifically, the most interesting section to analyze is at the *knee* of the ROC curve, where the best tradeoff between the false positive rate $(1 - s)$ and recall (r) is usually found. In addition, a typical summary statistic often used for providing a concise evaluation of the classifier is the area under its ROC curve (AUC).

In order to determine a good classification algorithm for our purpose, we tested three popular common choices.

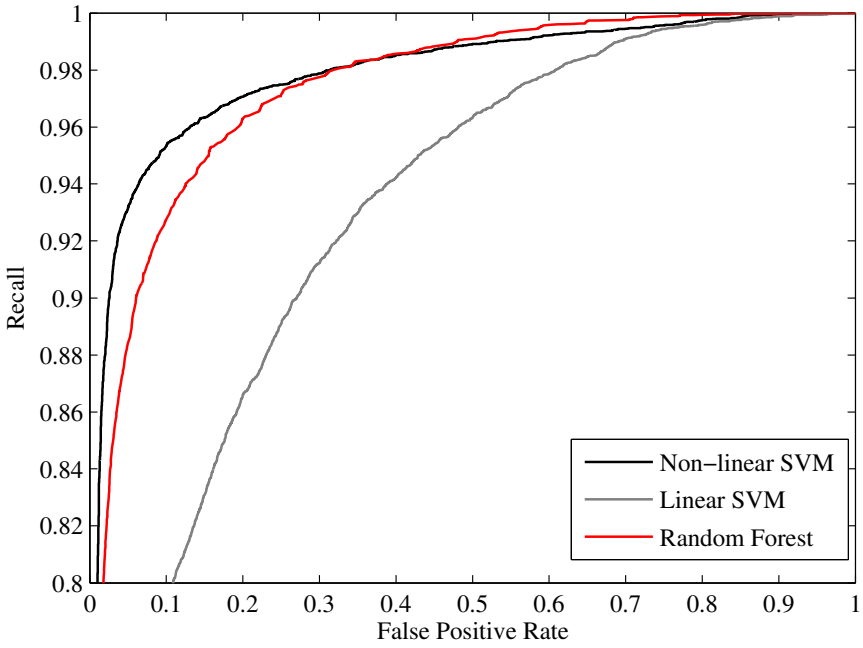
- Non-linear SVM with RBF kernel: the SVM setup is the one presented in Section 5.3.
- Linear SVM: with reference to the introduction to SVM classifiers presented in Section 5.3, this SVM classifier searches for the maximum margin hyperplane in the *original* feature space, without exploiting the *kernel trick*.
- Random Forest [15]: it is a learning method for classification that uses a multitude of decision trees to predict the class of a sample by averaging the output returned by individual trees.

The comparison is achieved by analyzing the ROC curves obtained through a 10-fold cross validation on the *training dataset* and focussing on two different and relevant configurations of the HOG parameters (see Section 5.2). The results are reported in Figure 8.2 and it is possible to notice how the non-linear SVM reaches higher levels of classification performances in both cases. Therefore, this classification algorithm is selected for our system and a more detailed analysis on its classification performance is carried out.

In order to select the best combination of HOG parameters, we perform again a 10-fold cross validation on the *training dataset* with additional HOG

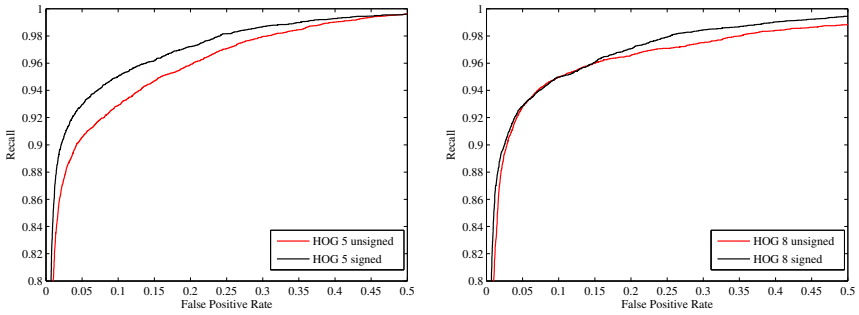


(a) HOG with window size of 5 cm, signed gradient



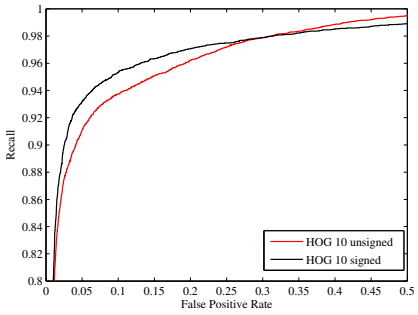
(b) HOG with window size of 10 cm, signed gradient

Figure 8.2: ROC curves of different classification algorithms obtained with two relevant configurations of the HOG parameters.

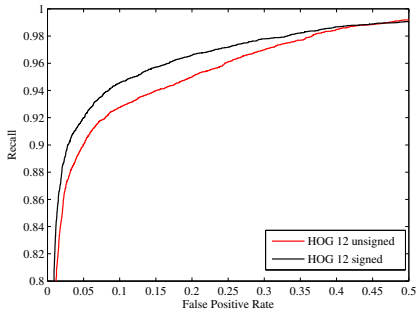


(a) HOG with 5 cm window size

(b) HOG with 8 cm window size



(c) HOG with 10 cm window size



(d) HOG with 12 cm window size

Figure 8.3: ROC curves of the non-linear SVM classifier obtained with different HOG configurations. Each graph highlights the knee *section of the curves, where the best tradeoff between sensitivity and specificity is located, and shows the benefits of using signed gradients (black curves) over unsigned gradients (red curves) with different values of window size.*

configurations, varying the window edge size and the option of using signed or unsigned gradients. In Figure 8.3 we report the resulting ROC curves, highlighting the *knee* section of the curves, where the best tradeoff between sensitivity and specificity is located. As it can be observed, here the signed gradients configurations generally outperform the unsigned ones, and so the latter ones are discarded from further consideration.

Regarding the window size parameter, it significantly affects the feature vector dimension and, consequently, the overall processing time; therefore, in order to choose its optimal value we also consider the mean time required for classifying a candidate. Figure 8.4 offers a comparison among the ROC curves obtained with the different choices of window size, having fixed the option of using signed oriented gradients. The curves, actually, slightly differ from each other, as well as their AUC values. Considering Figure 8.4 along with Table 8.2, we select a window size of 10 cm as a good overall tradeoff, obtaining a final ROC curve with an AUC value of 0.977.

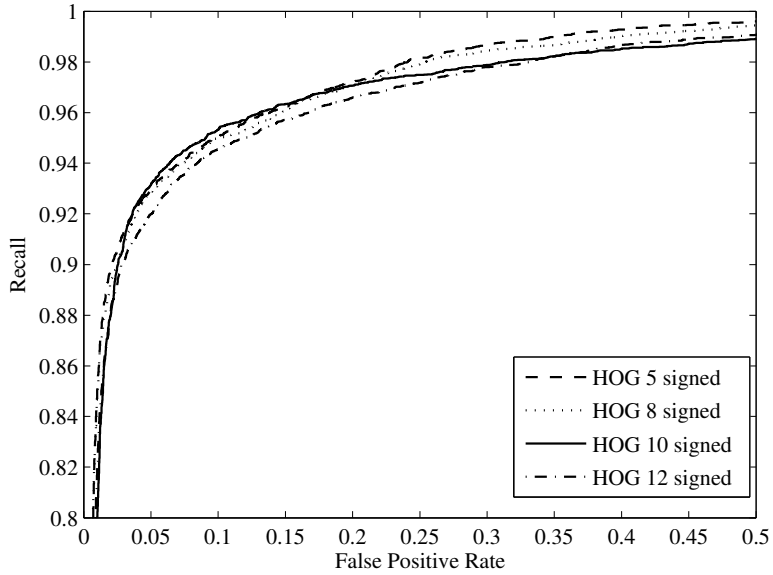


Figure 8.4: ROC curves of the non-linear SVM classifier obtained by evaluating various HOG configurations with signed oriented gradients and different window size.

	Signed Gradients	
Window Size	$time[ms]$	$ratio$
5	6.279	3.48
8	2.452	1.36
10	1.803	1.00
12	1.362	0.75

Table 8.2: Mean time required for classifying a leg candidate with the non-linear SVM classifier, using various HOG configurations with signed oriented gradients and different window size. The time includes both HOG feature extraction and actual classification and it is computed when running on an Intel Core i5-2415M laptop. The column of ratios gives a more CPU-independent idea of how the different configurations compare.

Comparison with Alternative Methods

With reference to the alternative methods presented in Section 8.2, we now compare the performance achieved by the different approaches in the *Legs Detection* task.

The plot in Figure 8.5 shows the precision-recall curves computed when varying the *leg probability threshold* τ, i.e. the probability used to discriminate legs and non-legs candidates. The reported results refer to the *S-Difficult* scenario in our testing dataset.

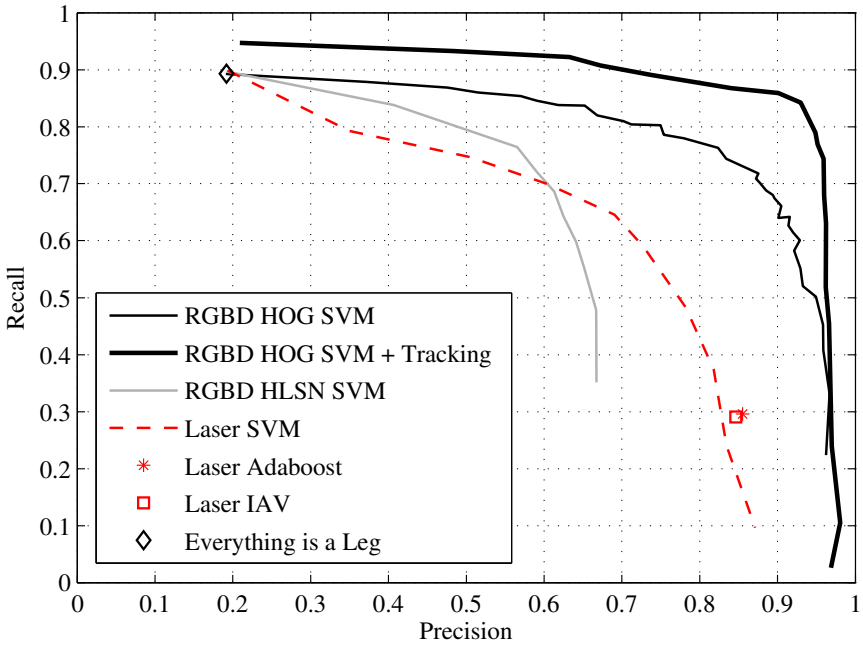


Figure 8.5: Legs Detection Performance in the Precision-Recall plane for different leg detection approaches. The ideal algorithm has precision = recall = 1 (which yields an F1-score of 1). By tuning the τ parameter, the tradeoff between precision and recall is explored, yielding a precision-recall curve. Some methods appear as single points since by design they only output a binary classification.

The *RGBD HOG SVM* curve is obtained with the classifier used by our system, which clearly outperforms all alternative approaches. Moreover, the *RGBD HOG SVM + Tracking* curve shows that tracking also improves frame-based detection performance, because the leg probability associated to each candidate is filtered over time, which limits the impact of occasional occlusions.

In the plot, the *Everything is a Leg* point indicates the precision and recall values obtained by assigning a leg class to all the candidates returned by the *Candidates Selection* process, as discussed in Section 8.3.1. This, in fact, corresponds to the value taken by each curve when considering $\tau = 0$. It is important to notice how the maximum recall value reached with the tracking increases (by nearly 5%) with respect to the starting level of the *Everything is a Leg* point. The reason of this improvement is that, thanks to the peculiar way data association is handled (see Section 6.4), our tracking algorithm manages to return two distinct legs close to each other, even when the preceding clustering algorithm merges both into a single cluster.

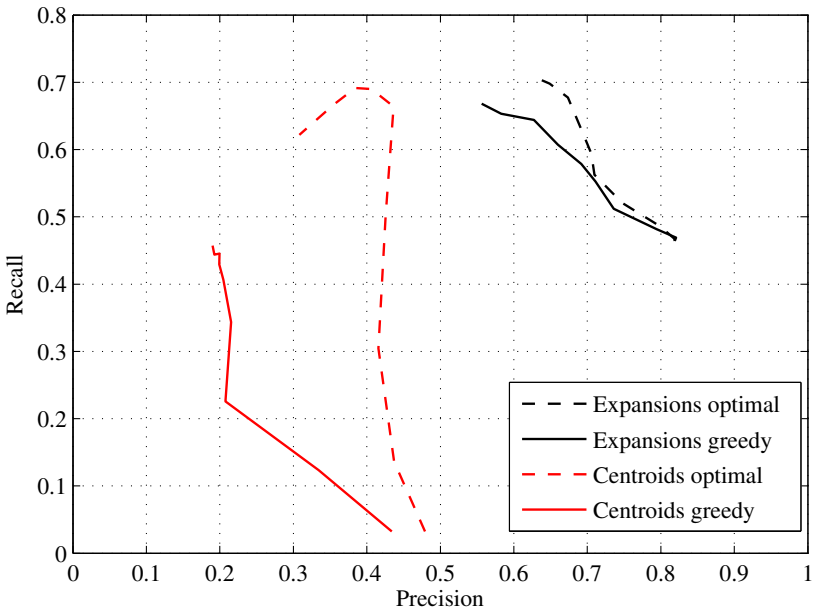


Figure 8.6: Legs Pairing Performance in the Precision-Recall plane for different heuristics while varying the threshold δ.

8.3.2 Legs Pairing

In this section, with reference to the problem formulation introduced in Section 6.5, we compare two different distance functions:

- Δ: it computes the minimum euclidean distance between the expanded 3D candidate points (up to the hip level, when possible);
- Δ': it simply computes the euclidean distance between the candidates feet centroids projected on floor.

We explore how the *legs pairing* precision and recall values change for both Δs functions while varying the threshold δ. A branch and bound approach for finding the optimal solution of the legs pairing problem could be computationally expensive for real time applications, expecially when the connected components in the graph have a high order (this happens when the threshold δ is selected to be more permissive). In order to deal with this situation, a suboptimal greedy approach is needed. Thus, we also evaluate how precision and recall of legs pairing based on a greedy solution compare with levels obtained with the optimal one.

In Figure 8.6 we report the results obtained on the *S-Difficult* scenario. As it can be observed, selecting the distance function Δ brings to higher precision values, which reach a maximum in correspondence of a very small

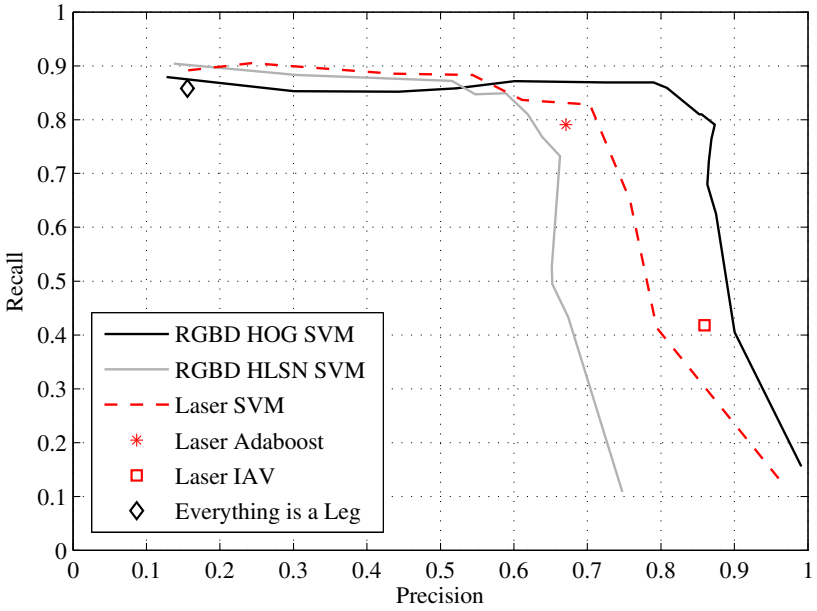


Figure 8.7: People Detection Performance in the Precision-Recall plane for different leg detection approaches. By tuning the τ parameter, the tradeoff between precision and recall is explored, yielding a precision-recall curve. Some methods appear as single points since by design they only output a binary classification.

value of δ. Its drawback consists of missing nearly half of the true leg pairs when using that value for δ. We decide to favor a high precision, at expense of possible low recall, since in our people tracking algorithm all the challenging legs pairings are dealt with a PDAF update that considers multiple pairing hypothesis. It can also be noticed from the plot that for both Δs functions the search for the optimal solution leads to better performances, but in the specific case of our Δ function the difference with the greedy approach is not very significant.

All these considerations suggest us to select the Δ function based on 3D candidates expansions, adopting a low threshold δ ($\delta = 7.5\ cm$) and a greedy technique for the search of the solution.

8.3.3 People Detection

The plot in Figure 8.7 shows the precision-recall curves concerning *People Detection*, using both our system and alternative methods for legs detection presented in Section 8.2. In particular, the performance of a given alternative method is computed by applying our own tracking approach to detections returned by such method.

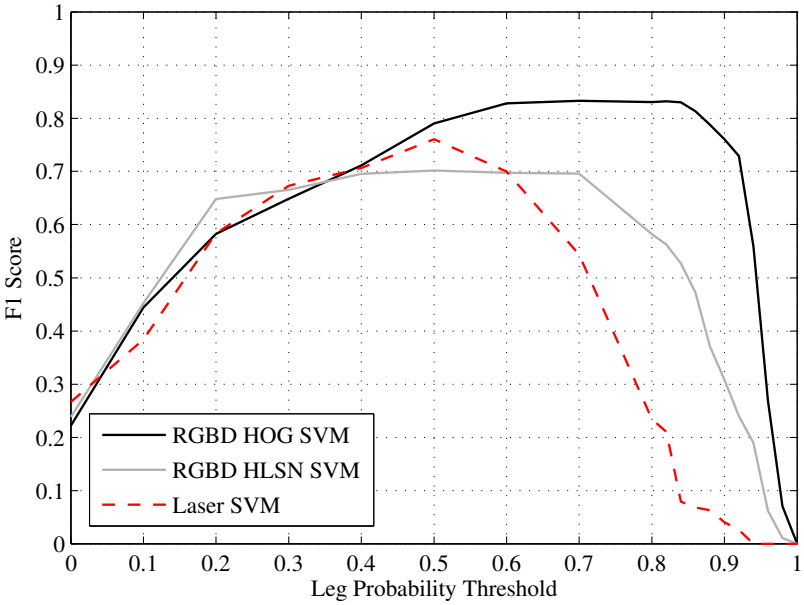


Figure 8.8: People Detection F1 Score Comparison. The plot shows how the F1 score changes while tuning τ, for both our system and alternative methods.

	Legs Detection			**People Detection**		
Scenario	*p*	*r*	*f1*	*p*	*r*	*f1*
S-Easy	0.97	0.91	0.94	0.88	0.91	0.89
S-Medium	0.98	0.81	0.88	0.93	0.83	0.88
S-Difficult	0.96	0.79	0.87	0.85	0.81	0.83

Table 8.3: Summary of the performance of our system on the three testing scenarios. For both legs and people detections, we report precision p, recall r and F1 score $f1$ using $\tau = 0.8$.

The reported results refer to the *S-Difficult* scenario in our testing dataset. It can be observed how our system, corresponding to the *RGBD HOG SVM* curve, still outperforms all the considered alternative approaches.

In order to determine the optimal *leg probability threshold* τ, in Figure 8.8 we further analyze the *People Detection* performance of our system while varying τ. We observe that the F1 score reaches a stable maximum level in the interval 0.5 - 0.9, which shows that the parameter is not a critical one. We select an intermediate value of $\tau = 0.8$ as the default best value. In the same plot we also report the F1 score obtained when tuning τ for the two alternative approaches returning probabilities. In the case of the *Laser*

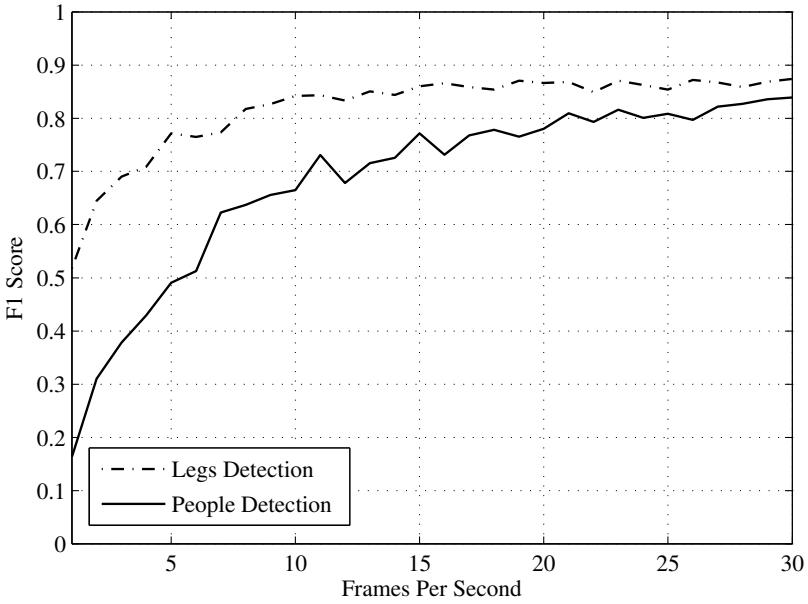


Figure 8.9: F1 score curve highlighting the effect of frame rate on system performance using $\tau = 0.8$.

SVM curve, in particular, we note that the maximum F1 score is both lower and less robust to variations in the threshold parameter.

Table 8.3 reports the performance of our system on the three different testing scenarios, when using $\tau = 0.8$ as determined above. Additional qualitative results and considerations are reported in Figure 8.11.

8.4 Effect of Frame Rate for Real-time System

Our tracking approach assumes a constant velocity state transition model. This simplifying hypothesis is a good approximation as long as the temporal interval between two observations remains small. Therefore, we study how the performance changes when limiting the input framerate – which may be useful in case of systems with low processing power. Adopting the optimal leg probability threshold determined above, we evaluate our system by simulating a given frame rate on the recorded *S-Difficult* scenario test video. Figure 8.9 reports the F1 score for both the legs detection problem and the people detection problem, versus the simulated frame rate. It can be observed that the performance penalty when limiting the framerate is marginal as soon as the framerate is kept above 10 fps. Under this threshold, the legs tracking performance starts degrading significantly, which also impacts the people tracking performance.

8.5 Deployments on Robots

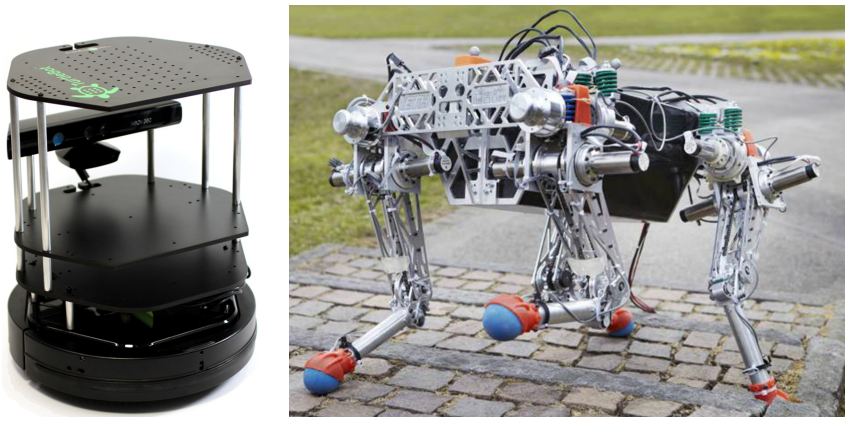
In order to test our system when the sensor is mounted on a physical mobile robot and subject to its movements in the environment, we deploy our system on two different robot platforms:

- Turtlebot [7]: it is a low-cost, personal robot kit with open-source software. It is a small footprint and wheeled robot, suitable for indoor environments and endowed with an RGB-D sensor in its default configuration (Figure 8.10(a)).
- StarlETH [29]: is a quadruped robot based on the high-compliant series elastic leg ScarlETH. It was developed at ETH Autonomous System Lab for fast, efficient, and versatile locomotion on four legs. It is also suitable for outdoor environments and uneven ground floor (Figure 8.10(b)).

The ROS interface of our system allows its straightforward integration in all robots which rely on a ROS middleware. Moreover, in this way it is possible to exploit the odometry of the robot to more accurately estimate people velocities regardless of the robot's own motion.

Having not restricted our system with hypothesis of planar movement of a robot, the performance has revealed good also when subject to more irregular and swinging movements typical of a legged robot.

Qualitative videos recorded from these robots and showing the output of our alghoritm on the processed RGB-D data are available at `http://bit.ly/perceivingpeople`.



(a) Turtlebot (b) StarlETH

Figure 8.10: Deployment on real robots.

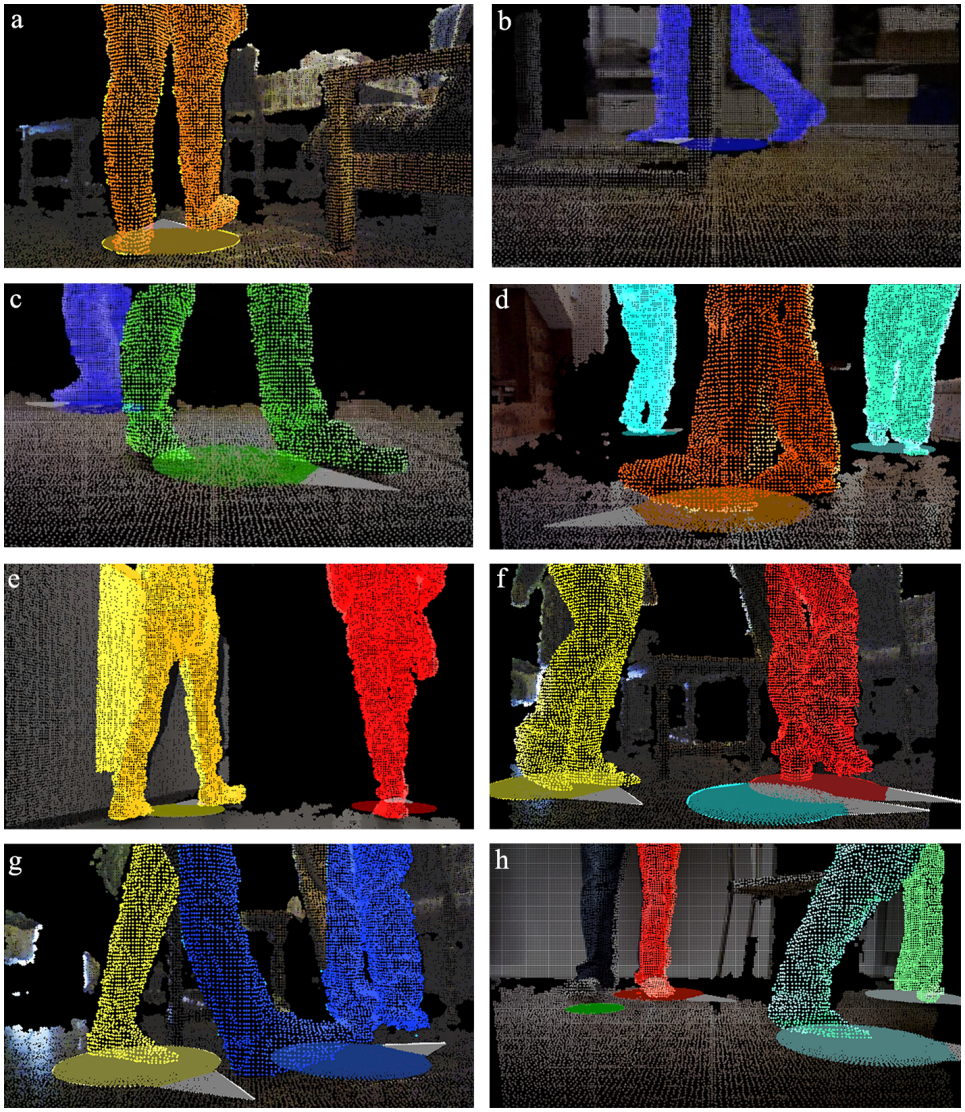


Figure 8.11: Example outputs from our system in challenging scenarios. All 3D points corresponding to each detected person are displayed in a different color. A large disk drawn on the floor represents the barycenter for each person, with an arrow pointing towards the computed motion direction. Images a), b), c) and d) report a correct behaviour of the system, also in presence of multiple people and various distracting objects. Remaining images show various failure modes. In e) part of the wall was merged with the person's leg (which was nonetheless correctly detected). In f) the person on the right was not re-associated to the correct person track (gray disk) after being briefly lost, but a new track was created instead. In g) two legs were not correctly joined, and each was associated to its own person track. In h) one leg of the person on the left is not joined to the other. Note that most of these failure cases are unstable, short-lived configurations which quickly resolve, as is apparent in supplementary videos.

Chapter 9

Conclusions

We presented a practical approach for detecting and tracking people indoors, from a small-footprint ground robot using an RGB-D sensor. Solving the problem requires careful handling of the peculiar challenges arising from this scenario. We demonstrated that, in all considered datasets, our system performs better than alternatives, and yields good results both quantitatively (Section 8.3) and qualitatively (see supplementary videos at `http://bit.ly/perceivingpeople`).

We provide a practical, real-time, ROS-enabled implementation ready to run on common robot platforms such as the Turtlebot and Kobuki.

In its current version, the system does not exploit the RGB data provided by the sensor (they are only used to provide a visualization of the system output and to allow a qualitative evaluation); this is precious information that could be exploited for improving data association and enabling additional functionality such as person re-identification once a track is temporarily lost.

The Microsoft Kinect and Asus Xtion RGB-D sensors properly work only in indoor environments (since they suffer from sunlight interference), but there exist other technologies that allow to capture the 3D point cloud also in outdoor environments, such as 3D laser scanners. Our system could be easily adapted to work also in this new condition.

Moreover, the assumption of a mostly flat and regular ground floor could be loosened by integrating, if available, some additional information about the sensor 3D position and orientation (e.g. derived by the actual knowledge of position and orientation of the robot estimated with an Inertial Measurement Unit).

Bibliography

[1] Asus Xtion PRO LIVE. `http://bit.ly/19Sfn62`.

[2] Kobuki. `http://bit.ly/1lj33Qj`.

[3] Microsoft Kinect. `http://bit.ly/1cAhjxI`.

[4] PrimeSense. `http://bit.ly/1bVzDCi`.

[5] PrimeSense NITE Middleware. `http://bit.ly/IAIHRb`.

[6] The Point Cloud Library. `http://pointclouds.org/`.

[7] The TurtleBot Robot development kit. `http://bit.ly/lqRgND`.

[8] Kai Oliver Arras, O Martınez Mozos, and Wolfram Burgard. Using boosted features for the detection of people in 2d range data. In *Proc. ICRA 2007*.

[9] K.O. Arras, S. Grzonka, M. Luber, and W. Burgard. Efficient people tracking in laser range data using a multi-hypothesis leg-tracker with adaptive occlusion probabilities. In *Proc. ICRA 2008*.

[10] Yaakov Bar-Shalom et al. The probabilistic data association filter. *IEEE Control Systems*, 29(6):82–100, 2009.

[11] Yaakov Bar-Shalom, X Rong Li, and Thiagalingam Kirubarajan. *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[12] Filippo Basso, Matteo Munaro, Stefano Michieletto, Enrico Pagello, and Emanuele Menegatti. Fast and robust multi-people tracking from rgb-d data for a mobile robot. In *Proc. Intelligent Autonomous Systems (IAS) 2012*.

[13] S.S. Blackman. Multiple hypothesis tracking for multiple target tracking. *Aerospace and Electronic Systems Magazine, IEEE*, 19(1):5–18, Jan 2004.

[14] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas NÃ$\frac{1}{4}$chter. The 3d hough transform for plane detection in point clouds: A review and a new accumulator design. *3D Research*, 2(2):1–13, 2011.

[15] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[16] Michael D Breitenstein, Fabian Reichlin, Bastian Leibe, Esther Koller-Meier, and Luc Van Gool. Online multiperson tracking-by-detection from a single, uncalibrated camera. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(9):1820–1833, 2011.

[17] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[18] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. CVPR*, 2005.

[19] Jakob Engel, Jurgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a monocular camera. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1449–1456. IEEE, 2013.

[20] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. Knowledge Discovery and Data Mining (KDD) 1996*.

[21] Gabriele Fanelli, Matthias Dantone, Juergen Gall, Andrea Fossati, and Luc Van Gool. Random forests for real time 3d face analysis. *Int. J. Comput. Vision*, 101(3):437–458, February 2013.

[22] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[23] A. Fod, A. Howard, and M.J. Mataric. A laser-based people tracker. In *Proc. ICRA 2002*.

[24] Armando Pesenti Gritti, Oscar Tarabini, Alessandro Giusti, Jerome Guzzi, Gianni A. Di Caro, Vincenzo Caglioti, and Luca M. Gambardella. Video: Perceiving people from a low-lying viewpoint. In *Proc. Human Robot Interaction (HRI) 2014*.

[25] J. Guzzi, A. Giusti, L.M. Gambardella, G. Theraulaz, and G.A. Di Caro. Human-friendly robot navigation in dynamic environments. In *Proc. ICRA 2013*.

[26] Frederik Hegger, Nico Hochgeschwender, Gerhard K Kraetzschmar, and Paul G Ploeger. People detection in 3d point clouds using local surface normals. In *RoboCup 2012: Robot Soccer World Cup XVI*.

[27] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *Proc. International Symposium on Experimental Robotics (ISER) 2010*.

[28] Nicholas R Howe, Michael E Leventon, and William T Freeman. Bayesian reconstruction of 3d human motion from single-camera video. In *NIPS*, volume 99, pages 820–6, 1999.

[29] Marco Hutter, Michael Gehring, Michael Bloesch, A Hoepflinger Mark, C David Remy, Roland Yves Siegwart, A Hoepflinger Mark, A Hoepflinger Mark, Roland Yves Siegwart, and Roland Yves Siegwart. *StarlETH: A compliant quadrupedal robot for fast, efficient, and versatile locomotion*. Autonomous Systems Lab, ETH Zurich, 2013.

[30] Thorsten Joachims. Making large scale svm learning practical. 1999.

[31] T. Kanade, A. Yoshida, K. Oda, H. Kano, and M. Tanaka. A stereo machine for video-rate dense depth mapping and its new applications. In *Proc. CVPR 1996*, 1996.

[32] T. Kanda, D.F. Glas, M. Shiomi, and N. Hagita. Abstracting people's trajectories for social robots to proactively approach customers. *IEEE Transactions on Robotics*, 25(6):1382–1396, 2009.

[33] Hadi Kheyruri and Daniel Frey. Comparison of people detection techniques from 2d laser range data.

[34] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[35] P. Koutlemanis, X. Zabulis, A. Ntelidakis, and A.A. Argyros. Foreground detection with a moving rgbd camera. In *Proc. International Symposium on Visual Computing (ISVC) 2013*.

[36] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer. Multi-camera multi-person tracking for easyliving. In *Visual Surveillance, 2000. Proceedings. Third IEEE International Workshop on*, 2000.

[37] Chi-Pang Lam, Chen-Tun Chou, Kuo-Hung Chiang, and Li-Chen Fu. Human-centered robot navigation – towards a harmoniously human-robot coexisting environment. *IEEE Transactions on Robotics*, 27(1):99–112, 2011.

[38] Jae Hoon Lee, T. Tsubouchi, K. Yamamoto, and S. Egawa. People tracking using a robot in motion with laser range finder. In *Proc. IROS 2006*.

[39] Hsuan-Tien Lin, Chih-Jen Lin, and Ruby C Weng. A note on platt's probabilistic outputs for support vector machines. *Machine learning*, 68(3):267–276, 2007.

[40] ROS Wiki Contributors. Ros on matlab. `http://wiki.ros.org/groovy/Planning/Matlab`, 2014.

[41] Stephen J McKenna, Sumer Jabri, Zoran Duric, Azriel Rosenfeld, and Harry Wechsler. Tracking groups of people. *Computer Vision and Image Understanding*, 80(1):42–56, 2000.

[42] OscarMartinez Mozos, Ryo Kurazume, and Tsutomu Hasegawa. Multi-part people detection using 2d range data. *International Journal of Social Robotics*, 2(1):31–40, 2010.

[43] Yasushi Nakauchi and Reid Simmons. A social robot that stands in line. *Autonomous Robots*, 12(3):313–324, 2002.

[44] Y. Ohno, J. Miura, and Y. Shirai. Tracking players and estimation of the 3d position of a ball in soccer games. In *Proc. Pattern Recognition, 2000.*, volume 1, pages 145–148 vol.1, 2000.

[45] Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *Proc. British Machine Vision Conference (BMVC) 2011*.

[46] K. Okuma, A. Taleghani, N. De Freitas, J. J. Little, and D. G. Lowe. A boosted particle filter: Multitarget detection and tracking. In *Proc. European Conference on Computer Vision (ECCV) 2004.*

[47] D. Ramanan, D.A. Forsyth, and A. Zisserman. Strike a pose: tracking people by finding stylized poses. In *Proc. CVPR 2005.*, volume 1, pages 271–278 vol. 1, 2005.

[48] T. Sasaki, D. Brscic, and H. Hashimoto. Human-observation-based extraction of path patterns for mobile robot navigation. *IEEE Transactions on Industrial Electronics*, 57(4):1401–1410, 2010.

[49] Dirk Schulz, Wolfram Burgard, Dieter Fox, and Armin B Cremers. People tracking with mobile robots using sample-based joint probabilistic data association filters. *The International Journal of Robotics Research*, 22(2):99–116, 2003.

[50] E.A. Sisbot, R. Alami, T. Simeon, K. Dautenhahn, M. Walters, and S. Woods. Navigation in the presence of humans. In *Proc. IEEE-RAS International Conference on Humanoid Robots (Humanoids) 2005.*

[51] L. Spinello, K. O. Arras, R. Triebel, and R. Siegwart. A layered approach to people detection in 3d range data. In *Proc. AAAI 2010.*

[52] Luciano Spinello and Kai O Arras. People detection in rgb-d data. In *Proc. IROS 2011.*

[53] M. Svenstrup, S. Tranberg, H. J. Andersen, and T. Bak. Adaptive human-aware robot navigation in close proximity to humans. *International Journal of Advanced Robotic Systems*, 8(2):1–15, 2011.

[54] Siyu Tang, Mykhaylo Andriluka, Anton Milan, Konrad Schindler, Stefan Roth, and Bernt Schiele. Learning people detectors for tracking in crowded scenes. *ICCV 2013*, 2013.

[55] Ba-Ngu Vo and Wing-Kin Ma. The gaussian mixture probability hypothesis density filter. *Signal Processing, IEEE Transactions on*, 54(11):4091–4104, 2006.

[56] G. Welch and G. Bishop. An introduction to the kalman filter, 1995.

[57] Lu Xia, Chia-Chih Chen, and J.K. Aggarwal. Human detection using depth information by kinect. In *In Proc. Computer Vision and Pattern Recognition Workshops (CVPRW) 2011.*