Università degli Studi di Padova
Facoltà di Ingegneria
Dipartimento di Ingegneria dell'informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Human action recognition from RGB-D frames

Candidato
Gioia Ballin

Relatore
Prof. Emanuele Menegatti

Correlatore
Dott. Matteo Munaro

Anno Accademico 2011-2012

To my parents

Acknowledgments

I would like to acknowledge Prof. Emanuele Menegatti for giving me the chance to work in a research lab. Special thanks go to Matteo Munaro and Stefano Michieletto that followed me during this work day-by-day. I want to thank also Riccardo De Battisti, Stefano Ghidoni, Riccardo Levorato, Andrea Marcato and Alessandro Zanella for having helped me to realize the dataset discussed in this work. My final thanks go to Arrigo Guizzo and Antonio Passamani that have supported me all the time during the thesis project.

Abstract

In this work we propose a novel approach to real-time human action recognition. We use the Microsoft Kinect sensor and an underlying tracking system to robustly detect people in the scene. Next, we estimate the actual 3D optical flow related to the tracked people from point cloud data only. Each point cloud associated with a track in a specific frame is matched against the same point cloud in the immediately previous frame in order to find correspondences between points. Furthermore, we investigate about possible methods to accomplish a proper correspondence rejection and noise removal. A suitable descriptor is then computed for the 3D optical flow information: we create a 3-dimensional grid surrounding each detected track and we summarize the flow by taking relevant information from each voxel in the grid. Experiments are performed on a newly created dataset which contains six human actions performed by six different actors. Experimental results show the effectiveness of the proposed approach.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recognizing human actions is one of the most challenging research fields in computer vision. This chapter aims to introduce the reader into this particular field and lays the foundations for a good understanding of the following chapters.

Human action recognition is an important research area in computer vision. First investigations about this topic began in the seventies with pioneering studies accomplished by Gunnar Johanssons [19]. From then on, interest in the field grew increasingly, motivated by a number of potential applications. In this discussion, we present the main application areas of human action recognition to provide the reader a concrete idea of the impact that vision-based recognition systems could have in everyday life.

Behavioural Biometrics

Biometrics, or biometric authentication, aims to identify humans by their characteristics or traits. By contrast to traditional biometric approaches, the so called "behavioural biometrics" believes that behavioural cues are as useful as physical attributes to recognize humans. By contrast to the acquisition of physical cues, behavioural cues do not require the cooperation of the reference subject, so they can be captured without interferring with the subject's current activity. It is clear it is necessary to observe individuals for a certain period of time to be able to obtain behavioural traits, and then approaches for action recognition extend naturally to this task. Recent applications of the behavioural biometrics are related to airport controls. Since the terrorism threat emerged in the past years, airport controls have been intensified and new approaches are currently investigated to detect suspicious behaviours in passengers. In order to indentify terrorists, the current research is devoted to the developement of automatic systems able to observe passengers during the check-in process or the gate controls, and to detect potential terrorists among them.

Video surveillance

Video surveillance is the most natural application of human action recognition. Indeed, it involves the monitoring of behaviours and activities, usually attributed to people, with the purpose of influencing, managing, directing, or protecting. Security and surveillance systems have traditionally relied on networks of cameras. The transmitted images are tipically monitored by a human operator who has to control the activities taking place in the camera's view field. The technological progress has led to the introduction of increasingly powerful cameras on the market, and in recent years also RGB-D sensors have become available as an alternative to standard color cameras. These technological advances have helped human operators to accomplish their monitoring job. Furthermore, security agencies are currently seeking for vision-based solutions able to completely replace the human labor. In fact, nowadays the need of an automatic system able to detect specific actions is dictated by real world contexts in which millions of cameras constantly monitor traffic, pedestrians, animals, environments, thus generating big amounts of videos. Therefore, the volume and complexity of those videos far exceeds the capabilities of human agents to analyze image sequences and take real-time decisions. To this aim, automatic recognition of suspicious actions, and more generally, automatic action recognition, are target issues that have to be solved.

HCI – Human Computer Interaction

Human computer interaction involves the design and the development of usable interfaces that enable humans and computers to communicate each other. Visual cues are the most important form of non verbal communication among humans, and the current challenge is to exploit such visual information to develop computers that can better interact with humans. When visual cues are represented by actions or activities, respectively human action recognition and human activity recognition are exploited. This is also the case of intelligent environments, such as smart homes and interactive workspaces, in which human gestures or actions have to be recognized to enable the environment to properly react. Furthermore, some special kinds of smart rooms can be devoted to the monitoring of children or elderly people, which are also tipical task in the field of service robotics.

Content-based video analysis

The content-based video analysis regards the design and the development of algorithms able to automatically analyze videos to detect and determine termporal events. Important applications belonging to this wide sector are: video indexing, video storage and video retrieval. All those tasks require to learn patterns from raw videos and then to summarize each video based on its content. If the content is represented by actions or activities performed by one or several people, it is clear that either human action recognition or human activity recognition comes into play.

In this work we propose a novel approach to human action recognition. First, our system relies on the acquisition of RGB-D data, by contrast to traditional approaches in which RGB images from color cameras are exploited. Second, a tracking phase is initally performed to detect people in the scene. Once people are tracked, we estimate the 3D optical flow for each detected track. The innovation lies in the way we compute 3D optical flow: we estimate the 3D velocity vectors from point cloud data only, thus obtaining a real-time calculus of the optical flow. When the flow is estimated, we compute a particular descriptor that allows us to classify actions by means of learning-based classification techniques.

1.1 Human actions and activities

This section addresses the issue of defining what a human action actually is. This is not straightforward as it could appear at first sight and it is required to define what is an action, because it helps to fully understand in which context this thesis work takes place and its motivations. While dealing with the computer vision literature, it seems that a lot of publications make an interchangeable use of the words **action** and **activity** and this can lead to frequent misunderstandings by the reader. The conceptual problem of defining what a human action is was first addressed in [10] by Aaron F. Bobick. Bobick distinguished between movements, activities and actions, but the meaning he associated to the word “activity” is far different from its current use. In this work, the definition given is much more similar to what could be found in [43]. Therefore an **action** is considered to have more complexity with respect to a single movement or gesture, whereas the concurrent, parallel or consecutive execution of two or more single actions constitutes what is referred with the term **activity**. An action refers to simple motion patterns and it is usually executed by one or at most two people. Furthermore an action typically has a short temporal duration. Examples of actions are: *bending*, *hand waving*, *running*, *jumping*, *shaking hands*. By contrast, an activity is a complex sequence of actions, as previously defined. Activities could be performed by several people at the same time or at different times and they are characterized by having a longer temporal duration with respect to actions. Examples of activities include the gathering and dispersing of groups of people.

However, there is not a crisp boundary between actions and activities and in fact there is a significant “gray area” between these two extremes. This gray area is mostly due to the subjectivity of the observer and to the wide variety of actions that a single person can accomplish alone or while interacting with other people. Turaga et al. [43] take as an example the gestures of a music conductor and they said that the music conductor gestures are not so “simple” to be categorized as an action but at the same time they are not so “difficult” to be categorized as an activity. This is just the point: in order to discern between actions and activities the discretion of the observer also comes into play, furthermore there are many types of actions, some of which will be easier to recognize and some of the others will constitute a challenge to the recognition although they do not represent activities. This work is focused on recognizing actions. The reason is simple: the work started from scratch for what concerns the recognition field, so it was reasonable to build an initial recognition system able to recognize actions at first, whereas activities have been left to further future developments.

1.2 Recognizing human actions

Human action recognition is a challenging problem that required efforts coming from the best researchers in the computer vision field. At first, researchers focused on recognizing primitive movements or simple gestures, then switched to the recognition of human actions, where the term “actions” has the meaning defined in Section 1.1, and at last they faced the problem of recognizing complex human activities. Recognizing human actions and activities are active research fields also nowadays. Furthermore, always new challenges have been introduced in recent years. These challenges mostly regard possible existing variations in the recording and execution of the action. For example, the environment in which an action takes place is an important source of variation in the recording, so action recognition in cluttered or dynamic environments has also been experimented. Moreover, the speed at which an action is performed by a certain actor could be different from the speed at which the same action is performed by another actor. A good human action recognition approach should also be able to generalize over possible variations in the execution of the actions.

Before continuing the dissertation, it is required to define the problem of human action recognition, and specifically what are the inputs and outputs to the problem and what are the guidelines to achieve the final results. The goal of human action recognition is to identify which action is represented in a finite sequence of images given as input to the system. Furthermore, the action associated with each video can be selected from a finite set of action classes. The entire recognition process can be divided in three major steps:

1. Extraction of features from the sequence of images given as input to the system.

2. Processing a features description suitable for the application in example.

3. Classification of the descriptors obtained at point 2.

Features are specific pieces of information which contain all the visual motion information relevant to the application at issue. Features play a central role in human action recognition: they actually have a discriminant power in discerning between different action classes. For example, suppose that different features have been extracted from the same sequence of images and these features are referred to A and B. It can happen that A is the best way to represent the motion information contained in the video frames. A recognition system that would use A as baseline features would have better classification results than a system that would use B instead, and finally this would result in a better recognition performance. This is essentially why this work is focused on features: the discriminant power of a recognition system heavily relies on features. New ways to detect and extract features could significantly improve existent recognition methods or even could give rise to new recognition methods. Obviously, not all the discriminant power relies on features (and their descriptors). A portion of this effective power is certainly due to the classifier used. However, classifiers are well-known instruments coming from machine learning field and it is very hard to substantially improve each of them.

1.3 Aim of the thesis

The main objective of this work consists on developing an automatic system able to recognize human actions in indoor office environments. Furthermore, this system is characterized by having a 3D tracking as underlying system and by using a widely available cheap sensor, namely the Microsoft Kinect, to acquire input data. Since the recognition system is targeted to applications in the field of video surveillance, it has been developed to:

1. execute real-time;

2. recognize the actions performed by each individual present in the scene at a certain time instant;
3. perform a continuous recognition task, if possible.

Point 1 is justified by two different motivations: first, the underlying tracking system is a real-time system and we really did not want to lose this great property; second, achieving real-time performances together with good recognition results is a challenging task in human action recognition. Indeed, an important trade-off exists between the recognition accuracy and the computational performances: fast algorithms are able to perform real-time computations, but these calculus can not be too much complex; by contrast, off-line recognition methods generally perform complex computations without worrying about time performances. So, we have preferred to develop a fast recognition system even at the expense of a lower recognition accuracy. It is important to remark that it is **possible** to obtain poor results with a fast algorithm, but this does not really represent a certainty. Furthermore, the recognition performances are offset by the usefulness that real-time performances hold in a video surveillance context. Finally, also points 2 and 3 represent challenging tasks in human action recognition, in fact:

- All the works at the state of the art of human action recognition with RGB-D data concentrate on recognizing isolated actions, namely actions performed by only one individual and defined as stated in Section 1.1.[1]
- Most of the work at the state of the art of human action recognition from RGB images concentrates on recognizing isolated actions, but in recent years new challenges have been addressed.[2]

For instance, such a challenge is represented by the continuous recognition. In the continuous recognition task, an actor performs several actions in a certain sequence and the recognizer has to correctly detect each executed action. This means that the algorithm has to recognize the series of actions, but in addition it has also to locate temporally the actions, and to assign to each temporal interval the correct action. It is clear that the job of continuously recognizing human actions is much more difficult with respect to the recognition of isolated actions, since some mechanism to perform the action segmentation is required before (or concurrently to) the execution of the recognizer. In this work, we have combined difficulties arising from the continuous recognition to those resulting from the recognition made over several people at once, all within a context in which only real-time computations are allowed.

[1]See Section 2.1.
[2]See Section 2.2.

1.4 Organization of the thesis

The thesis is organized as follows. Chapter 2 provides a review of the state of the art of human action recognition together with some important remarks about the existing differences between our system and systems already known in literature. Chapter 3 focuses on the fundamental components on which the recognition system we propose in this work is built. Chapter 4 describes a simple recognition system developed in the start-up phase of this work. In such a system global cues are used to discern between action classes. Chapter 5 and 6 describes the evolution of the work with respect to Chapter 4. In those chapters we propose a novel approach to human action recognition, in which 3D optical flow is estimated from point cloud data only, and a well-suited descriptor is built on it. Finally, Chapter 7 discusses the validity and the potential of our innovative recognition system both by analyzing experimental results and by showing future developments of the system.

Chapter 2

Literature review

This chapter provides a survey on the state of the art of human action recognition in videos. First, recognition from RGB video images is addressed. Then the discussion is focused on techniques that exploit RGB-D sensors to recognize human actions. Finally, brief considerations are given about the state of the art of comparison between different approaches.

As outlined in Section 1.2, the recognition process generally requires three essential steps. Each step is associated to a specific computational problem that has to be solved. First of all, it is necessary to extract some pieces of visual information that will be further processed in the subsequent steps. These pieces of information are called **features** in the computer vision field and are designed to completely summarize the motion information. Secondly, features are encoded in a suitable data structure: the **descriptor**. The role of the descriptor is to summarize specific properties of the features previously computed. In some sense, we are able to recognize human actions using only descriptors, because they are built explicitly to encode only the discriminant information coming from the features extraction stage. When descriptors are finally available for an observed frame, human action recognition becomes a classification problem. Indeed, each video is represented by means of a set of descriptors and this set represents what is actually fed into the classifier. Therefore a classifier is first trained with a set of labeled videos (namely sets of descriptors) and its role consists on determining the correct label for a previously unseen video. Each label represents an action, and each action is taken from a finite set of possible occurring actions. Finally an action label or a distribution over labels is given as output of the classification step at each frame.

Furthermore, the classifier can express a confidence value for the decision taken.

This chapter provides an overview of the existing literature on human action recognition field and it is organized as follows: in Section 2.1 we propose a brief discussion on the state of the art of human action recognition carried out from 2D video images, whereas in Section 2.2 we argue about related work with RGB-D sensor; finally, the chapter concludes with Section , in which some general considerations about comparison between different recognition techniques are provided.

2.1 Recognition from RGB video images

Human action recognition has been studied by a number of different authors. In literature, most of the works is focused on recognition from 2D images, where simple color cameras are employed. However, it is possible to find some works in which also thermo-graphic cameras, infrared cameras or wearable sensor are used (two examples are given by [15], [48]). In all these cases, recognition is carried out from 2D video images. In all these cases, recognition is carried out from 2D video images. Since 2D video images have really been exploited a lot, this section goes into deep about this argument in a structured way. The application fields, the computation of features and descriptors, together with classification choices are briefly reported here.

2.1.1 Application fields

Since investigations about recognition from 2D video images started at least 15 years ago, there are several application fields already explored. The applications range from video surveillance to smart houses, from motion capture to behavioral biometrics, and again from content based video analysis to health-care and elder-care. Other applications can be found in classical robotics: teaching by touching, teaching by demonstration, service robotics, human-computer interaction are only few of them.

2.1.2 Features

Many features representations have been developed in order to recognize actions from video sequences based on color cameras. In addition, the features extraction step represents the main source of diversification between various recognition techniques. The choice of which features to use often depends on the ultimate goal that has to be achieved by the recognition system. Some common goals require the development of robust representations when particular variations occur in the scene. Such variations can regard the appearance of the actors performing the actions (anthropomorphic variations, clothing variations, etc.), the background of the scene (light variations, variations due to cluttered or dynamic backgrounds), the viewpoint and so on. Furthermore, the extracted features shall enable a robust classification of actions, because this is the final objective of action recognition. Also the temporal aspect is important in action performance. Some features explicitly take into account the temporal dimension, while some others extract image features for each frame in the sequence. In this case, the temporal variations need to be dealt with in the classification step.

In this section, we discuss the features that have been extracted from 2D video images in literature. According to [32], we divide the features extracted into two broad categories: global representations and local representations.

Global representations

Global representations encode the visual information as a whole. These representations are typically obtained by means of a top-down approach: first a person is localized in the scene (using background subtraction or tracking methods) and then the region of interest (abbreviated ROI), identified in the localization step, is encoded as a whole. Global representations, as well as local representations, have advantages and disadvantages at the same time. The advantages are listed below:

- global representations are rich representations of the visual information: encoding much of the motion information enables to recognize actions with a good accuracy;
- good recognition performances in constrained environments: in such environments, variations are kept under control.

While the major drawbacks are:

- an accurate localization of people in the scene is required before starting the features extraction step;
- sensitivity to viewpoint variations, noise and occlusions.

However, new techniques have been developed to aim at a partial solution of the weaknesses mentioned above. These approaches are called *grid-based* and they perform a spatial division of the visual information into cells, each of which encodes part of the motion information locally (see [33], [9], [18], [42]). We distinguish between the following types of global representations:

- shape-based representations;
- optical flow-based representations;

- volume-based representations.

Popular shape-based representations include edges [6] and silhouettes of the human body [7]. In particular, silhouettes of the human body have been investigated a lot in literature and have shown promising results. The basic approach involves the extraction of specific features, called shape features, designed to summarize information about the pose of a person. Further advanced approaches [49] [3] [35] have been proposed. In these latter approaches, silhouettes are first extracted and then stacked along the spatial and the temporal dimension. The results of stacking silhouettes is a 3D spatio-temporal volume, and so these techniques can also be categorized as volume-based approaches.

Optical flow based representation have been used by lots of research groups [12] [20] [47]. Furthermore, new techniques that attempt to improve the standard approach have risen recently. An example of such a technique is outlined in [1]. Ali e Shah [1] compute a set of kinematic features derived from a optical flow calculation. Each kinematic feature encode some information related to the motion dynamics. After that, the salient characteristics of the kinematic features are summarized in structured spatio-temporal pattern, called kinematic modes. Finally, for the classification task, a Multiple Instance Learning approach is used and results are reported on publicly available datasets (see Section 2.3.1).

A 3D spatio-temporal volume, abbreviated in STV, is formed by stacking frames in a specific sequence. The already cited [49], [3], [35] can be classified as volume-based approaches. However there exists also volume-based works which do not use silhouettes and [21] [26] [39] are only some examples.

Local representations

Local representations describe the extracted visual information by means of a collection of local descriptors or regions of interest, the so called patches. By contrast to global representations, local representations are computed by following a bottom-up approach: first, spatio-temporal interest points are detected and then local patches are calculated around them. Finally, all the obtained local patches are combined into a final representation. Local representations, in turn, have advantages and disadvantages. The advantages are:

- Lower sensitivity to noise and partial occlusions with respect to global representations.
- The preventive application of a background subtraction or tracking method is not strictly required.

While the main disadvantages are:

- If the number of extracted interest points is too small, the final recognition system is not ensured to be robust.
- A pre-processing phase is sometimes needed, in order to detect a proper number of relevant points.

Before the widespread of RGB-D sensors, like Microsoft Kinect, spatio-temporal features really represented the direction in which researchers were focusing more. Obviously, research about local representations is still very active, but now it might not seem the most promising way to follow in research. With regard to local representations, works by Laptev et al. [24] [23], Dollár et al. [11], Schuldt et al. [38], Scovanner et al. [39], Niebles et al. [29], Kläser et al. [22] e Willems et al. [46] are a reference point for anyone who aims to implement a recognition technique based on the extraction of spatio-temporal features.

It should be noted that for each new spatio-temporal feature invented, often also a new descriptor was invented. Indeed, the cuboid detector was proposed by Dollár et al. [11] in conjunction with the cuboid descriptor, while the Hessian detector together with the ESURF descriptor were proposed by Willems et al. [46]. By contrast Laptev and Lideberg in [23] proposed a new detector, the Harris3D detector, but not a correspondent descriptor, while in [24] only new descriptors were introduced: this is the case of the well known descriptors HOG, Histograms of Oriented Gradients, and HOF, Histograms of Optical Flow. A modified version of HOG, was finally proposed by Kläser et al. [22]: here histograms of 3D gradient orientation were computed, and these gave risen to HOG3D descriptor. There is a last remarkable publication by Wang et al. [44]. In [44] a comparison between the most promising local representations is given. The comparison is based on common test beds, such as some selected publicly available datasets: KTH Action Dataset, UCF Sport Action Dataset and Hollywood2 Human Action Dataset (see Section 2.3.1) have been used to evaluate experimental results.

2.1.3 Classifiers

As stated in Section 2.1.1, human action recognition from 2D video images has been widely studied. Accordingly, lot of classifiers have been experimented till now. However, it is possible to distinguish classifiers into two broad categories: those which permit to model temporal variations, the so called temporal-state space classifiers, and those which not. The latters, are often called direct classifiers. Furthermore, very popular classifiers belong to this class: this is the case of nearest neighbors (abbreviated in k-NN) and discriminative classifiers, such as SVMs[1]. When a direct classifier is used, the temporal aspect cannot rely on the

[1]SVM, Support Vector Machine.

classifier and so it has to be handled in another way: one possible way to do this is to incorporate the temporal aspect directly in the features representations. An alternative method consists in performing the recognition on each frame individually. By contrast, the temporal evolution might not be explicitly handled before classification if a temporal-state space model is used. These kinds of models are generally representable by means of a graph structure which shows the evolving of an action. Temporal state-space classifiers can be ulteriorly distinguished in generative models and discriminative models. Conditional random fields belong to the class of discriminative models, while dynamic time warping and HMM[2] can be seen as generative models and have been extensively used in the action recognition field.

2.2 Recognition from RGB-D video images

Human action recognition from RGB-D images is really new in literature. The first work in this direction was done in collaboration with Microsoft Research [25]. In [25], a sequence of depth maps is given as input to the recognition system and there is no knowledge about the current RGB stream. In this work, relevant postures for each action are extracted and represented as a bag of 3D points. Then, the motion dynamics are modeled by means of an action graph. It is clear how our work differentiates from [25]: the input data, features and descriptors are all different. There are other seven papers that address the task of recognizing human actions with RGB-D sensors. However, not all these works have used Microsoft Kinect as a sensor: in two of them [16, 17], a Time of Flight sensor is used. Furthermore, [16, 17] refer to the same author. Let us analyze first works in which Microsoft Kinect is used.

The existing literature related to the exploitation of Microsoft Kinect sensor refers to [40], [41], [50], [31], [28]. [40, 41] refer to the same authors and represent the first attempt to exploit OpenNi skeleton tracking information to recognize human actions. In [40, 41], work is much more focused on the classification step, whereas our work concentrates on the features extraction step. Indeed, Sung et al. tested their method on different classifiers: an SVM classification is compared with a MEMM[3] classification.

By contrast, our aim was to develop innovative features in order to burst on the action recognition scene, so we chose not to concentrate too much on the classification phase. Our work follows the perspective outlined in [50] by Zhang and Parker instead. In [50], Zhang and Parker decided to extend spatio-

[2]HMM, Hidden Markov Model.

[3]MEMM, Maximum Entropy Markov Model.

temporal features (see Section 2.1) already developed for RGB data only, to the new dimension. The new features have been called 4D spatio-temporal features, where “4D” is justified by the 3D spatial components given by the sensor plus the time dimension. The descriptor computed is a 4D hyper cuboid, while Latent Dirichlet Allocation with Gibbs sampling is used as classifier. Another work in which typical 2D representations are extended to 3D is [28]. In [28], Ni et at. extend the exiting definitions of spatio-temporal interest points and motion history images [4] to incorporate also the depth information. At the same time, Ni et al. have constructed a new dataset which is declared to be publicly available. For the classification purpose, SVMs with different classifiers are used. Our work aims to extend an existing method for 2D images to the RGB-D field like [50] and [28], but differently from them, the definition of the new features already exists and we have developed a new efficient way to compute them. It is important to notice that probably both [40, 41], [50] and [28] are not real-time systems since nothing has been reported about time performances. Conversely, we expressively designed a real-time system. Furthermore, [40], [41] and [28] aim to recognize useful activities in the personal or assistant robotics field, whereas we look at useful actions in video surveillance applications. Another remarkable distinction factor between [40, 41], [50], [28] and our work is represented by the number of people that can simultaneously appear in the scene and execute actions. While [40, 41], [50], [28] have realized a single-person action recognition, our system can execute the recognition algorithm on each person present in the scene at a certain time instant. Unlike [40], [41], [28] Popa et al. in [31] concentrate on recognition of video surveillance actions. However, they refer to actions related to shopping behaviors, and these are really specific actions. By means of Microsoft Kinect, Popa et al. extract silhouette data for each person in the scene and then compute moment invariants to summarize features. Finally, descriptors are classified using different approaches: SVM, k-NN, LDC and HMM are used. Besides, [31] aims to build a real-time recognition system in which also continuous recognition is performed. These objectives are very similar to ours, but the features extraction and the description steps clearly differentiate us from [31]. Moreover, we are not interested on testing our system on more than one classifier yet, this may be left to future developments.

Finally, we compare our work with [16, 17]. In [16, 17], 3D optical flow is computed exploiting RGB-D data coming from the Time of Flight sensor. But there is not a substantial improvement with respect to the traditional way of computing optical flow. Indeed, Holte et al. compute 2D optical flow using the traditional Lukas-Kanade method and then extend the 2D velocity obtained to incorporate also the depth dimension. At the end of this process, the 3D velocity vectors are used to create an annotated velocity cloud. 3D Motion Context (3D-MC) and Harmonic Motion Context (HMC) serve the task of representing the extracted motion vector field in a view-invariant way. With regard to the classification task, Holte et al. have not followed a learning-based approach and preferred to apply a probabilistic Edit Distance classifier in order to identify which gesture best describes a string of primitives. [17] differs from [16] because it has been developed in a multi-view camera system. So, in [17] 2D optical flow is estimated for each view, then each of this flows is extended to the 3D definition and finally is combined into a unique 3D motion vector field. It should be noted that both [17] and [16] are devoted to solve a gesture recognition problem, while we look at recognizing actions. Furthermore, Holte et al. do not really exploit all the 3D information provided by the sensor while computing optical flow. Indeed, they calculate optical flow 2D whereas we have developed a new method to estimate optical flow 3D directly from RGB-D information. Optical flow 2D is generally computed for all pixel in the images and requires relevant computation times. We have overcome the problem related to slow computation by estimating optical flow 3D only for salient points, namely points associated with a person moving in the scene. Lastly, the lack of a learning-based classification method in [16, 17] diverges from our intention to classify actions by using GMM, Gaussian Mixture Models. At last, in Table 2.1 we propose a structured comparison between our work and all the works that have been cited in this section.

2.3 Comparison between different approaches

Currently, standard testing modes are missing in the human action recognition field. Establishing common test beds to all researchers is a prerequisite in order to compare different algorithms and evaluate research progress. Recently, a certain number of datasets has been released by research groups at the cutting-edge of the action recognition field, and this seems encouraging. Furthermore, new publications are expected to report experimental results on these datasets in order to help algorithmic comparisons. Video datasets released to the public are commonly referred to as **publicly available datasets**. However, most of the publicly available datasets is dated. Indeed, these datasets are generally composed by video clips representing simple actions and generally, only recognition from 2D images can be achieved. Nowadays, there are still to few datasets that can allow researchers to evaluate algorithmic performances for complex actions or activities, and much more remains to be done in this direction. Besides, for what concerns recognition from RGB-D images, there is a complete lack of datasets on which pioneering researchers can base their work. We really faced with this problem: there is not a publicly available dataset with the RGB image information aligned with depths information that allows us to recognize the actions we want to recognize. So, in order to build a recognition system we have realized our own dataset, and the decision about making it publicly available has been left to the future.

Table 2.1: Comparison between our work and works at the state of the art of human action recognition from RGB-D images.

Work	**Recognition Task**	**Application Fields**	**Features**	**Classifiers**
Li et al. [25]	Action Recognition	Gaming	Salient postures	GMM
Sung et al. [40, 41]	Activity Recognition	Service robotics	Body joints	SVM, MEMMs
Zhang et al. [50]	Activity Recognition	Surveillance HCI	4D spatio-temp. features	LDA with Gibbs sampling
Popa et al. [31]	Action Recognition	Video surveillance	Silhouettes	SVM, k-NN, LDC[6], HMMs
Ni et al. [28]	Activity Recognition	Service robotics	DL MC STIPs[7] 3D MHIs[8]	SVM with χ-square and RBF kernels
Holte et al. [16, 17]	Gesture Recognition	HCI	3D optical flow	No learning, Edit Distance classifier
Our work	Action Recognition	Video surveillance	Real 3D optical flow	Nearest Neighbor GMM, expected

2.3.1 Publicly available datasets with only RGB information

Some popular public datasets that allow only recognition from RGB images are cited below:

- Weizmann Action Dataset[9] [14]
- KTH Action Dataset[10] [38]
- UCF Sports Action Dataset[11] [34]

[4]LDC, Linear Discriminant Classifier.
[5]DL MC STIPs, Depth-Layered Multi-Channel Spatio-Temporal Interest Points.
[6]3D MHIs, Three-Dimensional Motion History Images.
[9]See the Weizmann Action Dataset website.
[10]See the KTH Action Dataset website.
[11]See the UCF Sports Action Dataset website.

- Hollywood Human Action Dataset[12] [24]
- Hollywood2 Human Action Dataset[13] [27]
- Virat Video Dataset[14] [30]
- INRIA XMAS multi-view dataset[15] [45]

In Table 2.2 we propose a structured comparison between these datasets. For each dataset, the number and the resolution of existing video samples, as well as the number and the class of the actions represented by the samples in the dataset are reported.

2.3.2 Color-depths publicly available datasets

Just recently, new inexpensive sensors have appeared in the technological scene. This is the case of Microsoft Kinect (see Section 3.2). The global diffusion of Microsoft Kinect has excited the researchers' curiosity about working not only with RGB information but also with aligned depths information. This way, new works that achieve recognition from RGB-D frames have emerged (see Section 2.2). At the same time a new need has arisen: the construction of new datasets in which the RGB stream is aligned with the depth stream. Currently, only two of such datasets have been released:

- Indoor Activity Database [40]
- RGBD-HuDaAct Database [28]

In Table 2.3 we propose a structured comparison between these two datasets. For each dataset, the number and the resolution of existing video samples, as well as the number and the class of the actions represented by the samples in the dataset are reported.

Both these two datasets are targeted to recognition tasks in indoor environments. Furthermore, both these works aim to realize applications in the field of personal or service robotics. In Table 2.4 further information about Indoor Activity Database and RGBD-HuDaAct are provided, each field of the table is self-explanatory.

[12]See the Hollywood Human Action Dataset website.
[13]See the Hollywood2 Human Action Dataset website.
[14]See the Virat Video Dataset website.
[15]See the INRIA XMAS multi-view dataset website.

Table 2.2: Comparison between the most important publicly available datasets in which only RGB information is provided.

Dataset	**Resolution (pixel)**	**Number of video samples**	**Number of actions**	**Action classes**
Weizmann Action Dataset	180x144	90	10	*running*, *walking* *skipping*, *bending* *jumping-jack* *jumping forward* *jumping in place* *gallopsideways* *waving with two hands* *waving with one hand*
KTH Action Action Dataset	160x120	2391	6	*walking*, *jogging* *running*, *boxing* *hand waving* *hand clapping*
UCF Sport Action Dataset	720x480	197	10	*diving*, *golf swinging* *kicking*, *lifting* *horseback riding* *running*, *skating* *swinging*, *walking* *pole vaulting*
Hollywood Human Action Dataset	540x240 on average	About 700	8	*answering the phone* *getting out of a car* *handshaking* *standing up* *hugging*, *kissing*
Hollywood2 Human Action Dataset	600x450	3669	12	*answering the phone* *driving a car*, *eating* *fighting*, *handshaking* *getting out of a car* *hugging*, *kissing* *running*, *sitting down* *sitting up*, *standing up*
Virat Video 2.0 release	1920x1080	N/A	23	Some of them are: *walking*, *running* *standing*, *throwing* *gesturing*, *carrying* *loitering*, *picking up* *get into a vehicle* *get out of a vehicle* *opening or closing trunk* *unloading*, *dropping off*
INRIA XMAS multi-view Dataset	390x291	429	13	*checking the watch* *crossing arms* *scratching head* *sitting down*, *getting up* *turning around*, *walking* *waving punching*, *kicking* *pointing*, *picking up* *throwing (over head)* *throwing (from bottom)*

Table 2.3: Comparison between the existing publicly available datasets in which RGB-D data are provided.

Dataset	**Resolution (pixel)**	**Number of video samples**	**Number of actions**	**Action classes**
Indoor Activity Dataset	640x480	N/A	12	*brushing teeth* *cooking (stirring)* *writing on keyboard* *working on computer* *talking on the phone* *wearing contact lenses* *relaxing on chair* *opening a pill container* *drinking water* *cooking (chopping)* *talking on a chair* *rinsing mouth with water*
RGBD-HuDaAct	640x480	1189	6	*making a phone call* *mopping the floor* *entering the room* *exiting the room* *going to bed* *getting up* *eating meal* *drinking water* *sitting down* *standing up* *taking off the jacket* *putting on the jacket*

Table 2.4: Characteristics of the existing RGB-D publicly available datasets.

Dataset	**Number of different actors**	**Cluttered background**	**Different environments**
Indoor Activity Dataset	4	No	Yes, 5
RGBD-HuDaAct	30	No	No

Chapter 3

System baselines

This chapter provides an overview of the fundamental components on which the recognition system has been built. The discussion is accompanied by specific remarks related to the implementation of the system.

In order to build a recognition system, the first step involves the definition of the features to be extracted. As described in Section 2.1 and 2.2, it is possible to choose features that require or not a preventive tracking or background subtraction. For our work, we could exploit a people tracking system based on RGB-D data. This system is described in [2, 13] and it has been developed internally at the IAS-Lab[1] in University of Padua. Then, we were faced with a twofold challenge: exploiting the existing tracking system while computing effective features to recognize human actions. We finally succeed by finding a new method to compute features belonging to the category of global representations. The computed features are described in Chapter 5, whereas in this chapter a brief overview of the recognition system components is given, together with an overview of the underlying tracking system.

3.1 Tracking system

The underlying tracking system implements in the C++ programming language what is known in literature as tracking by detection. Furthermore, this system is designed to work on mobile robots as well. Complete information about the tracking system we used can be found in [2, 13], however we think it is useful to report some of them also in this work. The tracking system is view here as a

[1]See `http://robotics.dei.unipd.it/`.

baseline system that helps us to realize the recognition system. The approach used to achieve an effective people tracker involves the continuous application of a detection algorithm in individual frames and the association of detections across frames. Therefore, the entire system can substantially be divided into two main steps:

1. the detection phase;
2. the tracking phase.

The detection phase is devoted to divide the scene in clusters. Then, specific features are computed for each cluster: these features are necessary for the subsequent tracking phase. This stage coincides with the detection phase of our recognition system. We need to extract a certain type of features in order to recognize actions, and the detector of the underlying tracking system already gives us what we need. Specifically, we used clusters representing people (namely clusters that have already been evaluated) as raw data to be given as input at our features calculator. In this case, each cluster is merely a point cloud representing a single person in the scene. The results of the detection phase, both for the tracking system and the recognition system, are finally sent to the tracker.

The tracking step takes the output given by the detection phase and tries to assign each person detected to the correct track. The way the tracker performs this work does not regard this discussion. However it is interesting to see the role of the tracker in our recognition application. It is important to remember that we want to recognize the action performed by each person in the scene at a certain time (see Section 1.3). In order to achieve this goal, we have to associate each "clustered" cloud coming from the detection phase to the correct track. Once we have made this association, it is possible to extract features from the clustered cloud for each track (namely person) at each frame and so it is possible to recognize the action performed by each person in the scene. The tracking has been implemented so that it is possible to manage each single track, but it is also possible to handle all the detected tracks together. This is a crucial point: in this work the majority of operations are performed at the level in which we can handle single tracks. Indeed, the features extraction and computation, the descriptors computation and the classification step are all referred to this level. The stage at which we can handle all the detected tracks is devoted to message passing, to realize visualization objectives and to perform what is computed at the level previously described. Finally, the tracking system is a real-time system and we really did not want to lose this great property. Therefore we decided to build a recognition system which is also real-time. Being able to satisfy our real-time constraints is not as easy as it could sound: sometimes we had to use small tricks to make computing feasible at a good frame rate.

The articulated structure of the tracking system allows us to see the potential inside our recognition application. In fact, if the classification step and some portion of the features extraction step are performed at the stage in which all the detected tracks can be handled together, also actions involving more than one person can be recognized. However, this enhancement of the system is left to future developments (see Chapter 7).

3.2 System components

It makes sense that if the recognition system is built on an existing tracking system, the components of the tracking system are also components of the recognition system. However, we believe useful to report the essential elements of the recognition system in this discussion.

3.2.1 Microsoft Kinect

Nowadays it is possible to develop an entire recognition system based on data acquisition by means of cheap sensors. An example of such a sensor is Microsoft Kinect. The Kinect sensor is distributed by Microsoft for the Xbox 360 video game console but it has recently gained in popularity also in the computer vision field. We used Microsoft Kinect in our system and we are currently among few pioneering works in this direction. This device consists of an RGB camera, an infrared structured light source for inferring depths and a multi-array microphone running a proprietary software. As a consequence, Microsoft Kinect returns an RGB video stream with aligned depth at each frame as output. The RGB video stream uses 8-bit VGA resolution (640 x 480) pixels with a Bayer color filter, while the monochrome depth sensor outputs a video stream at VGA resolution as the RGB stream, but with 2048 levels of sensitivity (corresponding to 11 bits). It has been estimated that Kinect sensor outputs videos at a frame rate of 30Hz. Furthermore, the small size of Microsoft Kinect makes it suitable to be mounted on mobile ground robots such as Pioneer 3-AT[2].

3.2.2 ROS - Robot Operating System

The recognition system has been designed in C++ programming language to run entirely with ROS. ROS stands for Robot Operating System but it do not represent an actual operating system according to the common meaning of these words.

[2]The PIONEER 3-AT is a highly versatile four wheel drive robotic platform. All information about it can be found at `http://www.mobilerobots.com/researchrobots/p3at.aspx`.

Instead ROS is better defined as a framework: it provides libraries, hardware abstraction, device drivers, visualizers, message passing, and more other functionalities in order to help the development of robot applications. Furthermore, it is geared toward a Unix-like system: currently only Ubuntu Linux is supported while other variants such as Fedora and Mac OS X are considered experimental. Originally realized at Standford Artificial Intelligence Laboratory, now ROS is being developed at Willow Garage[3], a robotics research lab devoted to the development of hardware and open source software for personal robotics applications.

ROS looks like an appealing framework to research teams because it provides a lot of utilities while being free for research use. This work is based on ROS and it has taken advantage of a lot of its functionalities. Only some of which are: the message passing, the computation by means of graph architectures and the visualizer rviz. It is necessary to say something more about graph architectures, because they really are the "bearing walls" of our application. In ROS graph architectures, processing takes place in nodes and each of these nodes may receive, post and multiplex sensor, control, state, planning, actuator and other messages. ROS nodes are simply executables that uses ROS to communicate with other nodes. This communication is realized by means of message exchanging: in order to send a message to other nodes, each node has to publish messages to a topic, whereas it has to subscribe to a topic to receive messages from other nodes. Communication between nodes can also be achieved by exploiting RPC services or the Parameter Sever functionalities. More information about this and other arguments can be found at the official ROS website[4].

There are two possible ROS distributions available to users: the stable version and the unstable version. The stable release of ROS is ensured to be working, but it may typically contain the old versions of certain libraries. It is therefore recommended to always use the stable version. However, it can happen a developer needs a function contained only in the cutting-edge version of a specific library. In the last case, the developer has to work with the unstable distribution of ROS. The latter is a rolling distribution that targets the next ROS distribution release. In the unstable version, updated libraries can be found but it can happen that something doesn't work because it is still under development. We have used the ROS stable version for what concerns the ramp-up phase of this work (see Chapter 4), whereas we have needed the ROS unstable version in order to compute the new features we propose in real-time (see Chapter 5).

[3]See `http://www.willowgarage.com/`.

[4]See `http://www.ros.org/wiki/`.

Chapter 4

Action recognition with global cues

In the start-up phase of this work, a simple recognition system has been developed in order to take confidence with the underlying tracking system and see the real potential of a future recognition system. This chapter covers all the details about this initial recognition system, including which type of features are used and which classification method is exploited.

Before developing a robust recognition system, we decided to realize a simple recognition system in which ad hoc rules are used to distinguish actions from each other. An overview of this simple recognition system is shown in Figure 4.1. In this system, we exploit some simple features coming directly from the tracking-by-detection system and we investigate about which actions could be recognized using this approach. Since these features are not so discriminant, we did not choose to extend this approach to a learning-based classification method. Thanks to the knowledge acquired during this phase of the work, at a later stage we succeed in developing an innovative recognition system (see Chapter 5 and Chapter 6).

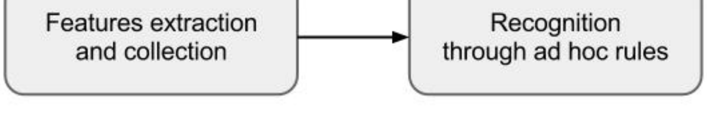

Features extraction and collection
Recognition through ad hoc rules

Figure 4.1: Overview of the recognition approach in which global cues are exploited.

The remainder of this chapter is organized as follows: Section 4.1 briefly explains the type of features we used to create a basic recognition system, Section 4.2 describes how the recognition is performed, while Section 4.3 reports the experimental results obtained while testing the program on a dedicated dataset.

4.1 Extracting simple features

One of our aims (see Section 1.3) is to recognize actions executed by each person present in the scene at a certain time. To achieve this objective, we had to associate each track with its current set of features. However, the tracking system already gave us what we needeed: the desired features associated with each individual track. Specifically, these features are the:

- 3D geometric coordinates of the centroid of each person in the scene;
- bounding box proportions of each person in the scene.

We refer to the 3D geometric coordinates of the centroid as the x, y and z coordinates at which the centroid is located in the real world, whereas we refer to the bounding box proportions as the width and the height of the bounding box in the camera reference system. Starting from these features, we decided to add a new one: the 3D velocity vector of the centroid for people in the scene. Table 4.1 shortly represents the features used at this stage of the work.

Features given by the tracking system	**Features computed**
- 3D geometric coordinates of the centroid	3D velocity vector of the centroid
- Bounding box proportions	

Table 4.1: The table shows the features used as global cues by the recognition system. The left column lists the features given by the tracking system, while the right column shows the features computed from those on the left.

The geometric coordinates of the centroid and the bounding box proportions are initially collected in vectors for each track and at each frame. In other words, for each track there are two vectors: one storing the centroid coordinates and one storing the bounding box proportions. Obviously, the two vectors are aligned: this mean that if the object A is collected at frame F and it is stored in a vector at the position K, then also the other vector will store the object B, different from A, at the position K (if it is collected at frame F). Furthermore, while developing a collector for the chosen features, a more extended approach was used: instead of storing only the geometric coordinates related to the centroid, it is possible to store geometric coordinates related to every desidered *key point*. Different key points are associated with different IDs, so that it is possible to refer to key points of the same type without any problem. Although this extension was available, it was not exploited because we decided to store only the 3D geometric coordinates of the centroid. Indeed, 3D trajectories of the centroids have already demonstrated to be discriminant features in human action recognition, e.g. in [5] these features are exploited to recognize human behaviour models in smart home enviroments. Once the collection of 3D centroids is started, the next step regards the computation of the 3D velocity vectors associated with them.

4.1.1 Velocity Estimation

We developed two different methods to calculate velocity vectors: these methods differ each other in the way they look backward for the previous centroid that has to be used in the calculus of the velocity vector. The first method allows us to search backward in terms of the number of frames, while the second one enables searches based on seconds of time. Furthermore, both these methods take as input the current centroid and, respectively, the number of frames in the former case, or the amount of time in the latter case, of which we have to go back to pick up the previous centroid. When the previous centroid is retrieved, the spatial difference and the time difference related to this pair of centroids are computed. Then, the spatial difference is divided by the temporal difference and the 3D velocity vector is finally returned. Concurrently with the collection of 3D geometric coordinates of the centroid, the 3D velocity vector is computed at each frame. It is clear that some time is needed before the computation of the velocity vectors becomes effective and this portion of time directly depends on the setting of the "backward search" described so far.

In addition, we decided to display the 3D velocity vector in the RGB visual image in order to verify the correctness of the computations outlined above. We have accomplished the visualization in more than one step:

1. at each frame, the 3D velocity vector related to the current centroid is retrieved;

2. the 3D velocity vector is then normalized;

3. the final point of the arrow to be displayed is computed by exploiting the 3D geometric coordinates of the current centroid and the normalized velocity vector;

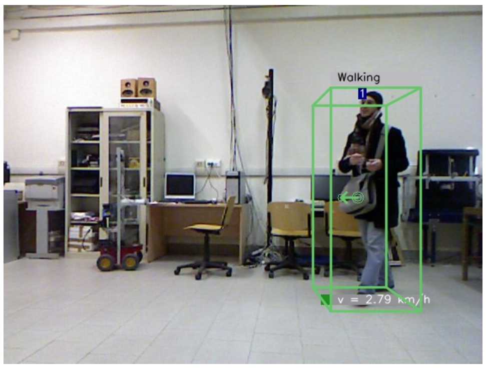
4. 3D geometric coordinates of the final point are converted to the camera reference system;

5. finally, an arrow is visualized between the current centroid and the so called final point of the arrow.

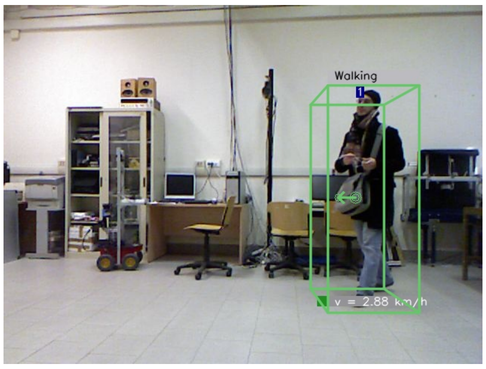
Point 1 and point 5 are self-explanatory, while point 2 is accomplished in order to visualize an arrow that keeps always the same lenght even if the velocity vector changes frame by frame. In point 3 we used a simple a simple formula to compute the final point:

$$\mathit{final\_point} = \mathit{current\_centroid} + \mathit{normalized\_velocity\_vector} \times \mathit{current\_time\_difference}$$

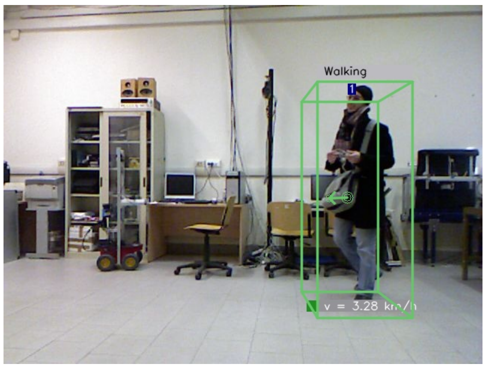
and the successive point 4 is necessary because the final point obtained in 3 is expressed in the real world coordinates system. Eventually, Figure 4.2 shows four frames in which the velocity vector is displayed while a person is moving in a lab environment.



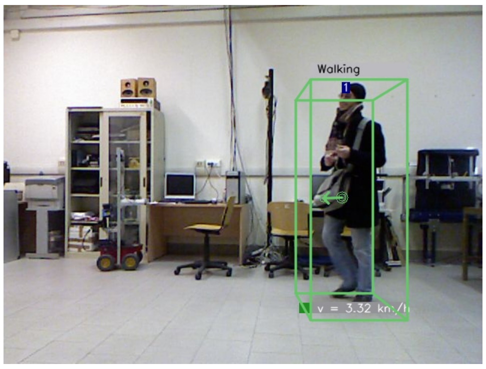
(a) First frame.



(b) Second frame.



(c) Third frame.



(d) Fourth frame.

Figure 4.2: Display of the velocity vector associated to the centroid of a person. Four consecutive frames are shown.

4.2 Discerning between actions

As stated in the introduction of this chapter, the recognition is realized here by means of ad hoc rules. For each specific action class, rules are used to determine if the action into consideration is taking place at a certain time instant. Since the features used to discern between actions are really limited, we had to investigate about which type of actions could potentially be recognized by following this basic approach. The result was that:

- actions like *standing*, *sitting down*, *standing up*, *crouching* (or *bending*) could ideally be recognized observing the movement of the centroid along the vertical axis;
- actions like *walking*, *jogging*, *running*, could ideally be recognized observing the norm of the current 3D velocity vector associated with the centroid of a person;
- finally, actions like *boxing* or *hand waving* could not be recognized with our simple approach, because there is no way to identify the movement performed by single parts of the human body.

Therefore, we decided to concentrate on the following actions: *standing*, *sitting down*, *standing up*, *crouching*, *walking*, jogging and *running*. Recognizing actions like *walking*, *jogging*, *running* is a challenging task due to their intra-class variations: in this case the speed at which an actor is moving strongly depends on the characteristics of the actor, so for example an actor may “run” at the same speed at which another actor is simply “jogging”. Table 4.2 shows the actions we aimed to recognize along with the features we used in order to achieve this objective. We started developing the ad hoc rules for the *standing*, *walking*, *jogging* and *running* actions and then we continued by adding more actions beyond these three ones. Firstly, rules associated with *standing*, *walking*, *jogging* and *running* took into account only the current speed of the centroid, but when new actions were added to these initial ones, it became necessary to add other parameters in order to avoid overlapping between actions. Specifically, we said that:

- A person is **standing** if the current speed of his/her centroid is less than 0.2 km/h, the centroid itself is not lowering or rising, and he/she is not sitting down currently.
- A person is **crouching** if his/her centroid is lowering by an amount greater than a predefined constant threshold at each frame. Furthermore the person has not to be already sitting.

Action	**Raw features used**
Standing	- 3D geometric coordinates of the centroid - 3D velocity of the centroid (smaller than 0.2 km/h)
Crouching	- 3D geometric coordinates of the centroid
Standing up	- 3D geometric coordinates of the centroid
Sitting down	- 3D geometric coordinates of the centroid - bounding box proportions (width - height)
Walking	- 3D geometric coordinates of the centroid - 3D velocity of the centroid (between 0.2 km/h and 6.0 km/h)
Jogging	- 3D geometric coordinates of the centroid - 3D velocity of the centroid (between 6.0 km/h and 8.0 km/h)
Running	- 3D geometric coordinates of the centroid - 3D velocity of the centroid (greater than 8.0 km/h)

Table 4.2: The table shows the actions recognized by the basic recognition system together with the features used to develop ad hoc rules in order to discern between the outlined actions. Further details are provided in the discussion.

- A person is **standing up** if his/her centroid is rising by an amount greater than a predefined constant threshold at each frame. The person could be previously sitting or not.
- A person is **walking** if the current speed of his/her centroid is between 0.2 and 6.0 km/h, the person is not lowering or rising, and he/she is not sitting down currently.
- A person is **jogging** if the current speed of his/her centroid is between 6.0 and 8.0 km/h, the person is not lowering or rising, and he/she is not sitting down currently.
- A person is **running** if the current speed of his/her centroid is greater than 8.0 km/h, the person is not lowering or rising and he/she is not sitting down currently.

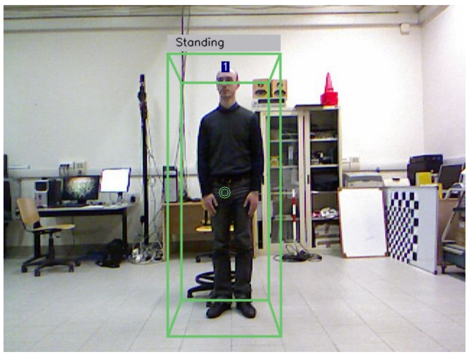
We set the thresholds related to the velocity of the centroid according to the conventional wisdom, whereas the thresholds related to lowerings and risings are set according to experimental evaluations. Furthermore, we observed that by adding more and more actions, the recognition task, as realized here, becomes more complicated due to possible overlapping actions. Let us consider two overlapping actions as two actions for which the associated ah hoc rules return "true" in both cases. When the output of two distinct rules is "true", the system is ideally recognizing two different actions at the same time, but this can not occur. If more than one discriminant rule gets a "true" as output value, it will be necessary to modify most of the rules present in the system in order to make them more discriminant. Obviously, this process is intended to produce increasingly detailed rules without the assurance that at the next step other actions will not overlap. All these considerations prove that such a basic approach to the recognition is limited and it is not scalable.

4.3 Experimental results

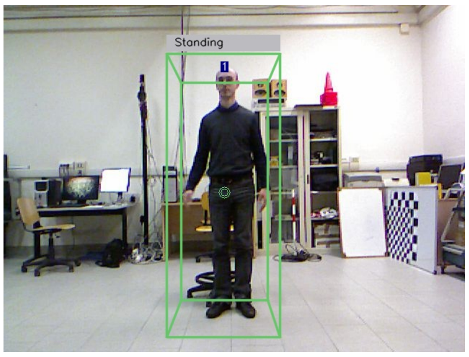
First, we tested the recognition system with global cues on-line. In this context, on-line means that the Microsoft Kinect is connected while the program is executing, and it sends the RGB-D stream real-time to the application. This live execution enabled us to properly estimate thresholds related to ad hoc rules (see Section 4.2). A remarkable observation coming from these first experiments regards the frame rate of the application. Indeed there is an important issue: if ad hoc rules are estimated while working on a personal computer which can perform only at low frame rates, and successively, the application is executed on a personal computer working at high frame rates, the thresholds will not be well estimated (also the vice versa holds). Furthermore, such "wrong" thresholds will lead to a wrong action recognition.

Secondly, we tested the system in the context of Section 6.2.1. Despite we created the dataset described in Section 6.2.1 for other purposes, we felt that it could be an interesting test bed also when applied to the recognition system based on global cues. In this section, each video sample, namely each recording, related to a different individual is entirely given as input to the system and the results are observed in the perspective of continuous recognition. Figures 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8 show the result of the recognition system based on global cues for each action of the dataset defined in 6.2.1. For each action class two frames are provided. Figures 4.4 and 4.7 confirm what was stated in Section 4.2: the system is not able to recognize actions in which only the movement of single body parts is involved. While, Figure 4.5 show that also the action *crouching* could be recognized by our simple system. Indeed, the action *crouching* is rightly inferred while the actor is bending but he is not already sitting down.

Finally, we provide final considerations about the continuous recognition task and possible developments of this basic system. In general, the system is not scalable with regards to the number of actions that can be recognized and it is not error-free when it attempts to infer the action for a piece of video stream. However, the continuous recognition is performed in a fairly good way, even if it occurs that the beginning and the end of an action are not always properly identified. Limits and weaknesses of the system, outlined in this discussion and in Section 4.2, led us to the design of a new recognition system in which more sophisticated features are used together with a learning-based classification algorithm.

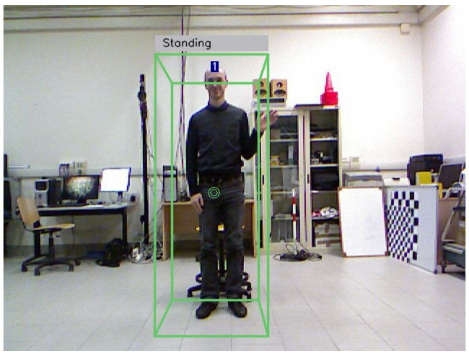


(a) First frame.

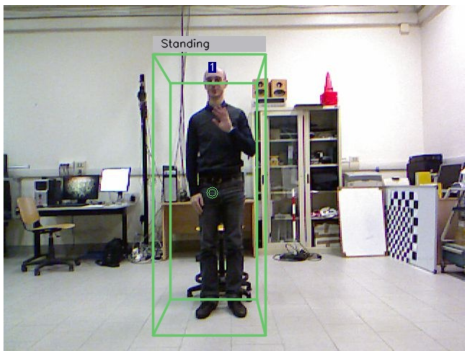


(b) Second frame.

Figure 4.3: Recognition results of the system based on global cues for the action *standing* taken from the dataset defined in Section 6.2.1.

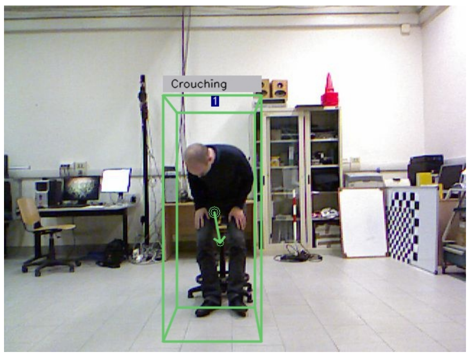


(a) First frame.

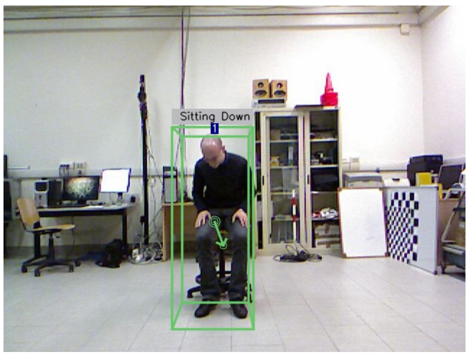


(b) Second frame.

Figure 4.4: Recognition results of the system based on global cues for the action *hand waving* taken from the dataset defined in Section 6.2.1.

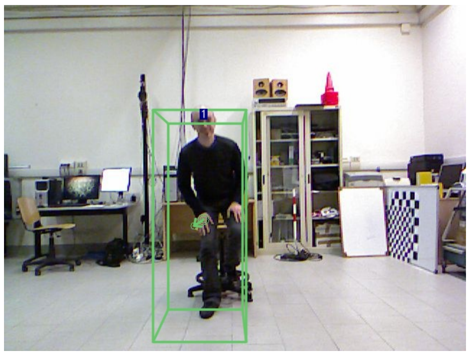


(a) First frame.

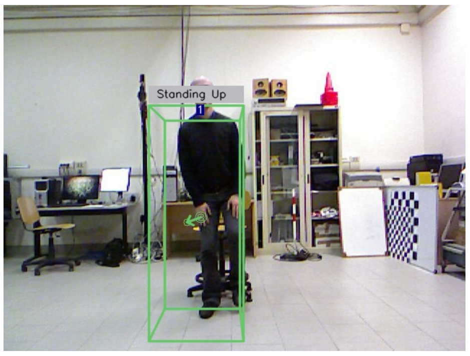


(b) Second frame.

Figure 4.5: Recognition results of the system based on global cues for the action *sitting down* taken from the dataset defined in Section 6.2.1.

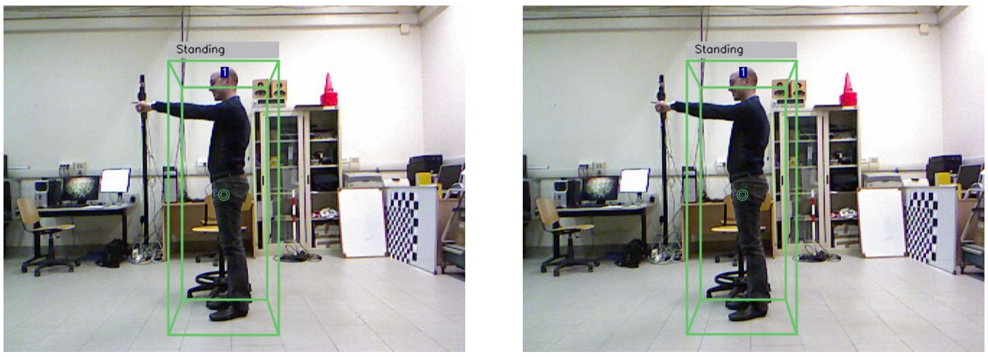


(a) First frame.



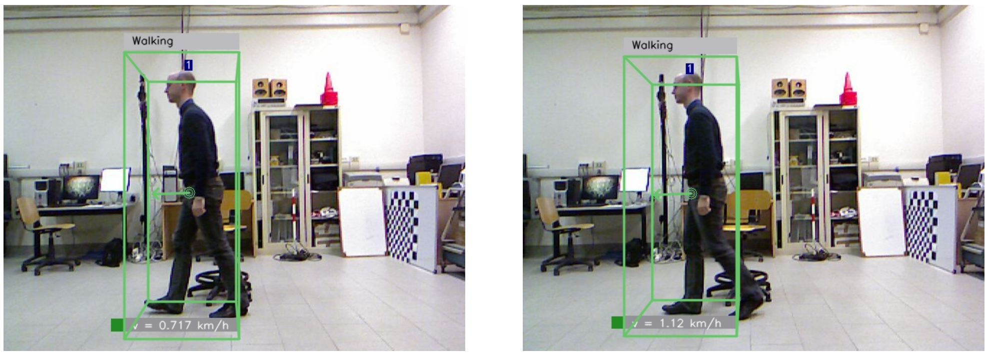
(b) Second frame.

Figure 4.6: Recognition results of the system based on global cues for the action *getting up* taken from the dataset defined in Section 6.2.1.



(a) First frame. (b) Second frame.

Figure 4.7: Recognition results of the system based on global cues for the action *pointing* taken from the dataset defined in Section 6.2.1.



(a) First frame. (b) Second frame.

Figure 4.8: Recognition results of the system based on global cues for the action *walking* taken from the dataset defined in Section 6.2.1.

Chapter 5

Detecting motion from point cloud XYZRGB data

"Using the right tool for the right job" represents the real essence of this chapter. The right tool for us is Point Cloud Library, while the right job regards the computation of 3D optical flow starting from point cloud data representing a person. Therefore this chapter firstly gives an overview of the Point Cloud Library and its tools, and then it goes into the deep about the issue of computing optical flow 3D from point cloud data only.

Poor results and limitations coming from the recognition approach in which global cues are exploited (see Chapter 4), encouraged us to create an enhanced recognition system. This new approach to human action recognition consists of three major steps:

1. the features extraction, in which we propose a novel approach to compute useful features to recognize actions;
2. the descriptors computation, in which features computed at the previous phase are conveniently summarized in a data structure, the so called descriptor;
3. the classification, in which previously unseen temporal sequences of descriptors are classified following a learning-based approach.

The point 1 is discussed in this chapter, while point 2 and and 3 are described in Chapter 6. Furthermore, a comparison between the approach to recognition based on global cues and the one proposed in this chapter is shown in Figure 5.1. As briefly seen in Section 1.2, features are powerful discriminants in human action

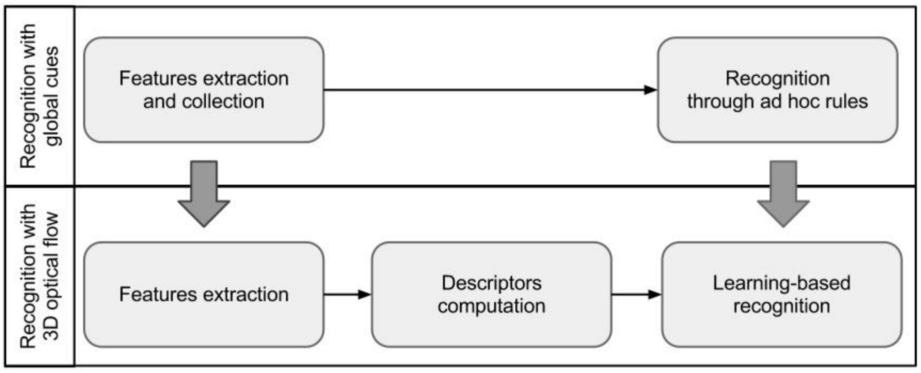

Recognition with global cues
Features extraction and collection
Recognition through ad hoc rules
Recognition with 3D optical flow
Features extraction
Descriptors computation
Learning-based recognition

Figure 5.1: Comparison between recognition approaches: the overview of the recognition with global cues is matched against the overview of the recognition based on 3D optical flow computation.

recognition and this work is focused on finding new features and effective ways to compute them. In particular, we propose a new way to compute 3D optical flow, namely the extension to the third dimension of the traditional 2D optical flow used to recognize actions from 2D video images (see Section 2.1). Furthermore, our approach is able to perform real-time computations for 3D optical flow, whereas most of the works based on computing 2D optical flow has not real-time performances. Indeed, computing optical flow is really a challenging problem. The approach we followed in order to achieve real-time performances aims to compute the velocity vectors of the 3D optical flow only for the portions that represent people in the scene. More specifically, each person is associated with a point cloud and 3D velocity vectors for the points of each cloud are calculated. While aiming to compute velocity vectors a new issue arose: the problem of finding corresponding points between a specific cloud in a current frame and the same cloud in a past frame. This problem has been solved exploiting the geometric coordinates of points together with the color information and is addressed in Section 5.3.1. The same section provides also further details about the calculation of 3D optical flow, whereas the earlier sections, 5.1 and 5.2, describe respectively the tools we used to compute the 3D optical flow and the first approach to the problem. The chapter ends with Section 5.4, in which a brief discussion about the performances of our approach to 3D optical flow is provided, together with a comparison with 2D traditional approaches.

5.1 Dealing with point clouds

A point cloud is defined as a data structure representing a collection of multi-dimensional points. When dealing with a 3D point cloud, points are commonly represented by the x, y and z geometric coordinates of an underlying sampled surface. In order to compute the optical flow 3D, we needed to deal with 4D point clouds, namely point clouds in which points are characterized by having a fourth field representing the color component. Past methods to compute optical flow 2D relied on pixel color intensity. Intensity was used to compute image derivatives in order to find correspondences between pixels in the past and the current frame. Similarly, we exploit the RGB color information to direct a search that aims to find correspondences between points in each past and current point cloud representing a person. As seen in Section 3.1, the recognition system is based on an existing tracking system. Originally, this tracking system accessed to only depth information from Microsoft Kinect sensor, so that point clouds were all 3-dimensional (i.e. each cloud associated with a person was uncolored). The first step needed to compute optical flow 3D has involved the addition of the color information to the tracking system, so that colored clouds for people in the scene could be available at any time. Next steps are described in Section 5.2 and 5.3, while this section is focused on providing an overview of Point Cloud Library and its data structures. We used the PCL 1.4 version, provided in the unstable ROS branch (see Section 5.3). The stable version of ROS includes PCL 1.1 version in which data structures are not fully working and a lot of bugs are not fixed. Advanced data structures are used to efficiently perform computations on point clouds and they proved to be decisive on realizing real-time operations in this work, therefore proving PCL 1.4 version has been a good choice.

5.1.1 PCL - Point Cloud Library 1.4

The Point Cloud Library [36], abbreviated in PCL, is a standalone, large-scale open source project for n-dimensional point cloud processing, with special focus on 3D geometry processing. The PCL framework contains various algorithms at the state of the art of filtering, feature estimation, surface reconstruction, point cloud registration, model fitting and segmentation. These algorithms make a number of functions available to users, including functions devoted to:

- filtering outliers from noisy data;
- stitching 3D point clouds together;
- segmenting salient parts of a scene;

- extracting keypoints and computing associated descriptors in order to recognize objects based on their geometric appearance;
- estimating surface normals and reconstructing object surfaces;
- visualizing point clouds at high frame rate.

Furthermore PCL is released under the terms of the BSD license and it is free for commercial and research use. Unlike other open source libraries, PCL is really well supported by a community of reference developers. Engineers and scientists from all over the world are currently contributing to the development of this framework, though the majority of them is concentrated in U.S.A. and Europe. PCL has also a rich community of users that can take contact with developers by means of the users forum[1]. The users forum is a great communication system that helps users to solve problems and also helps developers to find new bugs to be fixed. During the development of this thesis work, we have relied on this forum for what concern challenging point clouds processing and so we could really see the hidden potential in the community of developers and users. Finally, the PCL project is also well financially supported: large companies such as Google, NVidia, Toyota and Urban Robotics are active financiers.

5.1.2 Data Structures

In PCL, the basic data structure is a point cloud. There are actually two types of point clouds:

- unorganized point clouds;
- organized point clouds.

Organized point clouds resemble an organized image structure, where data is split into rows and columns. Clouds coming from Time of Flight cameras are examples of such organized clouds. The advantages of dealing with organized point clouds is that nearest neighbor searches can become much more efficient thanks to the possibility of exploiting relationships between adjacent points. By contrast, unorganized point clouds are simply collections of points with no structure. However, PCL provides advanced data structures by means of which it is possible to define a structure for unorganized point clouds. By using these advanced data structures nearest neighbor searches become effective as in the organized case. In this work, we have dealt with unorganized point clouds coming from the underlying tracking-by-detection system and we have exploited advanced data structures to

[1]See `http://www.pcl-users.org/`.

force a structure on these clouds. PCL provides two advanced data structures, **kdtree** and **octree**, which both are tree-based structures and have been used in this work. A k-dimensional tree, or kdtree, is a space partitioning data structure that stores a collection of k-dimensional points in a tree structure in which FLANN searches are enabled. FLANN stands for *Fast Library for Approximate Nearest Neighbors* and it is a library that allows for fast nearest neighbor searches in high dimensional spaces. In this work, k-dimensional trees have been used while inquiring about possible ways for matching point clouds coming from the current and past frame (see Section 5.3.1). To achieve this objective, also octree structures have been used. An octree is a hierarchical tree data structure in which each node has either eight children or no children. The root node identifies a cubic bounding box which encapsulate all points, furthermore at every tree level the space becomes partitioned by a factor of 2 and this results in an increased voxel resolution. Octrees, as well as kdtrees, enable for fast nearest neighbor searches including:

- radius searches
- k-nearest neighbor searches
- neighbors within voxel search.

Radius searches and neighbors within voxel searches return all neighbors of a query point that are within a given radius and a given voxel respectively, while k-nearest neighbor searches return the k nearest neighbors of a query point. Fast searches are the core operations that enable our system to perform real-time computations and octree has proved to be a data structure well suited for our purposes. Indeed, we have also used octrees to implement change detection (see Section 5.2) and correspondence rejection methods (see Section 5.3.2).

5.2 Change Detection

First, we approached the problem of detecting motion from point cloud data by exploiting a tool available in PCL (1.2 version or higher). This tool is represented by a particular octree data structure that allows for spatial change detection on unorganized point clouds. The octree implementation for detecting spatial changes takes two clouds as input and returns a vector of indices as output. When the first cloud is set as input cloud, an octree is built on it and describes its distribution. Also when the second cloud is fed as input an octree is built on it, but it is important to notice that when an octree is built on the second cloud this particular octree class has to be reset. Resetting allows to keep the first octree structure in memory while dealing with the second one. This mechanism is referred to as *octree double buffering* and it is necessary to efficiently process multiple point clouds over time. Once the two clouds have been added to the octree spatial change detector, their octree structures are recursively compared in order to find differences in voxel configuration. These differences mean that spatial changes have occurred. At the end of this process, a vector of point indices is returned. These indices refer to points coming from octree voxels which did not exist in previous buffer. As an additional detail, the voxel resolution of the octree spatial change detector can be set at the time octree is created. The voxel resolution determines how many points would be stored in each of the voxels, so by changing the voxel resolution it is possible to have any number of points within a single octree node. We set the voxel resolution so as to have a single point in each octree voxel and then we added the octree point cloud change detector to our system. In order to do that, we needed to associate each track with its point cloud at each frame. This objective was accomplished by sending the right clouds via ROS message from the detector to the tracker in the underlying tracking system. Furthermore we also needed to store, for each track, the point cloud associated to it in the current and the past frame. In other words, at each frame F and for each track T, a vector stores two elements: the point cloud associated to T at frames F and $F - 1$. Finally, these two point clouds are sent as input to the octree spatial change detector and indices for the moving points are retrieved at every time. The correct use of the octree double buffering technique allows us to achieve good results while keeping real-time performances, indeed the tracking system with the add of the spatial change detector maintains its original time performances.

5.3 Computing 3D optical flow

Performing spatial change detection gave us the true insight on which this thesis work is based. As described in Section 5.2, in order to execute spatial change detection frame by frame, we had to store for each track the point cloud associated to it for the current and the past frame. At this stage, we realized that if we had found the existing correspondences between the points of the two clouds (the past and the current cloud), we could estimate the optical flow by simply calculating the 3D velocity vector for each pair of matching points. Therefore, we investigated about the possible ways in which the correspondences between points could be estimated. From the correspondence estimation problem, a new issue arose: the correspondence rejection problem. Since correspondences are “simply” estimated, some of them could represent a wrong estimation, namely the algorithm could find a matching between two points even if those points did not actually match.

The task of rejecting wrong estimations for points of two different point clouds is called *correspondence rejection*. We concentrated also on finding methods to reject wrong correspondences and finally, we developed a particular algorithm that aims to reject "bad" correspondences concurrently to the noise caused by the Microsoft Kinect sensor. In order to test the correctness of the estimation and rejection methods we decided to visualize results by means of the PCL visualizer (available from the 1.2 version).

Before going into the details of the discussion, it is important to remark that:

1. spatial change detection can or can not be applied before correspondences between points have been estimated;
2. 3D optical flow can be computed with or without the previous execution of the correspondence rejector.

As regards point 1, we initially developed a system able to estimate correspondences between points of the clouds without previously performing the spatial change detection algorithm. Applying spatial change detection could lead to some advantages and disadvantages at the same time. Advantages are:

- reduced number of points in the point cloud to be considered for further computations;
- pre-screening devoted to noise removal.

In order to achieve an actual reduced number of points we created a new cloud, starting from the current one, and then we added to this cloud all the points in the current cloud that are found to be moving by the spatial change detector. Furthermore, when dealing with Microsoft Kinect there is a bit of noise constantly present; this noise is due to the continuous mapping and remapping of the points from the camera reference system to the real world system and vice versa. The major drawback hidden inside this approach is related to the number of correspondences returned by the estimator. Indeed, if a person is currently standing (namely without making any movement), the number of output correspondences might be very low and this could lead to a loss of information potentially precious in successive steps.

As stated in point 2, once correspondences have been estimated, they could be fed as input directly to the optical flow calculator or they could first pass through a correspondence rejector. In the former case, the optical flow calculator returns the 3D optical flow computed for all the points in the current cloud that have a correspondence in previous cloud. In the latter case, a correspondence rejection method is applied just after the correspondence estimation and a new vector of matching points is returned. Obviously, the length of this vector could be ideally less than or equal to the length of the vector containing the estimated correspondences, but if it remains unchanged, then there is not any improvement with respect to the correspondence estimation step, since the two outputs are actually the same. After the correspondence rejection has been performed, the new vector of matching points is given as input to the optical flow calculator and finally the 3D optical flow is computed for all the points in the current cloud that have a correspondence in the previous cloud which is not been rejected by the application of the correspondence rejection algorithm. In Figure 5.2 a scheme of the process related to the computation of 3D optical flow is provided.

5.3.1 The problem of matching point clouds

Matching point clouds represents a central issue in this work. Indeed, the 3D optical flow calculus strictly depends on how many correspondences between points are well estimated. In this work, the tool devoted to the correspondence estimation is called *point cloud matcher*. The *point cloud matcher* takes four elements as input:

- the previous cloud to which we are referring at the current frame;
- the current cloud associated with the current frame;
- the previous vector of indices related to points in the previous cloud;
- the current vector of indices related to points in the current cloud;

and it attempts to estimate the existing correspondences between the points of the current cloud and the points of the previous cloud. Actually, the *point cloud matcher* returns a vector of correspondences between points: a correspondence is defined as a pair of matching indices, one taken from the current cloud and one from the previous cloud. In order to find pairs of matching indices, we discussed several ways that all had a common characteristic: the objective of performing searches based on the color information. Indeed, since the 2D optical flow calculus relies on the differences in pixel color intensity observed during the time evolution, we thought we could bring a similar approach to the point clouds world. Since the point clouds involved in this work are 4D point clouds (see Section 5.1) we chose to perform nearest neighbor searches driven by both the geometric coordinates of the points and the color information. First, we implemented a brute force algorithm: the current cloud is scanned and each of its points is compared against each point in the previous cloud, in order to find the match with the minimum distance. The distance is computed based on:

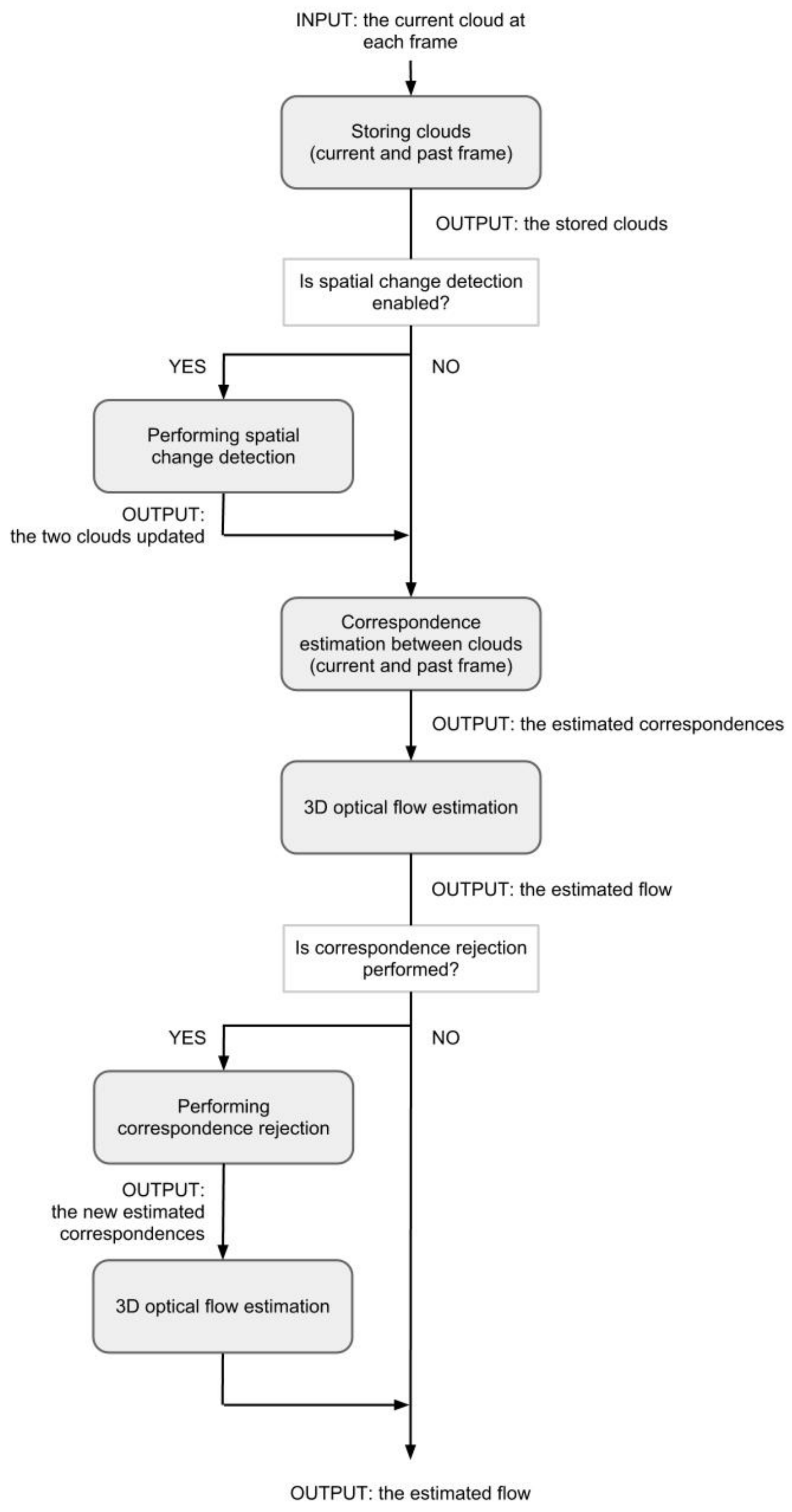

INPUT: the current cloud at each frame
Storing clouds (current and past frame)
OUTPUT: the stored clouds
Is spatial change detection enabled?
YES
NO
Performing spatial change detection
OUTPUT: the two clouds updated
Correspondence estimation between clouds (current and past frame)
OUTPUT: the estimated correspondences
3D optical flow estimation
OUTPUT: the estimated flow
Is correspondence rejection performed?
YES
NO
Performing correspondence rejection
OUTPUT: the new estimated correspondences
3D optical flow estimation
OUTPUT: the estimated flow

Figure 5.2: Overview of the computational process of 3D optical flow.

1. the geometric coordinates and the RGB color information related to the points, or
2. the geometric coordinates and the HSV color information related to the points.

As regards point 2, the RGB color information is converted to the related HSV representation by means of well-known conversion algorithms. Unfortunately, the brute force method did not estimate the correspondences between points very well and furthermore, it has demonstrated to be really high-costly in term of computation. In fact, at this level each cloud has on average 300 points, and performing such an algorithm could lead to the execution of 300 $\times$ 300 computations on average, since the brute force method is $O(N \cdot M)$ where N is the size of the current cloud and M is the size of the previous cloud. Furthermore, the $N \times M$ computations will be executed for each track and at each frame, potentially leading to a crash of the system.

To enhance the brute force algorithm, we needed an efficient tool to perform searches in point clouds and PCL was the right tool for our purposes. Furthermore, the knowledge acquired by performing spatial change detection (see Section 5.2) suggested us to use the octree data structure to perform efficient searches. Indeed, PCL provides the class *OctreePointCloudSearch* which enables to execute various kind of searches, like searches within voxels, k-nearest neighbor searches and radius searches (see Section 5.1.2). Then, we looked for a search algorithm that best suited our application and particularly we examined searches for the k-nearest neighbors at a query point and searches within a radius. In both cases the algorithm that attempts to estimate correspondences is organized as follows:

a. The cloud at the current frame is scanned, and a point belonging to it is extracted each time.

b. An octree search structure is set based on the previous cloud (associated with the current one).

c. Pretending that each point in the current cloud is a point in the previous cloud, an efficient search is performed: a current point is searched in the previous cloud each time a point is extracted.

d. Finally, a set of nearest point to the query one is retrieved.

Furthermore, searches are driven by:

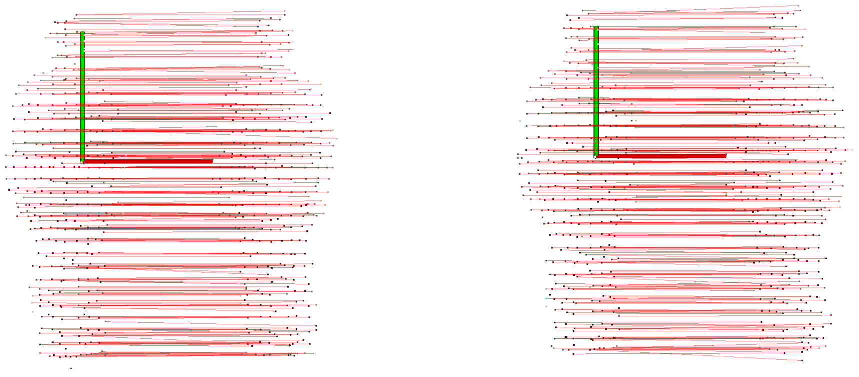
- the 3D geometric coordinates and,
- the color information expressed in the RGB representation,

so that points retrieved are really the nearest points in term of both spatial location and color. Results indicated that:

1. The k-nearest neighbor search algorithm estimates correspondences quite well, whereas the radius search algorithm do not.

2. Best results are obtained for $k = 1$.

With regard to point 1, we want to provide some further details. First, the number of correspondences returned by both algorithms is equal to the number of points in the current cloud. Since, each point in the current cloud is matched against each point in the previous cloud, it could happen that two different points are associated to the same point in the previous cloud. It is obvious that these kinds of association can not exists in the reality and there is the need of rejecting such "bad" associations (see Section 5.3.2). However, since we focused on discovering the best way to match point clouds, we did not investigate possible enhancements of those algorithms, leaving those improvements to the further developments of this work (see Chapter 7). In Figure 5.3 the correspondences estimated while a person is *standing* are shown in two successive frames. Furthermore, in Figure 5.4 we also show three successive frames in which the correspondences are estimated for a person who is *hand waving*. The left cloud is the cloud in the previous frame, while the right cloud is the cloud at the current frame and both clouds are referred to the same actor. Screenshots are obtained by means of the PCL visualizer (see Section 5.3.3).

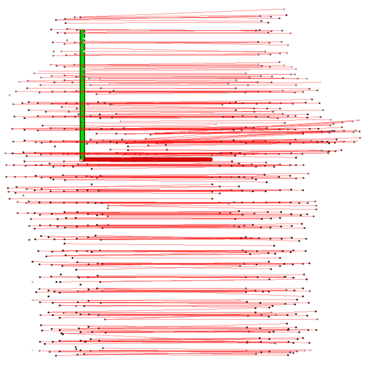


(a) First current frame to the right.

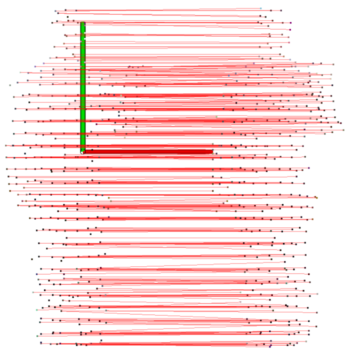
(b) Second current frame to the right.

Figure 5.3: Display of the estimated correspondences with regard to the action *standing*.

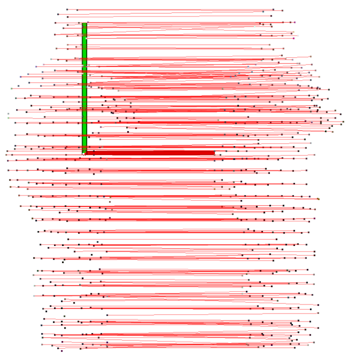
It is straightforward to see why point 2 holds. Setting k to 1 means that the algorithm will find, in the previous cloud, the nearest point to the current point under consideration at each iteration, and this completely fits our objective. If k



(a) First current frame to the right.



(b) Second current frame to the right.



(c) Third current frame to the right.

Figure 5.4: Display of the estimated correspondences with regard to the action *hand waving*.

is set to values greater than 1, then the algorithm will retrieve the farthest points with regard to both the geometric and the color space.

Real-time searching algorithms provided within the octree class allowed us to see how powerful PCL could be. Since, we were not completely satisfied with the correspondence estimation methods seen so far, we decided to search for such an algorithm in PCL. This algorithm exists and it is available as part of the registration module. Registration is a particular task involved in point cloud processing. More specifically, the registration task is defined as the problem of consistently aligning various 3D point cloud data views into a complete model. Registration aims to find the relative positions and orientation of the separately acquired views in a global coordinate framework, such that the intersecting areas between them overlap perfectly. As part of the registration process, the correspondence estimation is used to find correct point correspondences in the input datasets, and to estimate rigid transformations that can rotate and translate each individual dataset into a consistent global framework. Our purposes overlap with those of the registration module until the correspondence estimation is accomplished. After the correspondence estimation step, our task differs significantly from the registration task, since the human body is not subject to rigid transformations only. However, we have exploited the correspondence estimation algorithm founded in PCL discovering that it is well-suited for our application. In particular we dealt with the *CorrespondenceEstimation* class that provides two different methods to estimate pairs of matching points:

- *determineCorrespondences*;
- *determineReciprocalCorrespondences.*

The difference between these two methods stands in the number of correspondences returned, and so in the way they compute the correspondences between points. The method *determineCorrespondences* returns a number of correspondences equal to the number of points in the previous cloud or the number of points in current cloud, depending on how parameters are set. In fact, *determineCorrespondences* performs a one-way matching between the clouds, like that performed in our algorithm based on octrees. Matching point clouds in one-way means that:

- one cloud is fixed: this is the cloud in which searches are performed;
- points in the other cloud are searched in the fixed cloud.

It is easy to see how the number of correspondences returned can vary, depending on which cloud is set as fixed. The *determineCorrespondences* method is really similar to our octree-based method with the exception of the data structure used: in *determineCorrespondences*, kdtrees with FLANN searches are used as underlying data structure. The second and final method provided by PCL is *determineReciprocalCorrespondences*. In *determineReciprocalCorrespondences*, a two-way matching algorithm is applied in order to estimate correspondences. This means that, if we are referring to two clouds, namely A and B:

1. the one-way algorithm is applied twice: once with the cloud A as fixed cloud, and once with the cloud B as fixed cloud;
2. from point 1 two different vectors of correspondences are returned, those vectors are then intersected and a new vector of correspondences is returned.

Since *determineReciprocalCorrespondences* provides a more precise correspondence estimation, we chose to use it in this work. In Figure 5.5, we show the correspondences estimated while a person is *hand waving*. The left cloud is the cloud in the previous frame, while the right cloud is the cloud at the current frame and both clouds are referred to the same actor. Screenshots are obtained by means of the PCL visualizer (see Section 5.3.3).

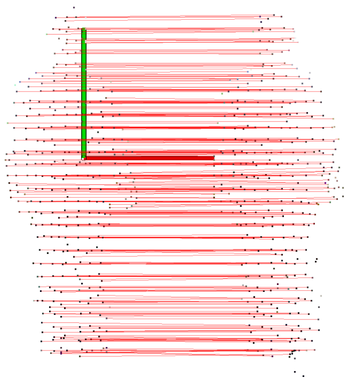
5.3.2 The optical flow calculator

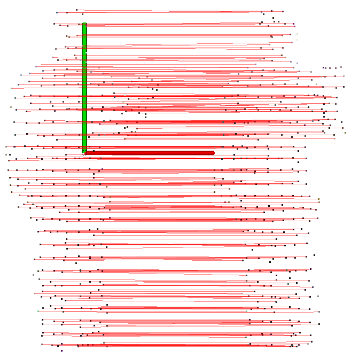
In this work, the *optical flow calculator* represents the tool that enables us to compute the 3D optical flow in each frame. The instantiation of the *optical flow calculator* requires five input elements:

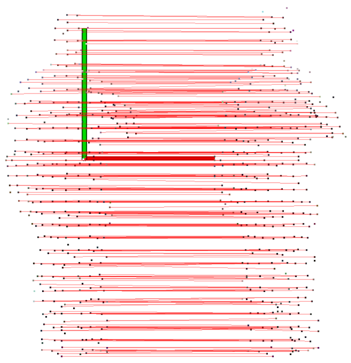
- a vector containing the estimated correspondences between points;
- the previous cloud to which we are referring at the current frame;
- the current cloud associated with the current frame;
- the previous vector of indices related to points in the previous cloud;
- the current vector of indices related to points in the current cloud.

The *optical flow calculator* has several methods that respectively return as output what follows:

1. A vector of 3D velocity vectors: this vector is aligned with the vector storing the estimated correspondences.
2. A vector of norms: each norm is related to the corresponding 3D velocity vector and the overall vector of norms is aligned with the vector storing the estimated correspondences.


(a) First current frame to the right.


(b) Second current frame to the right.


(c) Third current frame to the right.

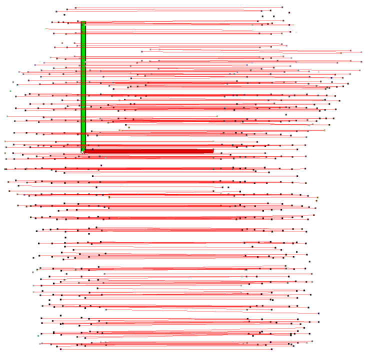

(d) Fourth current frame to the right.

Figure 5.5: Display of the estimated correspondences with regard to the action *hand waving*. Here the method *determineReciprocalCorrespondences* of PCL is exploited.

3. A vector of records: this vector is aligned with the vector storing the estimated correspondences. Furthermore, each record stores four different fields: the 3D velocity vector, the temporal difference used in computing the 3D velocity vector, the norm of the velocity vector and a final point.

4. An annotated point cloud: the current point cloud is customized by adding more information with respect to the default ones.

To achieve the results outlined in points 1, 2 and 3, the input vector of estimated correspondences is scanned and at each iteration a pair of matching indices is retrieved. From each pair of indices, the related points are caught, respectively from the current and the previous cloud, in constant time. Then, the spatial difference and the temporal difference between points are computed, and finally the 3D velocity vector and its norm are obtained. In particular, point 3 requires also the computation of a "final point". Given a 3D velocity vector and a current point to which this vector is applied, we define *final point* the point in which the current point will be, if the velocity vector computed is actually applied to it. We perform the calculus of this special point, in order to simplify the visualization of the optical flow on the RGB image. As concerns point 4, the *optical flow calculator* is enabled to return an annotated point cloud as output, in order to speed up the successive computations. In this work, an annotated point cloud is a point cloud in which a customized point type is defined. Indeed, in addition to the implemented point types, PCL allows the user to define custom points, in which the traditional fields can be accompanied by personalized fields. The point type associated with our annotated clouds contains:

a. Three fields related to geometric coordinates of the point: x, y, z.

b. Three fields related to the color information associated to the point: r, g, b.

c. Three fields related to the velocity vector associated to the point in the previous frame: $prev\_v_x$, $prev\_v_y$, $prev\_v_z$.

d. Three fields related to the velocity vector associated to the point in the current frame: v_x, v_y, v_z;

e. One field related to the temporal difference to which the optical flow computation is referred: $timeDifference$;

f. One field related to the norm of the velocity vector associated to the point int the current frame: $norm$;

g. One field representing a flag: isDefined. If the flag is set to zero, then there is no valid velocity vector associated with that point (namely, the point has not a correspondent one in the previous cloud), by contrast, if the flag is set to 1 a valid velocity vector is associated to the point.

Originally, our custom point type was composed only by the fields outlined in a. , b. , f. . Successively it has been extended to contain also c. , d. , e. and g., in order to easily compute features described in Section 6.1.2. Therefore, we refer the reader to Section 6.1.2 for further details about the purpose of those additional fields, while we discuss here about the initial purpose of the annotated cloud. This purpose is related to the development of our customized correspondence rejection method.

Before developing our own correspondence rejection method, we sought to find such method in PCL. Specifically, we investigated the following methods in the registration module:

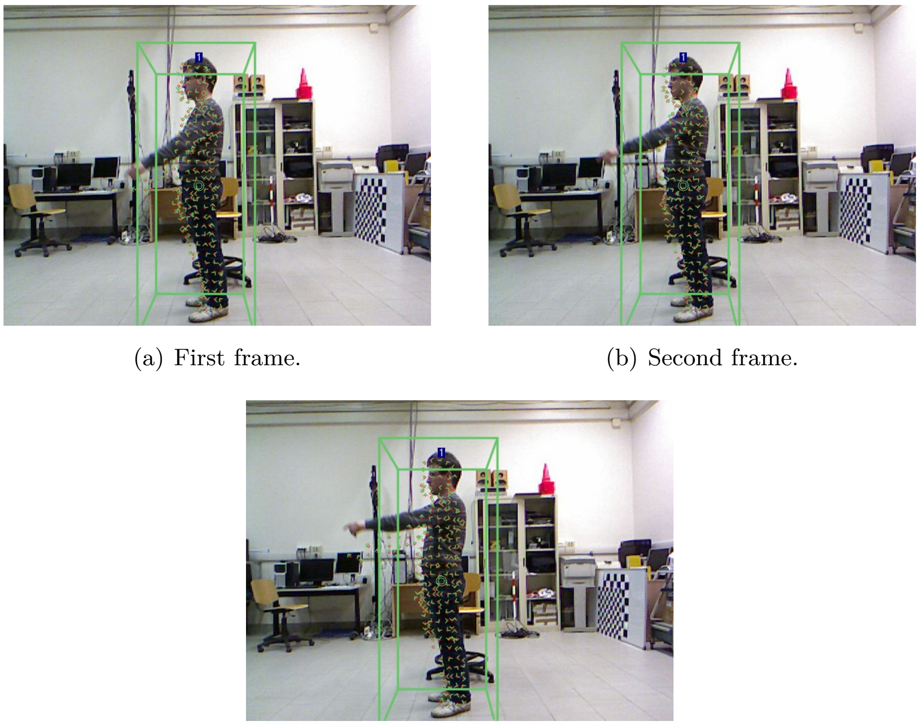
- *CorrespondenceRejectorDistance*, which implements a simple correspondence rejection method based on thresholding the distances between the correspondences.
- *CorrespondenceRejectorOneToOne*, which implements a correspondence rejection method based on eliminating duplicate match indices in the correspondences.
- *CorrespondenceRejectorFeatures*, which implements a correspondence rejection method based on a set of FPFH[2] descriptors.

We found that, *CorrespondenceRejectorDistance* and *CorrespondenceRejectorOneToOne* are somewhat useless when combined with the correspondence estimation method owned by PCL, because they do not substantially improve the previous estimation. The same methods could potentially be useful when our custom correspondence estimation method is used: especially *CorrespondenceRejectorOneToOne* could be effective since our algorithm does not discard duplicate match indices. Besides, the main issue when dealing with *CorrespondenceRejectorDistance* regards the difficulty in estimating the distance to be set as threshold. Finally, we decided not to use any of these methods. The method which seemed more attractive to us was *CorrespondenceRejectorFeatures*. *CorrespondenceRejectorFeatures* aims to reject "bad" correspondences based on estimating FPFH descriptors. Unfortunately, this algorithm is based on an input features space, i.e. it takes as input correspondences related to FPFH point clouds, and so it is not suited for our purposes. Furthermore, also the remaining correspondence rejection methods in PCL, namely *CorrespondenceRejectionTrimmed* and *CorrespondenceRejectorRandomSampleConsensus*, are not well suited for our application, since they have been designed expressively to the purpose of registering point clouds.

It is clear to the reader that PCL did not meet our needs for what concerns the correspondence rejection issue. Therefore, we decided to develop our own correspondence rejector in order to target a specific objective: removing correspondences associated to noisy data. Indeed, when dealing with Microsoft Kinect, the mapping and remapping of points from and to the Kinect reference system, the camera reference system and the real world reference system, are subject to errors. Those errors affect the geometric coordinates of the points, namely if a point is not actually moving between a frame and the successive one, the system detects different geometric coordinates for the same point in the next frame. Obviously, if coordinates between frames are not constant, the 3D velocity vectors will not result equal to the zero vector, thus identifying points as moving points even if

[2]FPFH, Fast Point Feature Histogram.

they are not actually moving. To delete these noisy points together with their correspondences, the optical flow is first computed and returned as an annotated point cloud. Since the annotated point cloud stores the norm of the current velocity vector in each point, we performed a thresholding based on the norm of the 3D velocity vector. More specifically, our algorithm combines a simple thresholding to a complex thresholding: the former discards correspondences associated to points for which the norm is lower than a determined value, while the latter applies to points which pass the previous step. With regards to those points, a radius search is performed initially to find closest points to the query ones, then the norm of the founded neighbors is taken into account to perform the final thresholding. Figure 5.6 shows three frames in which the optical flow is displayed in the RGB image. The optical flow is obtained without applying the correspondence rejection method described so far. Furthermore, Figure 5.7 shows the same frames, but this time the flow is referred to a previous computation of the rejection algorithm outlined.

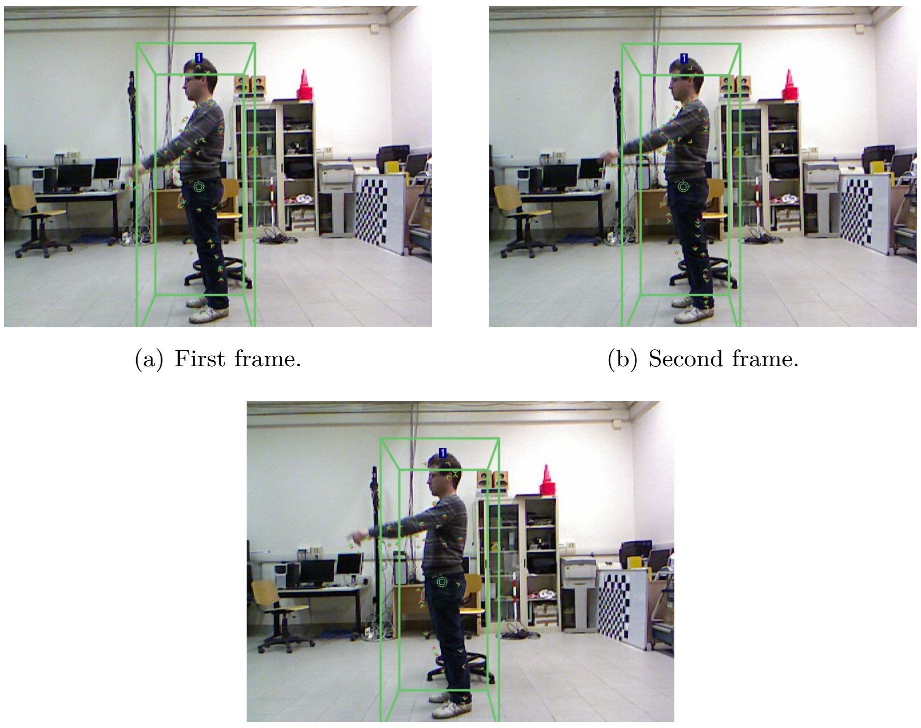

(a) First frame.

(b) Second frame.

(c) Third frame.

Figure 5.6: Display of the estimated 3D optical flow with regard to the action *pointing*. Here our correspondence rejection method is not applied.

Eventually, there is a substantial trade-off between the noise removal and the rejection of correspondences. Indeed, if an algorithm is developed to strictly re-


(a) First frame.

(b) Second frame.

(c) Third frame.

Figure 5.7: Display of the estimated 3D optical flow with regard to the action *pointing*. Here our correspondence rejection method is applied.

move the noise derived from the data acquisition and from the subsequent mapping, it could happen that the rejection discards possible useful matches, and on the contrary, if an algorithm is build to explicitly reject "bad" correspondences then the noise might not be removed effectively. We decided to perform a noise removal not so strict to prejudice the number of final correspondences returned by the algorithm.

5.3.3 Visualizing results

In order to visualize results coming from point clouds processing, we used the *PCLVisualizer*, a tool available in PCL from version 1.2. The *PCLVisualizer* is a full-featured visualization class, indeed it currently offers features such as displaying normals, drawing shapes and multiple viewports. In this work, the *PCLVisualizer* has been customized and used in order to display:

- point clouds with their original colors;

- point clouds with modified colors in relation to the norm of the estimated 3D optical flow;
- normals associated to point clouds;
- the estimated correspondences between the point cloud at the current and the previous frame (see Section 5.3.1);
- the norm and direction of the 3D optical flow computed;
- the grid developed to compute the descriptor associated to the 3D optical flow computed (see Section 6.1.1).

We have implemented the *PCLVisualizer* as an independent ROS node, so if specific data need to be visualize by our personalized graphical tool, they have to be sent through ROS as a message. In this work, point clouds or elements needed by the *PCLVisualizer* are sent to the tracker. Then, the tracker packs all those data in a single message, called *PCLVisualization*.msg, and it finally sends the *PCLVisualization* message over ROS, at each frame. The *PCLVisualizer* node in turn catches the upcoming messages and it performs the computations required. As a side note, we want to point out that there is a trade-off between the overall amount of data sent under a specific topic and time performances: if too much data are sent, the entire application could experience slowdowns. So when dealing with point clouds characterized by having a discrete number of points, care has to be taken while adding them to the *PCLVisualization* message.

5.4 Comparison with 2D optical flow

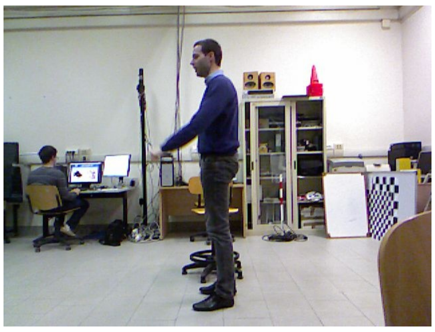
The approach used in this work to compute 3D optical flow is really new in literature. To demonstrate the validity of our approach, we propose a comparison with a well-known 2D computation of the optical flow: specifically, we refer to the approach followed by Dalal in the context of [8]. The comparison has been executed as follows:

- First, we chose two recordings representing two different actions: one in which only arms are moving, and one in which the actor's entire body is moving.
- Second, we collected some frames representing each actions: those frames are successively fed to the 2D optical flow calculator.
- Third, we computed separately the 3D optical flow with our application and the 2D optical flow for the selected frames with the Dalal approach.

- Finally, we visually compared the results.

As additional details our application is implemented in the C++ programming language, while as for Dalal's algorithm we used a Matlab implementation. Furthermore, as test bed, we chose the actions *pointing* and *walking*. Results regarding the action *pointing* are shown in Figure 5.8, 5.9, 5.10, 5.11 and 5.12. While, results regarding the action *walking* are show in Figure 5.13, 5.14, 5.15, 5.16 and 5.17.

It is easy to see how the Dalal approach allows to fully detect the motion, while in our approach some motion could be only partially described by motion vectors (see Figure 5.11 and Figure 5.12). However in order to achieve this precision, tradional 2D methods need the entire images to perform the optical flow computation, thus resulting in poor time performances. Event though our method could not achieve the precision of traditional 2D approaches, it is able to detect correctly the direction and the magnitude of the majority of the motion vectors. The size of the cloud associated to each track depends in turn on the voxel grid filter applied in the detection phase of our application (see [2, 13]). By default, the voxel grid filter is set to be restrictive: this allows the algorithm to achieve good rates, namely about 23 frames per second. If the filtering power of the voxel grid filter is lowered, then clouds will have more points and thus the 3D optical flow will be composed by more motion vectors with respect to the restrictive case. However, if the filtering power is lowered too much, the computational performances will necessarily degrade. Figures 5.11, 5.12, 5.16 and 5.17 are referred to the application of a restrictive voxel grid filter (default choice). While, in Figure 5.18 and in Figure 5.19 we propose analogous screenshots, but this time they are referred to a less restrictive voxel grid filtering.


(a) First frame.

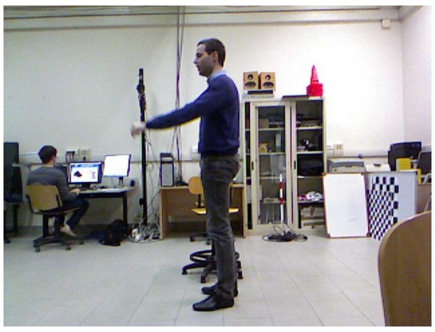

(b) Second frame: this frame is the frame just following 5.8(a).

Figure 5.8: Comparing 2D versus 3D optical flow. The chosen frames related to the action *pointing* are shown.

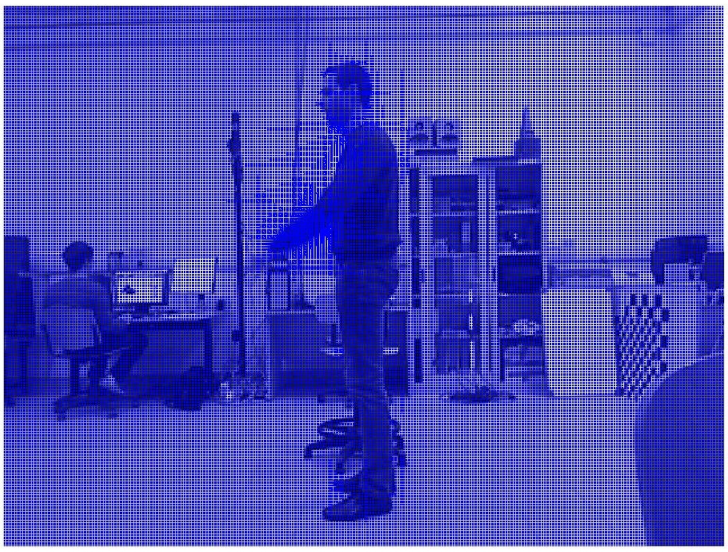


Figure 5.9: Comparing 2D versus 3D optical flow. The 2D optical flow computed for the *pointing* action with the Dalal method is shown.

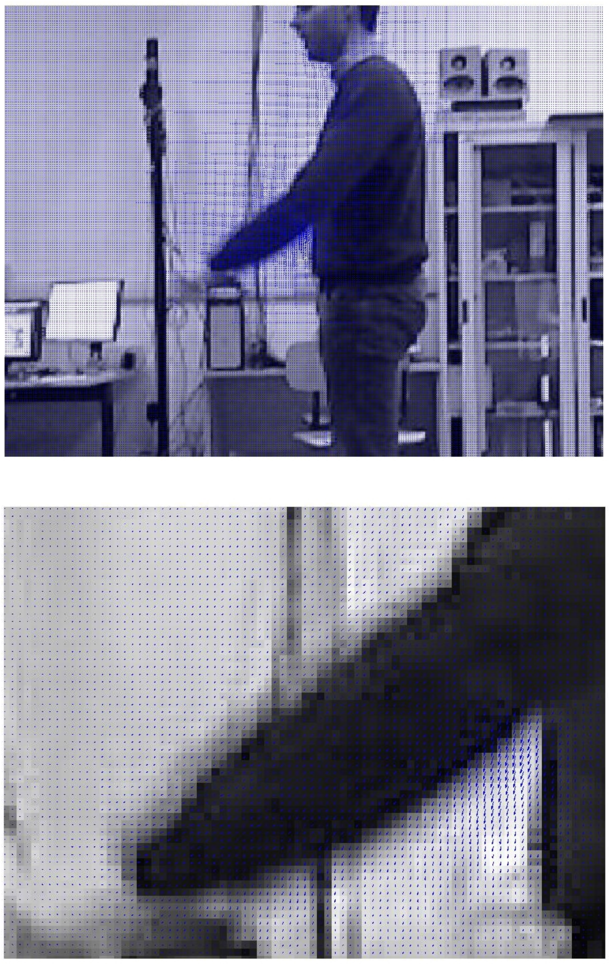
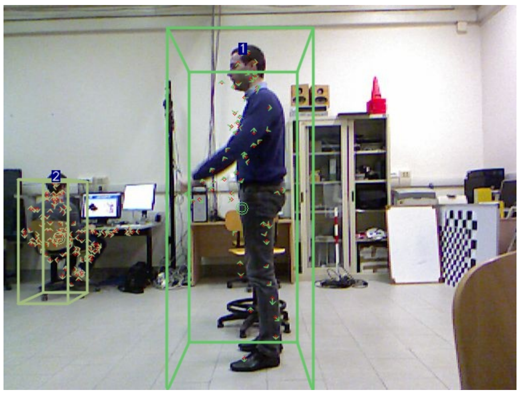
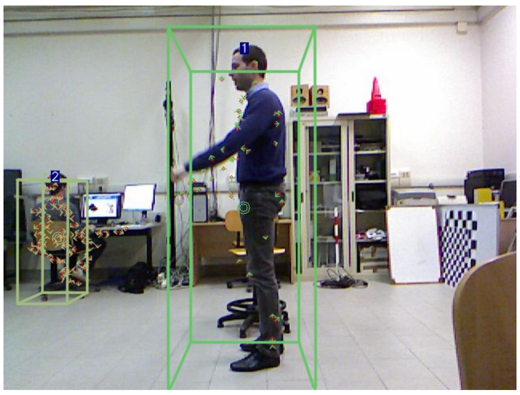


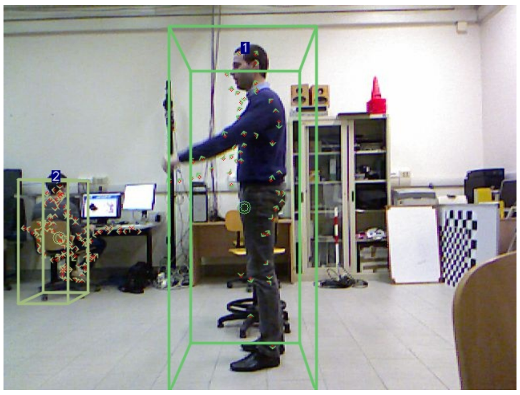
Figure 5.10: Comparing 2D versus 3D optical flow: two different zoom of Figure 5.9.



(a) First frame.

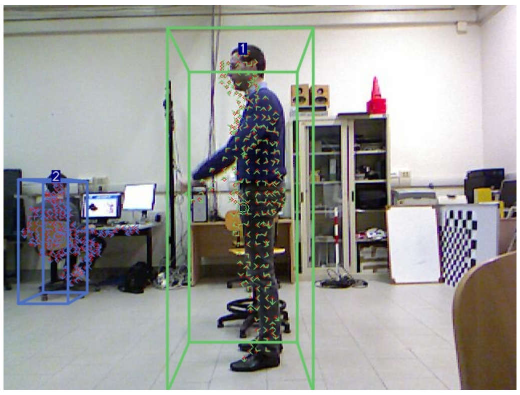


(b) Second frame.

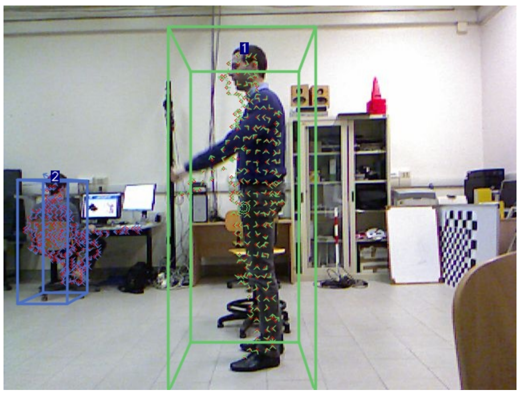


(c) Third frame.

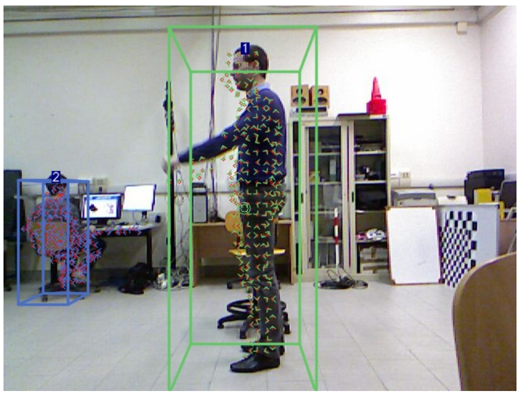
Figure 5.11: Comparing 2D versus 3D optical flow. The 3D optical flow captured from our application in three successive frames is shown. The frame in 5.11(a) is the same frame shown in 5.8(a). Furthermore, screenshots are referred to a computation of the 3D optical flow in which the correspondence rejection method described in Section 5.3.2 is previously applied.



(a) First frame.

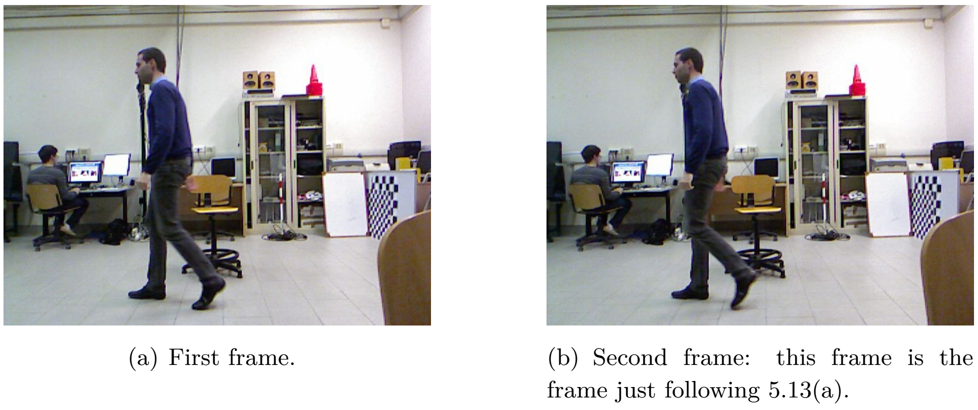


(b) Second frame.



(c) Third frame.

Figure 5.12: Comparing 2D versus 3D optical flow. The 3D optical flow captured from our application in three successive frames is shown. The frame in 5.12 is the same frame shown in 5.8(a). Furthermore, screenshots are referred to a computation of the 3D optical flow in which the correspondence rejection method described in Section 5.3.2 is not previously applied.



(a) First frame.

(b) Second frame: this frame is the frame just following 5.13(a).

Figure 5.13: Comparing 2D versus 3D optical flow. The chosen frames related to the action *walking* are shown.

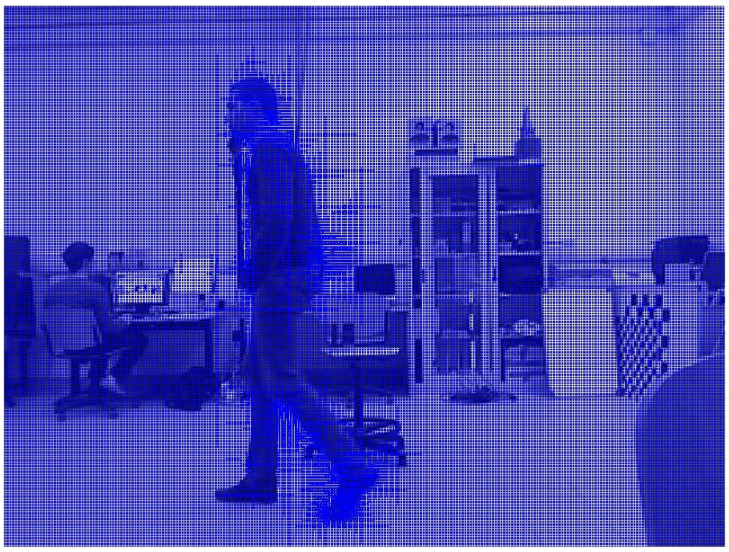
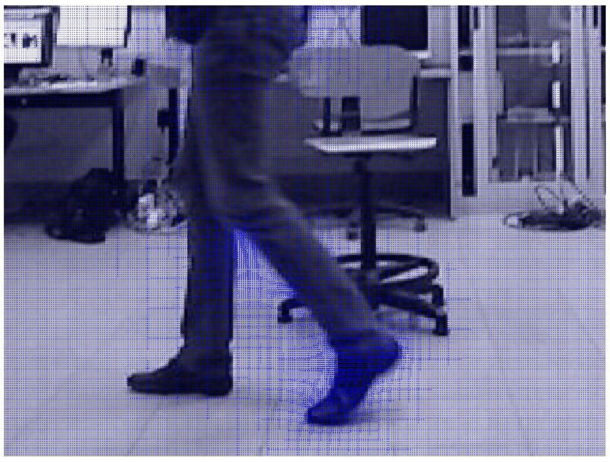


Figure 5.14: Comparing 2D versus 3D optical flow. The 2D optical flow computed for the *walking* action with the Dalal method is shown.



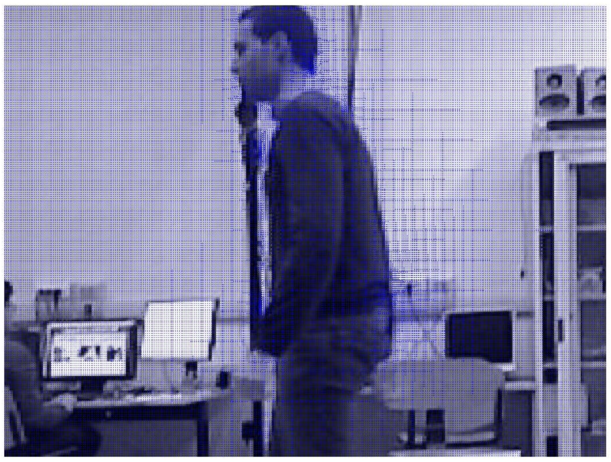
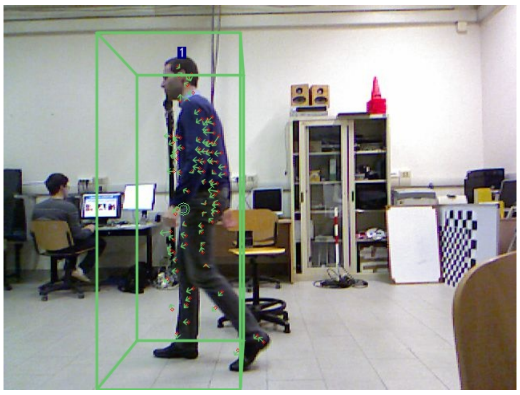
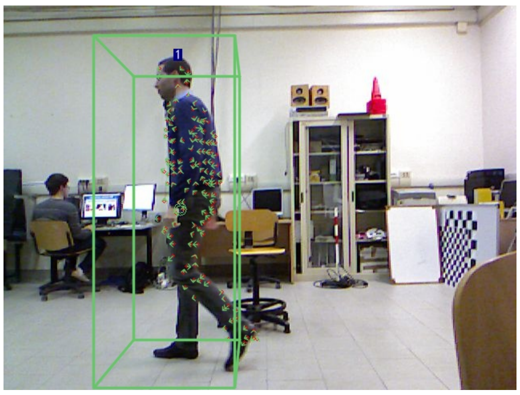


Figure 5.15: Comparing 2D versus 3D optical flow. Two different zoom of Figure 5.14.


(a) First frame.


(b) Second frame.

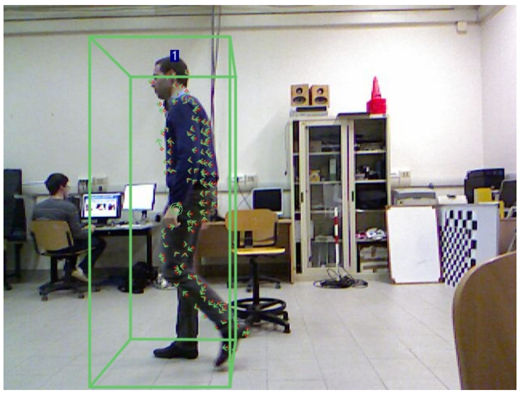
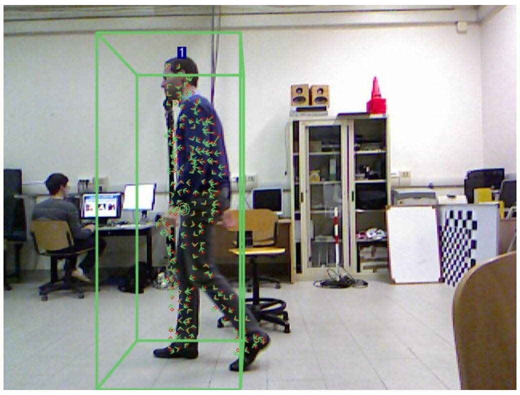
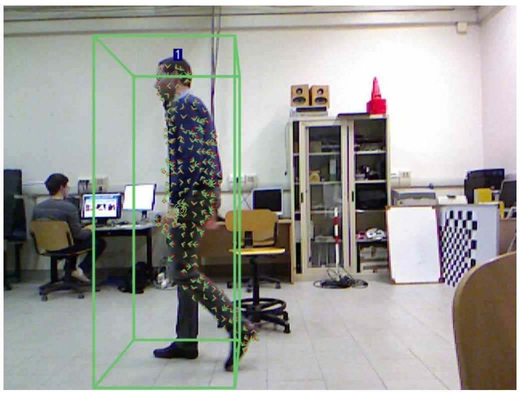

(c) Third frame.

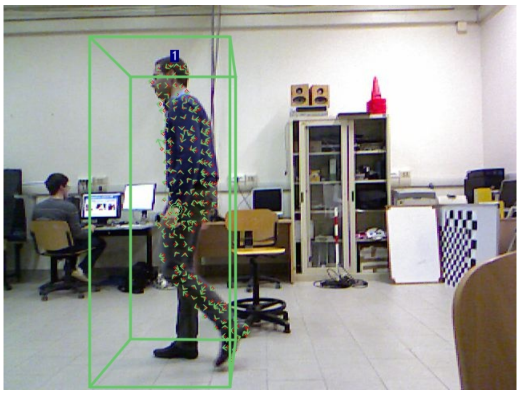
Figure 5.16: Comparing 2D versus 3D optical flow. The 3D optical flow captured from our application in three successive frames is shown. The frame in 5.16(a) is the same frame shown in 5.13(a). Furthermore, screenshots are referred to a computation of the 3D optical flow in which the correspondence rejection method described in Section 5.3.2 is previously applied.



(a) First frame.

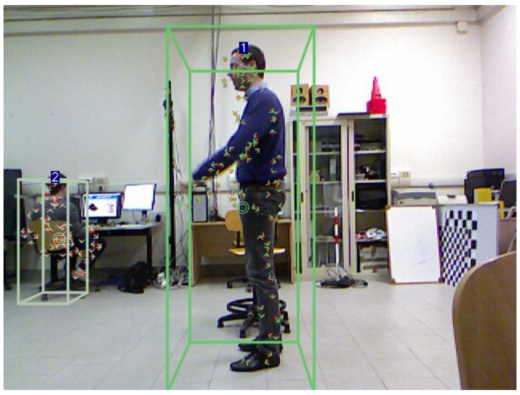


(b) Second frame.

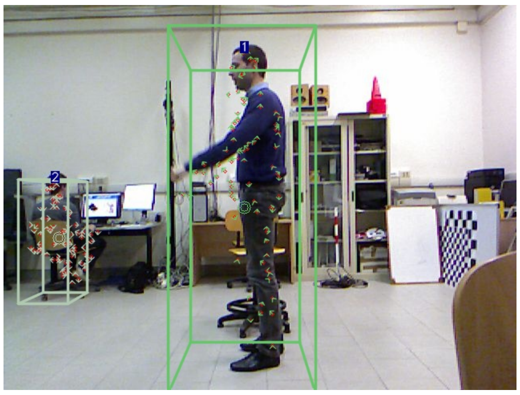


(c) Third frame.

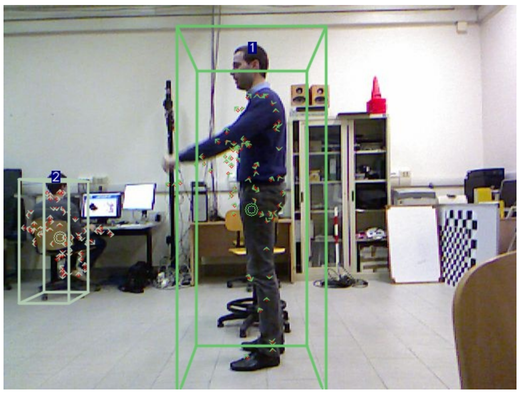
Figure 5.17: Comparing 2D versus 3D optical flow. The 3D optical flow captured from our application in three successive frames is shown. The frame in 5.17 is the same frame shown in 5.13(a). Furthermore, screenshots are referred to a computation of the 3D optical flow in which the correspondence rejection method described in Section 5.3.2 is not previously applied.



(a) First frame.

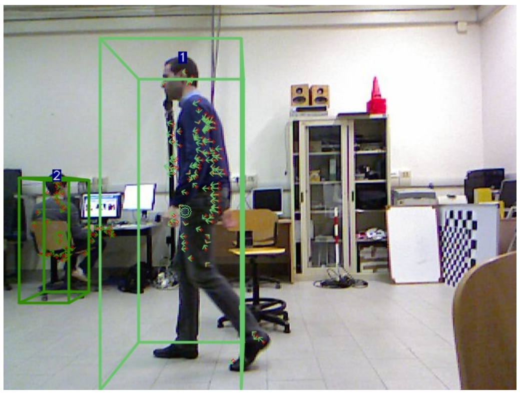


(b) Second frame.

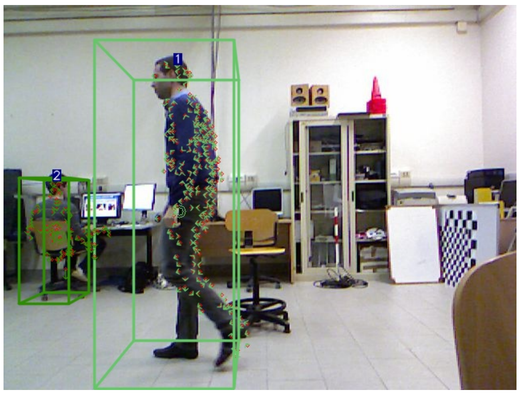


(c) Third frame.

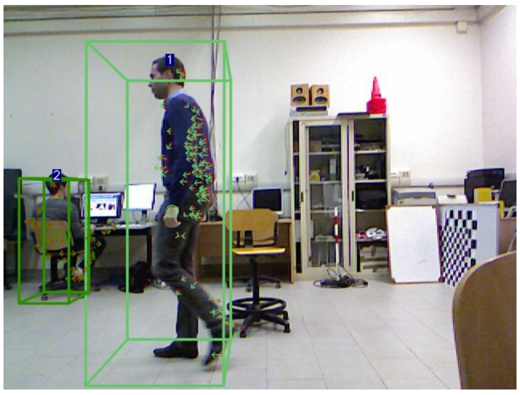
Figure 5.18: Comparing 2D versus 3D optical flow. The 3D optical flow captured from our application in three successive frames is shown. Like in Figure 5.11, the frame in 5.18(a) is the same frame shown in 5.8(a). Furthermore, screenshots are referred to a computation of the 3D optical flow in which the correspondence rejection method described in Section 5.3.2 is previously applied and the voxel grid filtering is set to be less restrictive with respect to Figure 5.11.



(a) First frame.



(b) Second frame.



(c) Third frame.

Figure 5.19: Comparing 2D versus 3D optical flow. The 3D optical flow captured from our application in three successive frames is shown. Like in Figure 5.16, the frame in 5.19(a) is the same frame shown in 5.13(a). Furthermore, screenshots are referred to a computation of the 3D optical flow in which the correspondence rejection method described in Section 5.3.2 is previously applied and the voxel grid filtering is set to be less restrictive with respect to Figure 5.16.

Chapter 6

Recognizing actions from 3D optical flow information

This chapter is focused on the description and classification phases of the entire recognition process. The first part of the chapter describes how we have computed a suitable descriptor from the 3D optical flow calculated in Chapter 5, while the second part of the chapter is focused on typical aspects of the classification of actions.

Having calculated the 3D optical flow associated to the point cloud of an individual, a suitable description for the flow was required. In fact, the 3D optical flow computed in Chapter 5 is generally composed by a different number of velocity vectors in each frame, even if those vectors are associated to same person. Therefore, we faced with the challenge of finding a descriptor:

- able to efficiently summarize the flow;
- characterized by a constant size in each frame.

To this aim, we computed a 3D grid surrounding each person in the scene and, successively, we captured some relevant piece of information from each voxel of the grid. The *voxel grid* computation is described in Section 6.1.1, while the features we extracted from each grid cube are discussed in Section 6.1.2.

The second part of the chapter is focused on the classification of the descriptors computed for the 3D optical flow. Since the classification is based on multiple frames, first a series of descriptors is collected. The entire collection is then treated as a unique vector to be normalized. Normalization enables to partially discard the presence of noise in the raw data. Finally, a nearest neighbor classification is performed between collections of training and test examples. To see the real effectiveness of our work, we created a new dataset that we used as test bed while executing the application. The creation of a new dataset is motivated by the actual lack of publicly available datasets in which RGB-D data are provided (see Section 2.3.2). Indeed, our work is targeted to video surveillance applications and typical actions accomplished in the field of personal and assistant robotics did not meet our intents. In particular, Section 6.2 is focused on classifying descriptors, while Section 6.3 briefly discusses the experimental results related to the recognition based on the 3D optical flow calculus.

6.1 Computing descriptors

Training classifiers generally requires structured data as input. In the most of the cases data are organized in matrices or files in which each row represents a labeled descriptor. Each descriptor is simply a data vector and all training vectors must have the same size. Furthermore, the labeling refers to the action class which the descriptor belongs to. Since our aim is to perform a classification based on descriptors, we first had to compute a descriptor with fixed-size from the 3D optical flow. Section 6.1.1 describes how we could achieve a fixed-size for the descriptors, while Section 6.1.2 discusses the elements each descriptor is composed of.

6.1.1 Voxel grid

In order to achieve a descriptor of constant size, we developed a 3-dimensional grid surrounding the point cloud associated to each track. The 3D grid provides an effective spatial partition of the points in a cloud. Furthermore, since each point of the current cloud is associated with a 3D velocity vector (see Section 5.3.2) by means of an annotated point cloud, the *voxel grid* provides also a 3D optical flow partition.

With regards to implementation details, it is important to observe that PCL does not contain any tool able to accomplish a spatial partition usable for our purposes. Therefore we developed a structure, named *voxel grid*, that enables us to efficiently summarize the 3D optical flow. The instantiation of a *voxel grid* takes only two elements as input:

- the current point cloud;

- the current vector of indices related to points in the current cloud;

but it requires that:

- the spatial divisions along the x, y, z axis, and
- the annotated point cloud associated to the current cloud

are set before actually calculating the descriptor. To compute the grid, first minimum and maximun bounds along the three dimensions are defined. Bounds are established so that the current cloud is centered in the grid. Secondly, the bounds of the grid are combined with the spatial divisions in order to define the minimum and maximum ranges, along the three dimensions, associated to each voxel of the grid. Once minimum and maximum ranges are defined for each 3D cube, the current cloud is scanned and each point of the cloud is put into the right cube based on its 3D geometric coordinates. More specifically, for each voxel of the grid only indices related to points are stored. In any case, indices enable to access the points of a cloud in constant time. Furthermore, the current cloud indices are perfectly aligned with the indices of its associated annotated cloud. Finally, the *voxel grid* is returned as a dynamic matrix in which:

- each row represents a 3D voxel of the grid;
- each row contains a vector of indices: indices are related to points in the cloud which stands in the voxel at issue.

After the calculus of the optical flow has completed, the *voxel grid* is computed for the current cloud associated with each track. Figure 6.1, 6.2, and 6.3 shows three different views of the grid computed. In Figure 6.1and 6.2, the displayed grid is characterized by having, respectively, 2 and 4 divisions along the x, y, and z axis. In Figure 6.3 instead, the grid is featured by 5 divisions along the x axis, 3 divisions along the y axis, and finally, 2 divisions along the z axis. The points of the visualized point cloud are expressed in the Kinect reference system, in which the y axis represents the height of the cloud and the z axis is related to the depth. Finally, from Figure 6.3, it should be noticed that if we want to distinguish the left side from the right side of the human body, an odd number of partitions should be set for each dimension.

6.1.2 Extracting features from 3D voxels

Once the 3D grid is computed we defined two different descriptor types, in order to see if at least one of them has a real effectiveness. The two descriptors differ according to the features extracted from each voxel of the grid. The first kind of descriptor summarizes the defined 3D velocity vectors contained in each 3D voxel by means of the 3D average velocity vector. Therefore, each 3D

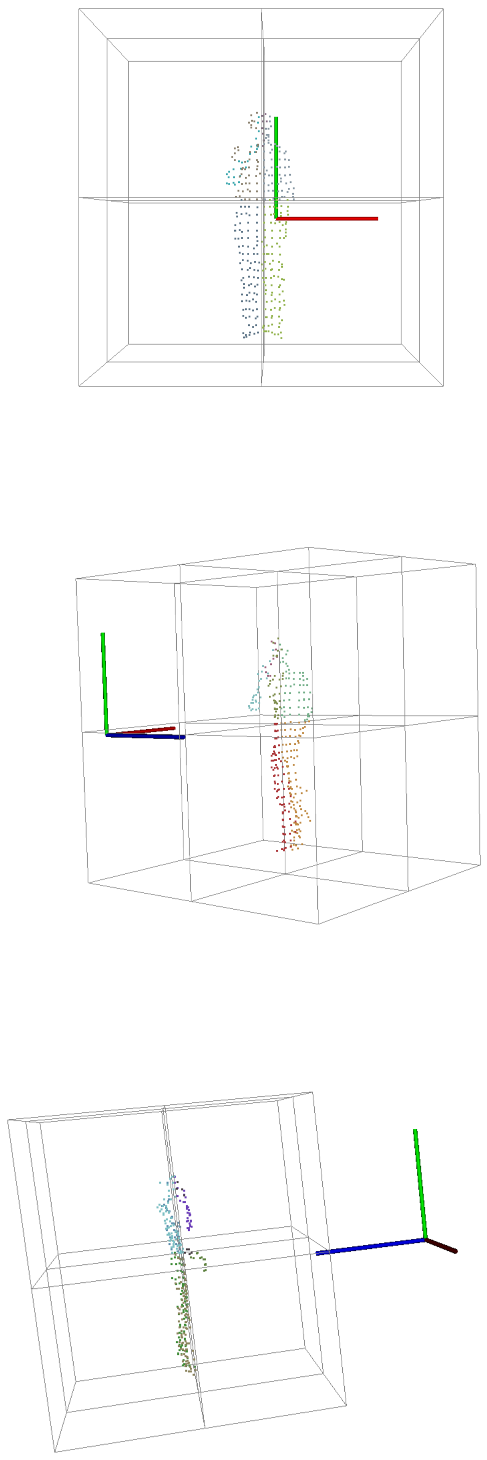
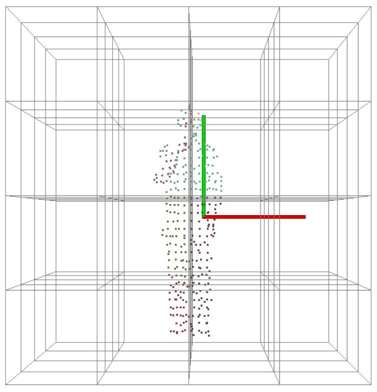
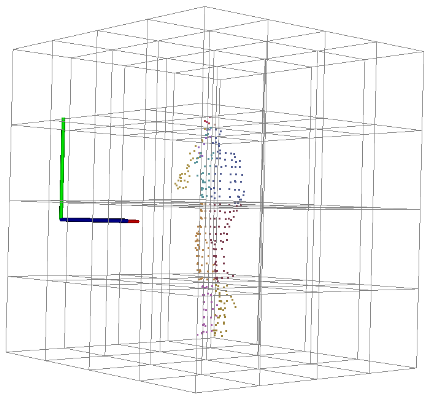


Figure 6.1: Example views of the voxel grid computed: 2 partions along the x, y and z axis are designed.





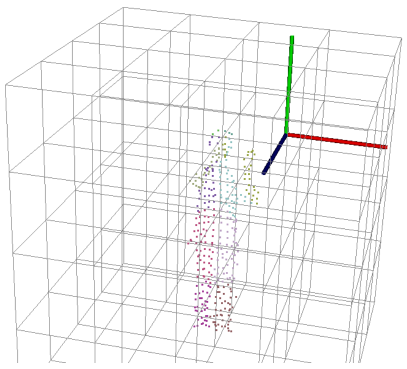


Figure 6.2: Example views of the voxel grid computed: 4 partions along the x, y and z axis are designed.

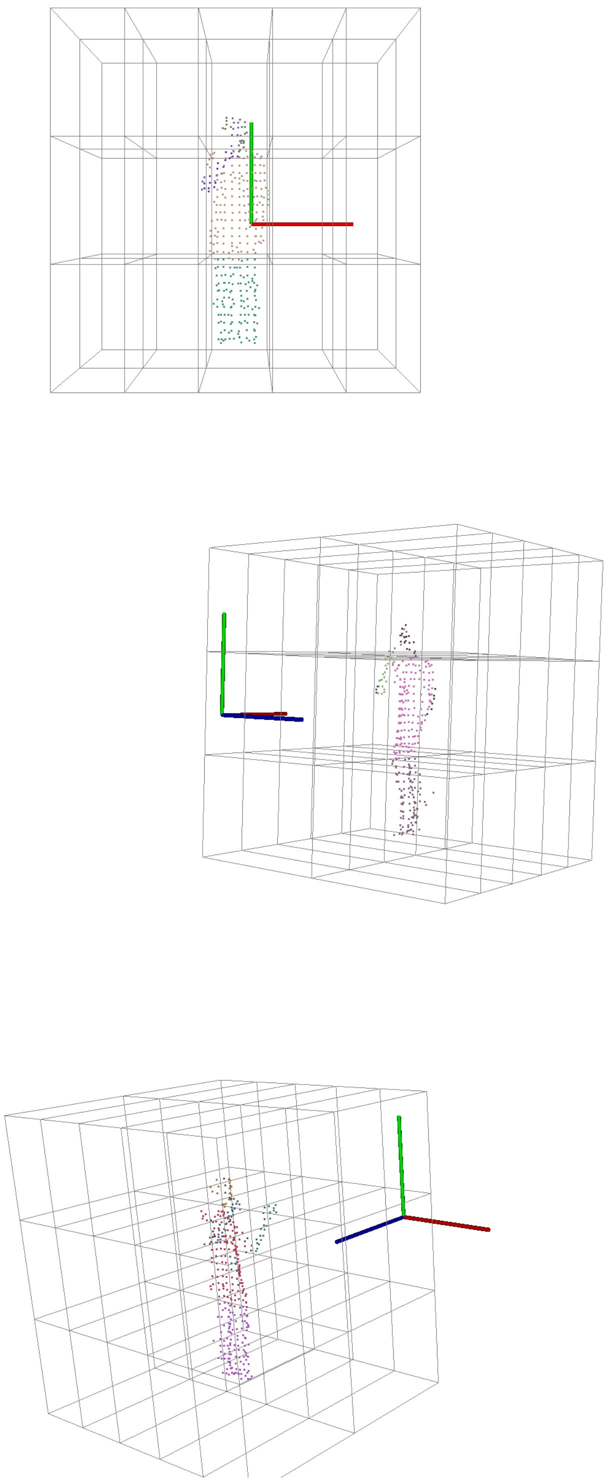


Figure 6.3: Example views of the voxel grid computed: 5 partions along the x, 3 partions along the y axis and 2 partitions along the z axis are designed.

voxel actually contains three elements: the x, y, and z components of the average velocity vector calculated for the voxel at issue. The total size of the descriptor is equal to 3 $\times$ $size\_of\_the\_grid$, where $size\_of\_the\_grid$ is in turn equal to $x\_partitions$ $\times$ $y\_partitions$ $\times$ $z\_partitions$ in which $x\_partitions$, $y\_partitions$ and $z\_partitions$ are the grid partitions along the x, y, z axis.

The second kind of descriptor provide additional information with respect to the first one. In particular, the 3D velocity vectors in each voxel are summarized by the following data:

- the 3D average velocity vector (3 elements, i.e. the three components of the vector);
- the minimum, maximum and average divergence (3 elements);
- the minimum, maximum and average curl along the x, y, and z axis (9 elements).

We calculated the divergence and the curl for each of the 3D voxel according to the formula 6.1 and 6.2.

$$divergence = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \tag{6.1}$$

$$curl = \left(\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z}, \quad \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x}, \quad \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right) \tag{6.2}$$

The total size of the descriptor is equal to 15 $\times$ $size\_of\_the\_grid$ this time, where $size\_of\_the\_grid$ is defined as outlined above. As a remarkable observation, we chose not to apply the change detection algorithm and the correspondence rejection method discussed respectively in Section 5.2 and in Section 5.3.2. Indeed, since the system was not previously tested with a classification algorithm, we did not want to lose possible information coming from the 3D optical flow. So, we accepted to work with a noisy flow and we accomplished a classification based on it. Depending on the results obtained, the change detection and the correspondence rejection algorithms can be applied to see if better results could be achieved subsequently to an improvement of the flow quality.

6.2 Classifying descriptors

The features extraction step outlined in Section 6.1.2 returns as output a vector. There is a bijection between the vector indices and the 3D voxel of the grid. Indeed, the vector stores at each index the extracted features for the corresponding voxel. This vector is the descriptor that summarizes the 3D optical flow information captured in a single frame. Since after the descriptor computation the human action recognition task becomes a classification problem, we defined a precise approach to classify actions. It should be noted that, even if the approach to recognition has to be well-defined, the one implemented in this work will be subject to further improvements. Indeed, we first developed a "simple" classification system with the aim of modifying it with regards to the results obtained. In this work, the classification process is divided into two main steps. The first step concerns the creation of a new dataset and it is addressed in Section 6.2.1, while the second step is related to the actual classification (see Section 6.2.2).

6.2.1 Collecting video samples

In this section, we discuss the video dataset collected in the context of this work. The database we propose is oriented to video surveillance tasks. Each video sample is captured in the form of a ROS bag, that is a particular file in which ROS topics are recorded. ROS bags enable to record data from a running ROS system and then to play back the data to produce a similar behaviour in the running system. The data recorded are stored in a .bag file which represents the file format of each video sample in our dataset. The recorded topics are:

- `/camera/rgb/points`, which represents the combination of the color camera output stream and the depth sensor output stream;
- `/camera/rgb/camera_info` and `/camera/rgb/image_color/compressed`, which provides respectively the color camera intrinsic parameters and the color camera compressed stream of images;
- `/tf`, which allows to send and receive multiple coordinate frames over time;
- `/skeleton_output`, a particular topic that keeps track of the estimated skeleton joints of the human body.

The reader could observe that the information related to the color camera are duplicated in our recordings. That is true, but we aim to avoid this repetition in further developments of this work.
In particular, when only `/camera/depth/points` will be recorded together with `/camera/rgb/camera_info` and `/camera/rgb/image_color/compressed`, an explicit alignment between the depth information and the RGB images must be embedded in our application. Finally, the particular topic `/skeleton_output` is due to our purpose of comparing the recognition performances of the application based on the 3D optical flow information with respect to a system based on acquiring OpenNI[1]

[1]See the OpenNI webpage.

skeleton joints data.

The video dataset is entirely collected in a lab environment. There are minor variations in the camera position and orientation due to repeated mountings of the camera. The framing was arranged so that the actor is centered in the scene with clearly visible feet. The current video dataset is inspired by the INRIA Xmas Motion Acquisition Sequences (IXMAS) dataset (see Section 2.3.2) and it currently contains six types of human actions:

1. *standing*
2. *hand waving* (with right hand)
3. *sitting down*
4. *getting up*
5. *pointing* (with right arm)
6. *walking*

performed once by six different actors. The six volunteers were invited to naturally execute the actions, and no indication about how to accomplish movements was given to them. Each actor was asked to perform all the six actions in the sequence outlined above, and we collected a single ROS bag file for each actor. Subsequently, each ROS bag was manually segmented to represent only a single action. This mean that we chose the instant in which an action begins and ends according to the common sense. Each of the segmented video samples spans from about 1 second to 7 seconds. Three example frames from each action class are illustrated in Figure 6.4, 6.5, 6.6, 6.7, 6.8 and 6.9.

Finally, we want to make two remarkable observations. First, when the human body acts in the environment, different depth layers come into play. Generally, this implies that incorporating the depth layer information could bring additional discriminating capability for the action feature representation. Secondly, the first aim of our dataset was to provide us with reliable training data. Indeed, in order to show the real power of our application we have to take video samples (to be used as test examples) in which more actors are present in the scene at the same time, and each of them is performing a different action. But this is left to further extension of the dataset, that it could also become publicly available in the future.

6.2.2 Nearest Neighbor classification

The first issue we met during the classification process involves making a decision about whether to perform a classification based on a single frame or on multiple

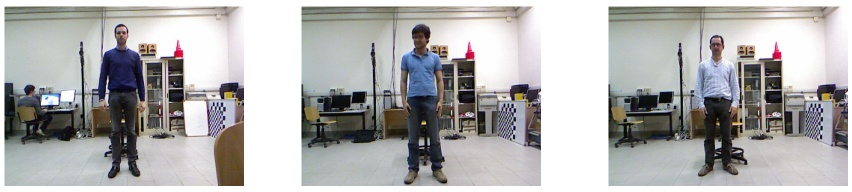


Figure 6.4: Example frames of the action *standing* taken from our dataset.

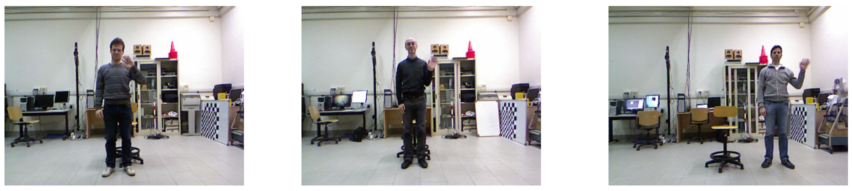


Figure 6.5: Example frames of the action *hand waving* taken from our dataset.

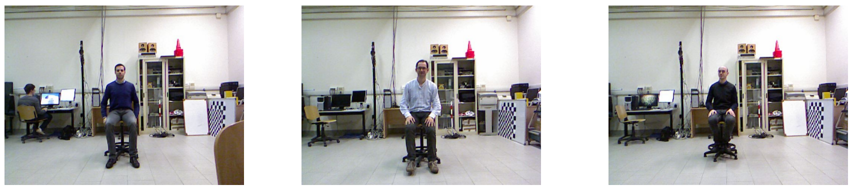


Figure 6.6: Example frames of the action *sitting down* taken from our dataset.

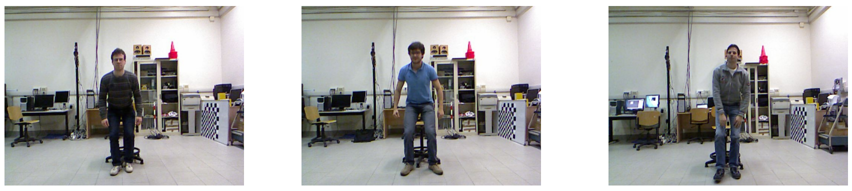


Figure 6.7: Example frames of the action *getting up* taken from our dataset.

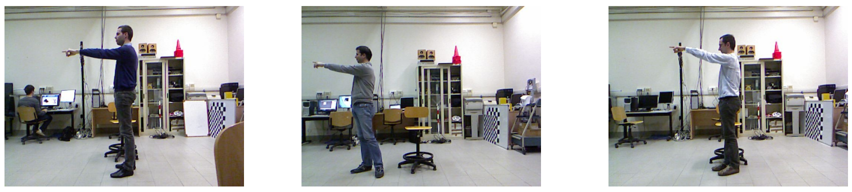


Figure 6.8: Example frames of the action *pointing* taken from our dataset.

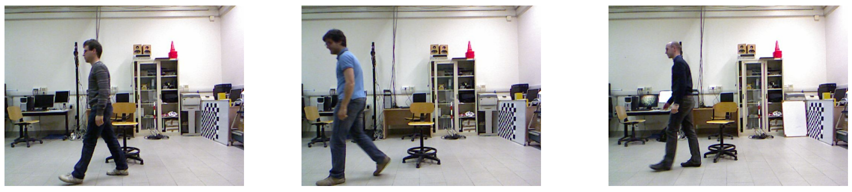


Figure 6.9: Example frames of the action *walking* taken from our dataset.

frames. Performing a classification based on a single frame means that only descriptors related to single frames are collected and classified. Classifying actions based on multiple frames means that first descriptors already computed for single frames are collected to form a more complex descriptor; then, in a second time, final descriptors are in turn collected and classified.

We decided to perform a classification based on multiple frames: since an action actually represents a sequence of movements over time, considering multiple frames could potentially provides more discriminant information to the recognition task with respect to approaches in which only a single-frame classification is performed. If a classification based on multiple frames has to be achieved, it is necessary to define two parameters:

1. the number of frames that have to be considered;
2. the way in which frames have to be captured from the entire sequence of frames that composes the action recording.

With regards to point 1 we decided to consider 10 frames. This choice is justified by [37] and by our willingness to maintain a "small" descriptor size, since we did not apply any dimensionality reduction method to descriptors. In future, we expect to apply Principal Component Analysis to descriptors, in order to maintain only the useful dimensions and to reject the others. Point 2 refers to the method in which data related to the selected frames are collected to form the final descriptor. During the training phase, it is important to distinguish three possible cases:

1. the action at issue lasts less than 10 frames;
2. the action at issue lasts exactly 10 frames;
3. the actions at issue lasts more than 10 frames.

If the action at issue lasts less than 10 frames, we perform suitable computations to bring its size to 10 frames: to this aim, the interpolation technique was used. A data vector can be interpolated in a basic way, in which dummy data are added starting from the beginning of the vector. Alternatively, interpolation can be realized by first selecting a window centered on the data vector, and then by interpolating data that stand inside the window. We chose this second approach, in this work. When the action lasts exactly 10 frames, all the information captured are used to learn the classifier. Finally, if the action lasts more than 10 frames, a sampling technique is used. We have developed the three sampling methods outlined in Table 6.1. We used the window-based sampling method with the purpose of testing the classification based on the uniform and mixed sampling technique in further improvements of this work.

Sampling technique	**Description**
Window-based sampling	*Input*: a data vector and its desired size after sampling. *Output*: the sampled vector. The algorithm simply chose a window centered on the data input vector. The windows size is equal to the desired size of the sampled vector. Finally, the window is returned as a vector.
Uniform sampling	*Input*: a data vector and its desired size after sampling. *Output*: the sampled vector. The algorithm first estimates the sampling interval based on the desired size of the sampled input vector. Then, it picks up elements according to the sampling interval. The final vector of chosen elements is returned.
Mixed sampling	*Input*: a data vector and its desired size after sampling. *Output*: the sampled vector. The algorithm combines both the window-based sampling and the uniform sampling algorithms.

Table 6.1: Sampling techniques developed in this work.

During the test phase, we decided not to rely on interpolation methods, since we observed that they do not support vectors with a high dimensionality. In doing so, it is not possible to recognize actions by means of recordings that last less than ten frames. However, this is justifiable by the real-time purpose of our application: since the application is able to execute real-time computations, it can naturally be devoted to recognize actions on-line rather than off-line and generally, on-line video streams last more than 10 frames (i.e. about 0.42 seconds).

We decided to first test the recognition performance of the 3D optical flow by means of a nearest neighbor classifier. The training phase involved four main steps:

1. Collecting raw data, namely the single-frame descriptors.
2. Computing the final descriptors by aggregating 10 single-frame descriptors with the sampling and interpolation algorithms described above.
3. Normalizing the final descriptors in order to reduce the presence of noisy data.
4. Storing the normalized descriptors related to each action performed by each actor in the dataset.

And also the test phase involved four main steps:

1. Collecting single-frame descriptors in a so called final descriptor until 10 frames are reached. When the 10 frames are exceeded, a window sampling is applied to obtain a 10-frames final descriptor.
2. The final descriptor is normalized in order to reduce the presence of noisy data.
3. The distance between the test descriptor and all the training examples in the dataset is computed.
4. The label related to the training descriptor that has the shortest distance to the test descriptor is chosen as predicted class.

We used two types of distance function:

- the Euclidean distance function
- the Mahalanobis distance function

The Euclidean distance function for fixed-length vector of real numbers is defined as:

$$d(x,y) = (\sum_{i=1}^{m}(x_i - y_i)^2)^{1/2}.$$

where x_i and y_i are points in $\mathbb{R}^m$, and m represents the dimensionality of the data vector. While, the Mahalanobis distance function is defined for a multivariate vector $x = (x_1, x_2, \ldots, x_n)^T$ from a group of values with mean $\mu = (\mu_1, \mu_2, \mu_3, \ldots, \mu_n)^T$ and covariance matrix S as:

$$d_M(x) = \sqrt{(x-\mu)^T S^{-1}(x-\mu)}.$$

The approach outlined in relation to the test phase is suitable to achieve the recognition of isolated actions, but it is less suitable to perform a continuous recognition task. Indeed, in order to continuously recognize actions, it is necessary to find a way to estimate the beginning and the ending of each action in a sequence. There are two possible ways to easily obtain an on-line segmentation. The former is based on the observation of the classification results on successive frames while the latter is based on the confidence of the classifier. Classification results can be used in successive frames to claim that "the action A is really the action performed by the actor if and only if the classifiers predicts the class A for at least N successive frames" for example. The confidence of the classifier can be used to segment a sequence of actions if it is set as a threshold. This means that if the classifier predicts the action A with a confidence greater than a defined threshold, then the action A is really the action performed by the actor in the scene and a new action, after A, will start subsequently. In this work, the continuous recognition task is left to future developments.

6.3 Experimental results

This section discusses the experimental results we achieved by performing a nearest neighbor classification on final descriptors. Tests have been executed following a leave-one-out approach: first we chose an actor from the dataset to use its recordings as test bed, then we trained the classifier with the examples related to the other five subjects. Finally, we collected the classification results related to the unseen actor with respect to the training samples. The process described is then performed for each actor in the dataset. Unfortunately, final descriptors containing kinematic features, such as the divergence and the curl, did not enable us to achieve a consistent action recognition so they are not reported in this discussion. By contrast, the other kind of computed descriptor (see Section 6.1.2)

	STA	**HAW**	**SIT**	**GET**	**POI**	**WAL**
STA	**0.67**	0.17			0.17	
HAW	0.17	**0.50**			0.17	
SIT			**0.83**			
GET		0.17		**1.00**		0.17
POI	0.17	0.17	0.17		**0.50**	
WAL						**0.83**

Table 6.2: Confusion matrix related to a nearest neighbor classification in which the Mahalanobis distance is computed. In the matrix: **STA** stands for *standing*, **HAW** stands for *hand waving*, **SIT** stands for *sitting down*, **GET** stands for *getting up*, **POI** stands for *pointing* and finally **WAL** stands for *walking*.

	STA	**HAW**	**SIT**	**GET**	**POI**	**WAL**
STA	**0.83**	0.33	0.17		0.33	
HAW		**0.50**				
SIT			**0.83**			
GET		0.17		**1.00**		
POI	0.17				**0.67**	
WAL						**1.00**

Table 6.3: Confusion matrix related to a nearest neighbor classification in which the Euclidean distance is computed. In the matrix: **STA** stands for *standing*, **HAW** stands for *hand waving*, **SIT** stands for *sitting down*, **GET** stands for *getting up*, **POI** stands for *pointing* and finally **WAL** stands for *walking*.

allows to recognize actions quite well and results have to be considered with respect to that particular descriptor calculus.

Results have been collected in the form of confusion matrices and they are shown in Table 6.2 and Table 6.3. Table 6.2 is related to a nearest neighbor classification in which the Mahalanobis distance is used, while Table 6.3 is referred to a nearest neighbor classification based on the Euclidean distance computation.

From Table 6.2 and 6.3, it could be observed that Mahalanobis distance leads to poor results with respect to the Euclidean distance. Indeed, the computation of Mahalanobis distance require the calculus of the covariance matrix and its inverse, that it is high costly when dealing with data characterized by having high dimensionality. The result is that our application suffers the calculus of the Mahalanobis distance and we had to execute recording at low speed in order to

	Accuracy	**Precision**
NN with Mahalanobis distance	0.78	0.72
NN with Euclidean distance	0.80	0.74

Table 6.4: Accuracy and precision computed for the confusion matrices illustrated in Table 6.2 and Table 6.3.

be sure not to lose data.

Other remarkable observations could be done with regards to the recognized actions. Experimental results show that our application recognizes best the actions that involve the movement of the entire human body (e.g. *getting up and walking*), while the recognition of actions in which only arms are moved obtains poor results with respect to the first one. This could have several reasons. First, the voxel grid filter applied in the detection phase might be too much restrictive, so that only few points of the actors' arms (namely few velocity vectors related to the actors' arms) could actually be captured. Secondly, the descriptor might not summarize well the flow information: in this direction a descriptor in which the flow in each voxel is summarized by the median velocity vector could be implemented. Finally, since nearest neighbor classifier does not provide exceptional performance with respect to other classifier, it could be changed to see if a more effective recognition is achieved. In future developments of this work, we expect to use the *Gaussian Mixture Model* to classify actions.

Eventually, Table 6.4 shows the accuracy and precision computed from Table 6.2 and Table 6.3. Results are self-explanatory.

Chapter 7

Conclusions

This chapter summarizes the contribution of the thesis and discusses avenues for future research.

This thesis aims to contribute to the field of human action recognition. We have proposed a new approach to recognize human actions with RGB-D data. We focused on developing a new way to quickly compute typical features of the 2-dimensional field. So the contribution is twofold with regards to the features extraction step: first we extended features traditionally computed from RGB images to the RGB-D world, and secondly we invented a rapid approach to capture them. Time performances in this work are important: indeed it is targeted to applications in the video surveillance field, and we aimed to develop a real-time recognition system that could be usable in a video surveillance context. The extracted features are the 3D velocity vectors that constitute the 3D optical flow field. Computing optical flow is really a challenging problem: generally, works that rely on the 2D optical flow computation have not real-time performances since calculus are performed on each image pixel. We solved this issue by computing the 3D optical flow field only for the portions that represent people in the scene. More specifically, each person in the scene is associated with a point cloud and 3D velocity vectors are calculated by estimating correspondences between points in the clouds during the time passing. We therefore investigated on correspondence estimation problems, correspondence rejection and noise removal problems, and this is the first time, to our knowledge, that such issues are addressed in the field of human action recognition. The successive step in the recognition process regards the development of a suitable description to summarize the flow information. To this aim, we computed a 3D grid surrounding each person in the scene and successively, we captured relevant flow information from each voxel of the grid. The 3D voxel grid combined to the information extracted from the flow represents the descriptor computed with regards to a single frame. Since we decided to perform a classification based on multiple frames, we first collected a series of descriptors. The entire collection is then treated as a unique vector to be normalized. Normalization enables to partially discard the presence of noise in the raw data. Finally, the system is tested on a newly created dataset. The dataset contains six different action classes performed by six different actors. The creation of a new dataset is motivated by the actual lack of publicly available datasets in which RGB-D data are provided. Indeed, the existing datasets are mainly devoted to tasks in the field of personal and assistant robotics, therefore they did not meet our intents. A nearest neighbor classification is performed between collections of training and test examples taken from our dataset. In particular, two nearest neighbor classifications are performed: the former is based on the Mahalanobis distance function, while the latter is based on the Euclidean distance function. We achieved the 78% accuracy for the overall dataset with regards to the use of the Euclidean distance function, while we obtained a 72% accuracy with regards to the use of the Mahalanobis distance. Reaching an accuracy of 78% represents a good result for our application, since we developed the system to be real-time even at the expense of possible losses in accuracy.

7.1 Future improvements and developments

The system developed in this work can be improved in several parts. As for the features extraction step, new algorithms able to reject “bad” correspondences and noisy data can be implemented. Also the results visualization can be improved by minimizing the amount of data passed via ROS messages. The phase regarding the computation of a suitable description for the flow features can be enhanced by calculating more complex descriptors, such as 3-dimensional or 4-dimensional histograms of optical flow and their variants. It is however important to pay attention to the overall descriptor size at the end of the computation. Indeed, it is possible that after having calculated a more complex descriptor, its size grows by a relevant amount. If no dimensionality reduction method is applied to such a descriptor, the classification phase could provide poor results in term of both time performances and accuracy. Therefore, the selection of a new descriptor has to be combined with the investigation about traditional dimensionality reduction methods, like *Principal Component Analysis*. The new descriptor should also be designed to be view-invariant, so compensating for the major limit of all the approaches based on the optical flow calculus. Finally, as regards the classification phase, an advanced classifier should be used instead of nearest neighbor. We expect to use the *Gaussian Mixture Model* classifier to this aim.

Once our application achieves a robust recognition of isolated actions, it can be extended to accomplish more complex tasks. There are two possible extensions of the current work that can be taken into consideration. The former concerns the continuous recognition of actions. In the continuous recognition task, an actor performs several actions in a certain sequence and the recognizer has to correctly detect each executed action. This means that the algorithm has to recognize the series of actions, but in addition it has also to locate temporally the actions, and to assign to each temporal interval the correct action. While, the latter regards the recognition of actions executed by two or more people. Continuous recognition and recognition of group actions are currently two active topics in the human action recognition research field.

Bibliography

[1] S. Ali and M. Shah, “Human action recognition in videos using kinematic features and multiple instance learning,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 2, pp. 288 –303, feb. 2010.

[2] F. Basso, “Rgb-d people tracking by detection for a mobile robot,” Master’s thesis, University of Padua, 2011.

[3] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri, “Actions as space-time shapes,” in *Proc. Tenth IEEE Int. Conf. Computer Vision ICCV 2005*, vol. 2, 2005, pp. 1395–1402.

[4] A. Bobick and J. Davis, “The recognition of human movement using temporal templates,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 3, pp. 257 –267, mar 2001.

[5] O. Brdiczka, M. Langet, J. Maisonnasse, and J. Crowley, “Detecting human behavior models from multimodal observation in a smart home,” *Automation Science and Engineering, IEEE Transactions on*, vol. 6, no. 4, pp. 588 –597, oct. 2009.

[6] S. Carlsson and J. Sullivan, “Action recognition by shape matching to key frames,” in *IEEE Computer Society Workshop on Models versus Exemplars in Computer Vision*, 2001.

[7] K. M. G. Cheung, S. Baker, and T. Kanade, “Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture,” in *Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition*, vol. 1, 2003.

[8] N. Dalal, “Finding people in images and videos,” Ph.D. dissertation, Institut National Polytechnique de Grenoble / INRIA Grenoble, 2006.

[9] S. Danafar and N. Gheissari, “Action recognition for surveillance applications using optic flow and svm,” in *Proceedings of the 8th*

Asian conference on Computer vision - Volume Part II, ser. ACCV'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 457–466. [Online]. Available: http://portal.acm.org/citation.cfm?id=1775728.1775783

[10] J. Davis and A. Bobick, "The representation and recognition of human movement using temporal templates," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, jun 1997, pp. 928 –934.

[11] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie, "Behavior recognition via sparse spatio-temporal features," in *Proc. 2nd Joint IEEE Int Visual Surveillance and Performance Evaluation of Tracking and Surveillance Workshop*, 2005, pp. 65–72.

[12] A. Efros, A. Berg, G. Mori, and J. Malik, "Recognizing action at a distance," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, oct. 2003, pp. 726 –733 vol.2.

[13] S. M. Filippo Basso, Matteo Munaro and E. Menegatti, "Fast and robust multi-people tracking from rgb-d data for a mobile robot," in *Proc. of the 12th Intelligent Autonomous Systems (IAS) Conference, Jeju Island (Korea)*, 2012.

[14] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri, "Actions as space-time shapes," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 12, pp. 2247–2253, December 2007.

[15] J. Han and B. Bhanu, "Human activity recognition in thermal infrared imagery," in *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, june 2005, p. 17.

[16] M. Holte and T. Moeslund, "View invariant gesture recognition using 3d motion primitives," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, 31 2008-april 4 2008, pp. 797 –800.

[17] M. Holte, T. Moeslund, N. Nikolaidis, and I. Pitas, "3d human action recognition for multi-view camera systems," in *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 International Conference on*, may 2011, pp. 342 –349.

[18] N. Ikizler, R. G. Cinbis, and P. Duygulu, "Human action recognition with line and flow histograms," in *Proc. 19th Int. Conf. Pattern Recognition ICPR 2008*, 2008, pp. 1–4.

[19] G. Johansson, “Visual perception of biological motion and a model for its analysis,” *Attention, Perception, & Psychophysics*, vol. 14, pp. 201–211, 1973, 10.3758/BF03212378. [Online]. Available: http://dx.doi.org/10.3758/BF03212378

[20] S. Ju, M. Black, and Y. Yacoob, “Cardboard people: a parameterized model of articulated image motion,” in *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*, oct 1996, pp. 38 –44.

[21] Y. Ke, R. Sukthankar, and M. Hebert, “Efficient visual event detection using volumetric features,” in *Proc. Tenth IEEE Int. Conf. Computer Vision ICCV 2005*, vol. 1, 2005, pp. 166–173.

[22] A. Kläser, M. Marszałek, and C. Schmid, “A spatio-temporal descriptor based on 3d-gradients,” in *British Machine Vision Conference*, sep 2008, pp. 995–1004. [Online]. Available: http://lear.inrialpes.fr/pubs/2008/KMS08

[23] I. Laptev and T. Lindeberg, “Space-time interest points,” in *Proc. Ninth IEEE Int Computer Vision Conf*, 2003, pp. 432–439.

[24] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, “Learning realistic human actions from movies,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition CVPR 2008*, 2008, pp. 1–8.

[25] W. Li, Z. Zhang, and Z. Liu, “Action recognition based on a bag of 3d points,” in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, june 2010, pp. 9 –14.

[26] J. Liu, S. Ali, and M. Shah, “Recognizing human actions using multiple features,” in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, june 2008, pp. 1 –8.

[27] M. Marszalek, I. Laptev, and C. Schmid, “Actions in context,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition CVPR 2009*, 2009, pp. 2929–2936.

[28] B. Ni, G. Wang, and P. Moulin, “Rgbd-hudaact: A color-depth video database for human daily activity recognition,” in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, nov. 2011, pp. 1147 –1153.

[29] J. C. Niebles, H. Wang, and L. Fei-Fei, “Unsupervised learning of human action categories using spatial-temporal words,” *Int. J. Comput.*

Vision, vol. 79, pp. 299–318, September 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1380728.1380729

[30] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. Aggarwal, H. Lee, L. Davis, E. Swears, X. Wang, Q. Ji, K. Reddy, M. Shah, C. Vondrick, H. Pirsiavash, D. Ramanan, J. Yuen, A. Torralba, B. Song, A. Fong, A. Roy-Chowdhury, and M. Desai, "A large-scale benchmark dataset for event recognition in surveillance video," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, june 2011, pp. 3153 –3160.

[31] M. Popa, A. Koc, L. Rothkrantz, C. Shan, and P. Wiggers, "Kinect sensing of shopping related actions," in *Constructing Ambient Intelligence: AmI 2011 Workshops*, undefined, K. Van Laerhoven, and J. Gelissen, Eds., Amsterdam, Netherlands, 11/2011 2011.

[32] R. Poppe, "A survey on vision-based human action recognition," *Image and Vision Computing*, vol. 28, no. 6, pp. 976–990, 2010. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0262885609002704

[33] H. Ragheb, S. Velastin, P. Remagnino, and T. Ellis, "Human action recognition using robust power spectrum features," in *Proc. 15th IEEE Int. Conf. Image Processing ICIP 2008*, 2008, pp. 753–756.

[34] M. D. Rodriguez, J. Ahmed, and M. Shah, "Action mach a spatio-temporal maximum average correlation height filter for action recognition," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition CVPR 2008*, 2008, pp. 1–8.

[35] R. B. Rusu, J. Bandouch, F. Meier, I. A. Essa, and M. Beetz, "Human action recognition using global point feature histograms and action shapes," *Advanced Robotics*, vol. 23, no. 14, pp. 1873–1908, 2009.

[36] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[37] K. Schindler and L. van Gool, "Action snippets: How many frames does human action recognition require?" in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, june 2008, pp. 1 –8.

[38] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local svm approach," in *Proc. 17th Int. Conf. Pattern Recognition ICPR 2004*, vol. 3, 2004, pp. 32–36.

[39] P. Scovanner, S. Ali, and M. Shah, “A 3-dimensional sift descriptor and its application to action recognition,” in *Proceedings of the 15th international conference on Multimedia*, ser. MULTIMEDIA ’07. New York, NY, USA: ACM, 2007, pp. 357–360. [Online]. Available: http://doi.acm.org/10.1145/1291233.1291311

[40] J. Sung, C. Ponce, B. Selman, and A. Saxena, “Human activity detection from rgbd images,” in *Plan, Activity, and Intent Recognition*, ser. AAAI Workshops, vol. WS-11-16. AAAI, 2011.

[41] ——, “Unstructured human activity detection from rgbd images,” in *International Conference on Robotics and Automation, ICRA*, 2012.

[42] D. Tran and A. Sorokin, “Human activity recognition with metric learning,” in *Proceedings of the 10th European Conference on Computer Vision: Part I*, ser. ECCV ’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 548–561. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88682-2_42

[43] P. K. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea, “Machine recognition of human activities: A survey,” *IEEE Trans. Circuits Syst. Video Techn.*, vol. 18, no. 11, pp. 1473–1488, 2008.

[44] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid, “Evaluation of local spatio-temporal features for action recognition,” in *British Machine Vision Conference*, London, United Kingdom, Sep. 2009, cLASS. [Online]. Available: http://hal.inria.fr/inria-00439769/en/

[45] D. Weinland, R. Ronfard, and E. Boyer, “Free viewpoint action recognition using motion history volumes,” *Comput. Vis. Image Underst.*, vol. 104, no. 2, pp. 249–257, Nov. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.cviu.2006.07.013

[46] G. Willems, T. Tuytelaars, and L. Gool, “An efficient dense and scale-invariant spatio-temporal interest point detector,” in *Proceedings of the 10th European Conference on Computer Vision: Part II*, ser. ECCV ’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 650–663. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88688-4_48

[47] Y. Yacoob and M. Black, “Parameterized modeling and recognition of activities,” in *Computer Vision, 1998. Sixth International Conference on*, jan 1998, pp. 120 –127.

[48] A. Yang, S. Iyengar, S. Sastry, R. Bajcsy, P. Kuryloski, and R. Jafari, “Distributed segmentation and classification of human actions using a wearable

motion sensor network," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, june 2008, pp. 1 –8.

[49] A. Yilmaz and M. Shah, "Actions sketch: a novel action representation," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, june 2005, pp. 984 – 989 vol. 1.

[50] H. Zhang and L. E. Parker, "4-dimensional local spatio-temporal features for human activity recognition," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, sept. 2011, pp. 2044 –2049.