CIMAT

Centro de Investigación en Matemáticas, A.C.

RGB-D NEAR ONLINE MULTI-OBJECT TRACKING

THESIS

Submitted in partial fulfillment of the
Requirements for the degree of

Master of Science

with orientation in

Computer Science and Industrial Mathematics

by

Orlando García Álvarez

Thesis supervisor:

Jean-Bernard Hayet, PhD.

Thesis examiners:

Héctor Manuel Becerra Fermín, PhD.
Rafael De La Guardia González, PhD.

Guanajuato, Gto., December 15, 2018

CIMAT Centro de Investigación en Matemáticas, A.C.

RGB-D NEAR ONLINE MULTI-OBJECT TRACKING

THESIS

Submitted in partial fulfillment of the
Requirements for the degree of

Master of Science

with orientation in

Computer Science and Industrial Mathematics

by

Orlando García Álvarez

Thesis supervisor:

Jean-Bernard Hayet, PhD.

Thesis examiners:

Héctor Manuel Becerra Fermín, PhD.

Rafael De La Guardia González, PhD.

Guanajuato, Gto., December 15, 2018

Abstract

RGB-D Near Online Multi-Object Tracking (NOMT)

by

Orlando García Alvarez

Maestría en Ciencias in Ciencias de la Computación y Matemáticas Industriales

Centro de Investigación en Matemáticas A. C., Guanajuato

Dr. Hector Becerra, Chair

Dr. Rafael de la Guardia, Co-chair

Tracking pedestrians and other moving objectives over RGB-D data is increasing in popularity, due to the access of cheaper and more precise depth devices. Keeping a consistent identity of these objectives over time is a combinatorial problem that, just like detection itself, is susceptible to occlusions, non-smooth movement, and even more so when the nature of a target so dynamic as a human being makes it difficult to maintain a consistent appearance measurement without using modern GPU-based techniques.

In the literature, few tracking strategies have been designed to fully exploit 3D data and keep operation in real-time performance. In this thesis, we present a tracking pipeline that manages to quickly solve data-association on targets through a graphical model inspired by W. Choi's Near-Online Multiple Tracking (NOMT) [CVPR15], with a redesigned energy optimization model for better usage of depth and color data and with the use of a simple histogram-based descriptor which is low cost for the processor and copes well with non-static targets.

Along with this strategy, we present a new easy-to-use tracking software library that is compatible with ROS, providing code for faster RGB-D based tracker development. We also present an implementation and evaluation of our RGB-D NOMT with tests made over ground-truth data taken with Kinect sensors.

To my mother, for her unconditional love and support.
To my grandmother, for she lovingly raised me, always so proud of me.

Contents

Acknowledgments

I would like to express my very great appreciation to Dr. Jean-Bernard Hayet, for giving me so many chances and believing in me. For his valuable direction and time invested in this research, for which I have my highest regards. Also thanks to Dr. Hector Becerra and Dr. Rafael de la Guardia, for their interest and valuable feedback on integrating this work.

Thanks to CIMAT for all resources made available to me, facilities, library, digital resources, food service and closeness to all fellow researchers that provided helpful tips and advice. Also thanks to Concejo Nacional de Ciencia y Tecnología (CONACYT), for the two year financial support during master's degree program. And many thanks to INTEL Labs for the scholarship given during my participation on the "Move and let Move" project.

To my mom, brothers and sisters, whom are always in touch, checking on me, looking up to my achievements and cheering me up, for whom I push myself the furthest I can, my most sincere thanks and love belongs to them. I also thank my close friends back home, Reyna Ovalle, Myriam Olalde, Lucero Zertuche, whom are always happy to know about my progress and misadventures.

To all laboratory colleagues, particularly to Edgar Martínez, Noe "Pity" Aldana, Emmanuelle Ovalle, Giselle Megchun and all those who stayed for a little while, which made the whole experience much more enjoyable, and most importantly to Andrei Raya and Yuriria Acuña, which were always willing to keep me company when I most needed it.

Chapter 1

Motivation and Theoretical Background

1.1 Introduction

In our everyday life, we are instantaneously able to identify ourselves, family members, friends and colleagues at simple sight. Even more, we are able to identify strangers on the streets and keep track of them along time by memorizing in some way their body type, their haircut, their clothing and the accessories they are wearing. Also, we have a sense of how far they are relative to ourselves. If we track a person and see the same person moments later, for example, when we turn our head back and forth, or right after blinking, we can keep on tracking and estimating his trajectory at some extent.

Computer vision and robotics aim to emulate, ease and outperform human beings in such activities, those that we do everyday, or to allow robots to do what we cannot do, like keeping track of hundreds of people at the same time. This objective of surpassing human capabilities is shared with many other disciplines that end up providing us with better and more sophisticated hardware and tools like smaller devices, more powerful processors and better cameras. An easy access to these technologies permits for new uses and applications for them, individually or combined in what we call a system or robot.

These words are commonly used interchangeably. In an abstract sense, both are the combination of computers that perform reasoning and problem solving on data collected by sensors. They perceive the world with sensors such as cameras, lasers, contact bumpers, heat/light and capacitive sensors, etc. and use them to determine actions, like navigating, changing the robot configuration, making sounds or showing a response on screen. When consistently solving a high-level problem, we call it intelligent system, e.g., camera systems that perform face recognition or pedestrian tracking for the sake of security. An autonomous car that employs mounted cameras, proximity sensors and specialized software for driving is another example. Flying drones with attached cameras that stream video for performing terrain classification also belong to this class of systems.

The use of cameras on these systems is one of the easiest ways to collect data from the world. Cameras can take pictures or a series of them in the form of video, and in this case the pictures are called frames. Computer Vision is the discipline that works with this type of input data and tries to extract as much information as possible from pictures/frames. This information may be the robot current pose, the relative distance to the objects in sight, and their identification/classification, among other.

One of the most challenging application is identifying persons in a single frame of a video sequence. It is by itself a difficult task for a computer whereas it is a skill taken for granted among humans as, without apparent effort, we can tell the difference between a person that is physically standing in front of us and a person in a photograph, a painting, a mirror or a reflection. We can differentiate a human from any other human-like entities, even though we can fail at this task under certain circumstances.

For a robot, the software component that identifies the presence of a person on every single frame is simply called detector, and every found person is called detection. Ensuring that detectors give accurate enough results is critical for a tracker, so that a target can be identified/recognized individually the whole time the person is in sight, in a succession of consecutive frames.

A tracking algorithm attempts to associate detections that correspond to the same person, that is now referred to as a target in this context. This problem of maintaining the identity of several targets along a sequence of multiple detections is what is called a data-association problem, and it is at the core of all the tracking algorithms. It may become extremely complicated when dealing with occlusions among targets or with clutter, with strong changes in the video acquisition conditions, etc. It may be solved, as in this thesis, through optimization algorithms that attempt to keep a reliable estimation and prediction on the motion of the tracked objects, since a single target is associated with a unique trajectory, composed by every detection belonging to this target.

Once the aforementioned problem is solved, it could be said that the robot has gained "consciousness" of the presence of other humans surrounding it and of their motion. As a result, it is not only able to identify a target but also to keep track of its position relative to some coordinate system, which allows for many applications. With this information, a service robot can plan its movement and navigate, predict the motion of every person that is being tracked, move while avoiding collision, or approach a group of persons to greet and give support for any given task.

However, there are many general challenges to this problem, even though robots become more and more well suited with powerful sensors and processing capabilities:

- Sensors like cameras and lasers are susceptible to occlusions, meaning that some objectives may become hidden by others, by obstacles, or be just too far to be sensed with data accurate enough to be used by the system.
- Sensors have specific frame rates, frequencies and formats in which collected data can be read. This creates synchronization issues, incomplete segments of data and a big load of information to process when the sensor produces dense sampling on the real world, like HD cameras.

- The way to represent the identity of a target is critical. Measuring similarity between observations of targets is still an open problem that has many interpretations on different approaches. It can be tackled by considering the geometry, the observed trajectory, the color, or the result of a more complex analysis like face recognition, limb dynamics and pose estimation. All these approaches require more and more processing power. However, failing to identify the same individual in a sequence of frames could mean that either it went out of sight or it got occluded. In case of error, an "identity switch" may happen, meaning that the similarity measure resulted in a confusion on the tracking system.

Much work has been done on pedestrian tracking over video sequences, and the idea of using a measure of how far is a target from the observer, called depth, is nothing new. Its combination with color information from the frames has shown to perform much better as it provides more information on the scene and reduces mistakes in data association.

Now, with the presence of relatively new devices such as the XBox Kinect, the ASUS Xtion or the Intel RealSense, which are called RGB-D Sensors, it is more plausible for a robot to collect tracking observations with such mounted devices, obtaining both color and depth, and allowing for 3D point-clouds to be constructed and new techniques to be used.

With the new paradigm of RGB-D data comes also new challenges:

- RGB data and depth data usually come in different streams of data, and though the frequency is similar, often synchronization errors show up on some frames, making a lot of pixels missing their real depth information, which varies a lot, especially when depth nears maximum operation ranges.

- No depth sensor is capable of measuring infinite depth, due to unavoidable discretization both on image pixel sizes and depth perception techniques. Higher resolutions allow for longer depth-ranges that can be accurately perceived, still with loss of density in re-projected points near the horizon that are physically further away from the sensor.

- RGB-D sensors are now recovering huge amounts of pixels, resulting in large 3D point-clouds with redundant information that are often down-sampled to reduce the burden of processing time at each iteration.

- The access to 3D point-clouds allows for more geometric analysis and better understanding on the pose of the target. Exploiting this amount of information and also maintaining real-time performance is difficult and requires faster ways to measure similarity. Also, the problem grows when attempting to track non-rigid objects which have limbs that occlude themselves. This makes it necessary[1] to encode somehow their dynamics when in search of optimal appearance extraction: to compare localized qualitative data that should be repeatedly matched over future instances of the same target in different positions.

[1]Such precise similarity measuring for dynamic targets is beyond the scope of this work, as we settle with a cheap yet robust enough strategy that can be applied without too much computing power as will be presented on Chapter 4

1.2 Related Work

Many modern person detectors are now working with RGB-D sensors, as they allow better depth ranges, and they are more precise and cheaper, and many works have been done on people detection on RGB-D data. Taking into account the depth information, one can perform more geometric evaluations for the pose and the body part detection, through the 3D point-cloud that can be re-constructed or that may be given directly by the sensor, reducing some of the processing burden and allowing for more complex high-level algorithms to be designed.

On the other hand, recent tracking systems are able to perform detection over RGB-D data to identify persons or various objects, but the tracking is done using many other geometric and similarity measurements that tend to dismiss much of the data collected and used by the detector, for the sake of performance.

There are still few RGB-D based trackers in the literature. In the following, we review works that have been published in this area.

1.2.1 People detection

Techniques for detecting a human being in a photograph have been developed by considering only color information, and with great success even on recent years, so it makes sense that the same ideas could still be useful for RGB-D systems, though techniques that also take advantage of depth data are expected to operate better and faster. Classic strategies have reached a point of modernization that copes with real-time performance, like the usage of classification for detection candidates using Support Vector Machines (SVM) on block-based Histogram of Gradients (HOG) descriptors as done by [10, Rodriguez], which yields fast detection latency and takes advantage of both the RGB and depth channels.

Other state-of-the-art techniques like contour and template matching on anthropomorphic figures such as those from [14, Jafari] are able to perform detection in RGB-D images and they approximate the target orientation through the training of orientation-based templates for the human upper-body silhouette while dealing with targets that are further away from the sensor, meaning imprecise depth information.

Still, these classic or ad-hoc approaches suffer from the same problems as their 2D versions, in that they need to make special considerations for detecting targets that are partially shown in the image and in positions other than standing. This problem has given rise to more complex state-of-the-art machine-learning and neural-network based techniques, as in the works of [37, Tian], [40, Zhou], [18, Liu] and [24, Ophoff], among others.

1.2.2 Pose estimation and body part recognition

These approaches tend to be more greedy on resources but with far better precision, since they take detections further, into more details on the target within a point-cloud, extracting more knowledge on the target identity. Works like the classic Kinect algorithm from [31, Shotton] are able to cluster the body parts through Randomized Decision Forests to give an optimal classification of the joints found in each person point-cloud for their individual tracking to be made efficiently on GPU, also requiring the training of the classifier.

Another approach is to fit a skeleton model directly on the clustered point-clouds, minimizing the error through Expectation Maximization (EM) between the model and the observed limbs as in the work from [2, Barros]. Hence, the knowledge on real pose is more accurate. It is also worth mentioning recent works from [5, Cao] which operates by using RGB data only and still manages to estimate with high accuracy the 2D human skeleton pose of almost every person in an image. It outperforms the techniques mentioned earlier, as it accurately detects people at non-standing positions, which is a major problem for most detectors. Operating without prior knowledge about human detections, it trains instead a Convolutional Neural Networks (CNN) that finds loose body-parts in the image and a corresponding degree of association between parts is found. Using confidence maps and affinity fields fed to a greedy inference process, 2D key-points for all people in the image are found and they could be projected to 3D if depth data is made available. Body part recognition and pose estimation has boomed in recent years as in works like [9], [7], [12], [13],[16] and [39], due to all the possibilities and benefits in many areas, specially for tracking.

Let us note that most of these methods are accurate, though expensive for the processor and with low frame-rate on most consumer hardware. If the detected pose is made available for a tracking system, measuring and comparing a pair of detections over time can be done even more accurately and ensure almost no mistake when keeping the identity of the target. This gives a chance to establish possible anticipations and to handle model occlusions, orientation, position, geometry and information about the target dynamics, making it harder for the software to fall into identity switches and make mistakes when re-identifying an already seen target.

1.2.3 Target Matching

Re-identification is important since we need to be able to recognize an already seen target and recollect our knowledge on it. Though more importantly, tracking needs to do this also at a smaller scale, by recognizing the same target across frames, in a process called data-association. In the work of [22, Munaro], a tracking module receives detections as an input and solves the data association problem as the maximization of a joint likelihood encoding the probability of ground plane motion and color appearance, using the RGB color histograms of the points corresponding to a set of parallelepipeds overlaid on the target body, meaning that a rough match is done by matching color distributions across the target limbs over each frame, though not taking as much advantage of the depth data as it could be possible.

Other approaches are similar to patch based matching, just like in [23, Nascimiento], where they find interest points in a pair of targets and each patch around the point is combined with depth data to turn it into a voxelized key-point that is described by a binary vector, allowing for a matching of the objective across different frames.

In the work of [8, do Monte], a more global description of the target, rather than a piecewise approach, is achieved by using point-cloud descriptors. This descriptor is what they called a Globally Aligned Spatial Distribution, which takes the 3D point-cloud from the partial view of a given object, by estimating a reference frame for the point-cloud and making it pose-invariant. It describes the shape and the spatial distribution of color along an aligned binning. Though this will result more helpful on static objectives, it could be used as another option when measuring short-term similarity.

Once a detection and a matching strategy is defined, only the tracking strategy is missing. The simplest is maybe the nearest-neighbors approach, that just matches the most similar detection to the closest most similar detection. However, many difficulties arise regarding the nature of persons being tracked.

1.2.4 Tracking strategies

Among the many approaches that have been designed to implement solutions to the problem of visual tracking, we can distinguish at least three paradigms:

- Global optimization: This approach uses all the information obtained since the beginning of the execution up to the current time to solve a global problem, making it more precise but expensive when the sequence gets longer. Most of the time, this approach is preferred when the result is not needed in real-time, but it is used more like an offline evaluation or a post-processing step that is done as part of a bigger component.

- Real-time optimization: When it is critical that the result has to be obtained at the same frame-rate as the sensors of the robot, this kind of techniques use typically the last available frame to update the optimization result. It allows for navigation and decision making to avoid dangerous configurations. Examples of these approaches are Bayesian filtering-based methods (Kalman filter, particle filter...).

- Near-online or semi real-time optimization: It is a compromise between the two previous approaches, as it allows a small delay for obtaining the results. This design is also known as sliding window, as it defines a time lapse of duration τ where any information within this lapse permits the tracker to solve the optimization problem with a subset of the old data and the set of newest detections. This allows for potential corrections on previous data associations when new data could help in taking better decisions, depending on the size of this sliding window. Since trackers based on sliding windows are allowed to solve within this time lapse and need to be sure of observed trajectories when data fall outside of it, they are said to perform almost in real-time, hence the name near-online.

In recent years, many works on real time optimization have thrived. In the work of [4, Bibi], they handle synchronization and registration problems that occur when working with RGB-D sensors of different lenses, stream frequencies that does affect tracking, along with the problems of occlusions, and a 3D tracking software is presented with these issues taken into account. This is a part-based sparse tracker in a particle filter framework, where both motion and appearance are formulated in 3D, under the idea that an objective can still be tracked when at least some of its parts are visible. This makes the tracker more resistant to partial occlusions. A particle filter is a Bayesian filter defined in terms of the distribution of the state of the target. More particles are sampled from a state that has been observed with a higher probability. For handling occlusions, their representation of the targets consist of cuboids that contain a certain number of 3D points. Since the number of points may decrease significantly in all particles at once, they keep track of the density of these points to find whether it drops from a certain threshold.

Work from [35, Spinello] presents an Extended Kallman Filter-based tracking algorithm that obtains trajectories of detected persons while considering various motion models for each tracked person and then making data association using a nearest-neighbor approach. It also uses an integrated detection system based on laser data mounted on a car and classifies detections with a SVM. Advantages of coupling tightly the detector and the tracker systems are performance-related, since both benefit from the parsing of the appearance data into descriptors.

Sliding windows cut down the problem of tracking by just considering the latest received data and a fraction of the previous results, hence making it possible for the tracker to operate almost in real-time. Any software that obtains results at the same rate of incoming data samples is said to perform in real-time or online.

The newest state-of-the art techniques found newer niches on sliding window designs such as [33], [36] and [30], which are based on the training of Neural Networks, that have also been gaining more and more interest, despite the difficulties that hamper the use of deep learning techniques for multi-object tracking, like the lack of training-data and the novelty of the usage of this technique specifically on tracking, in spite of its success on other problems such as detection and matching.

With this strategy, we pick all the incoming detections and decide which ones belong to the same objective, by considering only those within a fixed duration of seconds. Decisions made on the groupings among detections older than this duration are considered final and are treated as "existing targets" in the following iterations. Detections grouped together are given a label or target ID. After each grouping is done, we obtain a growing coherent set of targets identified by a label and their observed trajectories. Details on the full decision process are presented in depth on Chapter 2.

In this work we take a probabilistic approach by measuring the selection of different groupings through an energy function and by designing the data association as an energy minimization problem which requires testing various combinations and picking the one with lowest energy.

1.3 Contributions and overall description of our approach

In the work presented in this thesis, we propose a tracking pipeline that operates naturally on top of the Robotics Operative System (ROS) platform. It works on a near-online basis and it uses a fast similarity measurement to define the proximity of two detections with RGB-D images.

We solve the tracking problem on a near real-time fashion using the sliding window paradigm mentioned above. The goal is to automatically identify objects of interest and to reliably estimate the motion of the corresponding targets over time. The challenge is to accurately group the detections into individual targets with high accuracy (i.e., solve the data association problem). Mistakes made in the identity maintenance could result in a catastrophic failure in many high level reasoning tasks, such as future motion prediction, target behaviour analysis, etc.

Inspired by Wongun Choi's Near-Online Multi-Object Tracking (NOMT) framework [6], we add a new RGB-D occlusion measurement and solve the data association problem as a Conditional Random Field (CRF). We achieve robustness by integrating multiple cues, including fast orientation-invariant RGB-D descriptor metric, target dynamics, appearance similarity, and a long term occlusion-aware trajectory regularization into the model, in addition of the possibility given by Choi's approach to revisit the previous data association decisions: the algorithm behaves like one of real time performance since it is capable of giving results at every frame, with the key difference that any decision made in the past is subject to change once more observations are available, since it solves the problem within sliding window.

In Chapter 2, we present an in-depth description of our tracking strategy, also mentioning a step-by-step description of the system. Later on, in Chapter 3, we tackle the design of the method used to solve the data-association problem and the graphical model that results when using the proposed potentials for our CRF. Chapter 4 explains the fast measurement type we associate to detections in the 3D Point Cloud. Chapter 5 presents the code we have developed for testing and experimenting our tracking software. Inspired by the software architecture of the Spencer People Tracking [17] available in the ROS platform, our components re-utilizes the ROS Interface design created by this project to create a non-ROS dependant library. Finally, Chapter 6 gives evaluations of our algorithm on challenging data-sets.

Chapter 2

Near-Online Multi Object Tracking

2.1 Useful definitions

2.1.1 Handling time

Managing time in a software can be difficult: processors operate on different speeds and, depending on the workload, shared memory, number of simultaneous operations per second and time invested on a single program, iterations are not always the same. Data acquisition frequencies are never guaranteed on real systems, and when many sources of information flow at their own frequencies, it is necessary for any software to know, at any given time, the age of incoming data by recovering a time-stamp, which uniquely identifies a nanosecond. This is critical since time is a measure that affects estimations on speed of moving objects, for example. Failing to measure varying times could lead to undesired results, hence propagating them to later components that use the tracker output.

We will consider time as a discrete ordered collection of timestamps $\{T_1, T_2, ..., T_{t-1}, T_t\}$, which for simplicity will be referred to by the set of indexes $\mathbb{T} = \{1, 2, \ldots, t-1, t\}$. An abuse of notation allows us to use indices and write the integer $i \in \mathbb{T}$ to actually point to the corresponding time-stamp T_i. With this notation, 1 is the oldest time-stamp to be given since the whole tracking began, and t indicates the current instant, at the present time. We can now define the sliding window as a time-stamp set of size τ that spans the time interval $[t-\tau, t]$ and distinguish detections done by a detector over this interval by checking whether they fall in frames with time-stamps inside the sliding window or not[1]. In practice, τ is chosen depending on the frame-rate and the time-lapse we allow the algorithm to correct on its decisions. Its value is proportional to the size of data that is processed at each iteration. If we had a sliding window as large as the total number of observed frames, we would be performing global optimization each time, going increasingly further away from real-time performance.

[1]Each distinct time-stamp will define a frame. We use the term frame to refer to the set of all detections that share the same time-stamp as the frame. When referring to the fth frame, we mean that $f \in \mathbb{T}$ corresponds to a time-stamp indexed by f.

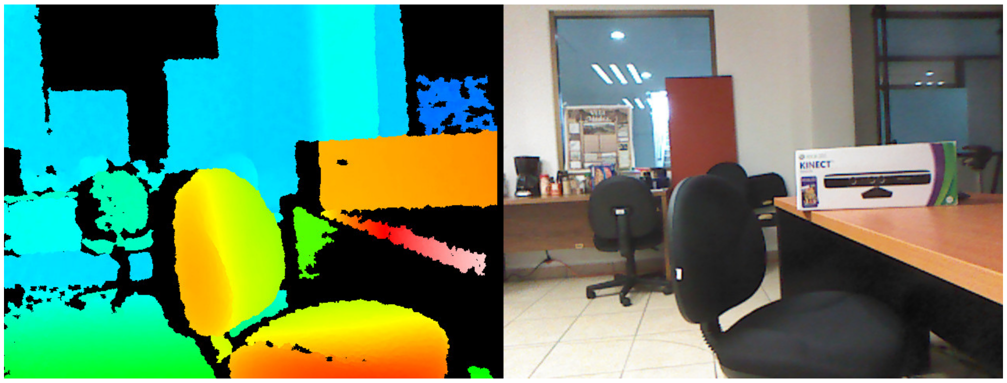


Figure 2.1: Sample data obtained with an XBox 360 Kinect. The image on the left is obtained by using an infrared sensor and a camera that triangulates depth. We present it as an intensity, here colored, where the nearest objects are red. Black regions represent blind spots, caused by out-of-range depths and reflections like on mirrors and polished surfaces. This is the case of the table, where the infrared is not perceived by the other camera.

2.1.2 Input from RGB-D Sensors

Let I_t be an image received at time-stamp t within the incoming video sequence $\mathscr{I}_1^t = \{I_1, I_2, ..., I_t\}$. This image consists in a set of pixels $I = \{(u, v) \mid 1 \leq u \leq W \wedge 1 \leq v \leq H\}$ arranged in the form of a rectangle with width W and height H over the plane $\mathbb{Z}^2$. This means that a pixel intensity can be reached by querying its 2D coordinates and it is represented as $I_t(u, v)$, which may be a scalar or a vectorial value and may be encoded in a color system, i.e., gray-scale, RGB, HSV, etc. In the case of RGB-D sensors, we may consider that for any given color image there is also an associated depth image D_t, with values $D_t(u, v)$, from a set of images $\mathscr{D}_1^t = \{D_1, D_2, ..., D_t\}$.

More specifically, RGB-D sensors provide two separate streams $\mathscr{I}_1^t$ and $\mathscr{D}_1^t$ which are supposed to be registered, i.e., the color at any pixel (u, v) of the RGB image corresponds geometrically to the depth at the same pixel of the depth image. By re-projecting color pixels with their associated depth in a 3D space with (u, v) giving the direction and $D_t(u, v)$ giving the depth along this direction, assuming that the calibration parameters of the camera are known, we can obtain a 3D point-cloud with color information of the whole scene at any time-stamp (see Figure 2.1 (b)).

However, the following problems occur with the usage of 3D point-cloud data:

- Saturation. RGB-D sensors provide $H \times W$ pixels, and even with low resolution images like 240×320, we get 76800 points that are produced at each iteration. Though modern processors are capable of working quite fast with much more data, the burden is high and reduces the processors availability for later operations. To deal with this, we perform down-sampling of the point-cloud by keeping only points separated by 5cm. More details are given in Chapter 5.

- Self-occlusions. Since the resulting point-clouds are surfaces on objects in the scene, human-like movements and limbs tend to produce surfaces that vary a lot between consecutive frames, which makes it more difficult to design robust measures on appearance. We propose a strategy to cope with the problem in Chapter 4.

- Range. RGB-D sensors nowadays are limited on their depth range, having a minimum and maximum distance of sight. Even if our detector software does not rely on depth to perform identification of targets, the point cloud to be extracted could contain incomplete data or even no data at all. Our NOMT algorithm is designed to be robust on these situations, making use of geometric information on the target position instead of its appearance when the latter is unavailable.

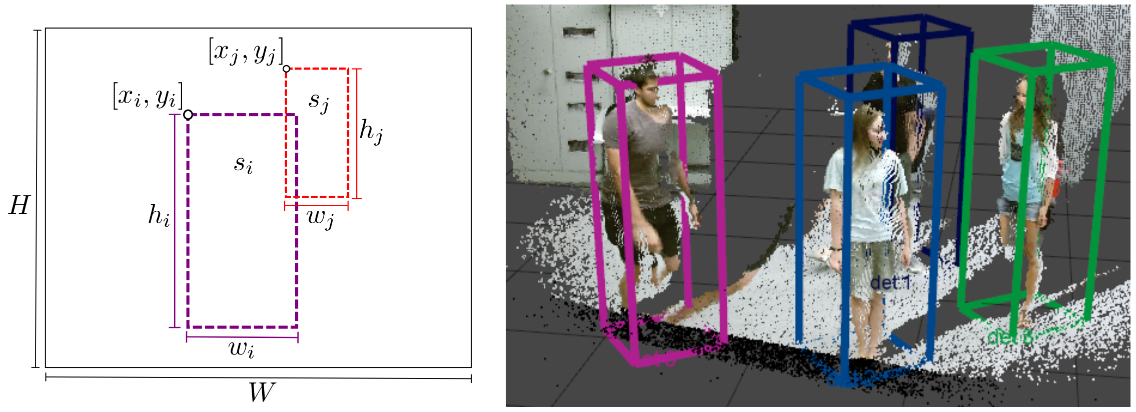
2.1.3 Detections

In the previous chapter, we have introduced the concept of detection as a detected human being in a frame. Each detection is enclosed by a bounding-box centered on its position in the world, a rectangle (in 2D) or a parallelepiped (in 3D) now depicted on Figure 2.2. The detector returns a collection of detections $\mathbf{D}_1^t = \{d_1, d_2, ..., d_N\}$ which is the list of all N detections made over the sequence of frames from time-stamp 1 to time-stamp t. In this proposed tracking system, we use bounding-cylinders since our current approach does not account for target's orientation[2] when building detection descriptors for matching since it simplifies calculations: pixels inside the bounding-cylinder form a reduced point-cloud that can be extracted by just a distance comparison from the center of a circle rather than a rotated square. Such cloud can be used to account for target's geometry, color and motion information, hence exploiting most of the data collected by the RGB-D sensor. More details on its usage are presented on Chapter 4.

To summarize we make the formal definition:

Definition 2.1.1 A Detection d_i is parameterized by a unique index i within $\mathbf{D}_1^t$, its frame number or time-stamp $t_i \in \mathbb{T}$, its bounding-cylinder $< d_i[x], d_i[y], d_i[z], d_i[r], d_i[h] >$ (in 3D, the center point (x, y, z) of a circumference of radius r and its height, respectively) and a detection score

[2]A persons could be said to be oriented as its own motion's direction, most usually aligned to the front or chest of the human body, though this could be debatable.



(a) Bounding boxes on RGB data (b) Bounding boxes on RGB-D data

Figure 2.2: (a) A detection is modeled as a bounding box over the position of the person (here in 2D). (b) Since we have RGB-D data, it is easy to find the (x, y, z) position of the person and extract the corresponding person point-cloud inside a 3D bounding-box, as depicted in the simulation. Internally, a bounding cylinder is utilized for all calculations, though a bounding-box is displayed.

s_i that measures how confident the detector is about that detection. d_i also keeps a reference to a point-cloud $\mathscr{P}_i$ which is extracted from the scene by considering only points that lie inside this bounding-cylinder. Let it be noted that d_i itself is not given a membership to any particular target.

A detection by itself is just an indication on where there could be an objective, without considering the frame in which it was found and its appearance. We use the term Track when referring to a single detection within an ordered sequence of similar detections that could belong to the same objective at different, ordered timestamps. Both track and detection can be used interchangeably here since most usages further are in a grouped manner.

2.1.4 Trajectories and Tracklets

Detections can be grouped together to form a sequence of them between different frames, not necessarily contiguous. Such ordered sets of detections are what we call a Trajectory, which, if correctly gathered, should belong to an unique objective. The tracker keeps a list of targets and their observed trajectories, past the sliding window and inside of it. We introduce the following definition for those trajectories that are strictly included within the sliding window:

Definition 2.1.2 A Tracklet $\mathcal{T}$ is a trajectory that is conformed by a set of detections $d_i \in \mathbf{D}_{t-\tau}^{t}$ and that is associated to a particular target, new or existing. Tracklets are temporary objects since they represent partial trajectories that are built by considering only detections in the latest τ frames and are re-generated at each iteration. Tracklets must conform with the following characteristics:

- $\mathcal{T}$ has a minimum length of one detection, and has a maximum length of τ detections.
- $\mathcal{T}$ has at most one detection of a particular time-stamp. Hence, two detections inside the same frame cannot belong to the same tracklet. Frame skips are allowed.
- The detections in $\mathcal{T}$ are ordered in time, meaning that the chain cannot involve decreasing time-stamps.

We make the distinction from regular trajectories that are not inside the sliding window, as those will keep growing as long as the target remains detected on the scene, whilst Tracklets are volatile candidates for extending new or existing trajectories up to frame t until new frames and detections are received.

2.1.5 Tracking

Given a video sequence $\mathscr{I}_1^t$ and a set of detections $\mathbf{D}_1^t$ done by a detector during this same interval $[1, t]$, we want to find a coherent set of targets, identified by $\mathbf{A}^t = \{A_1^t, A_2^t, ..., A_M^t\}$ where each target A_m^t is composed of a unique target identifier and a set of detections in different frames that represent the trajectory followed.

The convention of using the symbol A_m^t comes from the term Association, since we are considering a pairing between the set of detections inside the sliding window and the existing trajectories past the window, associated by the common membership to the same objective. This problem is what is usually known as a data association problem. Formally:

Definition 2.1.3 A tracked target is called Association A_m^t and is parametrized by a unique identifier $m \in \mathbf{M}$ and by an observed trajectory $\{d_p \mid p \in \mathbf{P}_m\}$ where $\mathbf{P}_m \subset \mathbf{N}$ is a subset of indices of $\mathbf{D}_1^t$.

We can then make a distinction between the detections involved in an Association by their temporal position:

- When the detections in A_m^t belong to $\mathbf{D}_1^{t-\tau-1}$ i.e. past the sliding window, we use the expression "final detections" or clean detections. They form a trajectory for which we introduce the notation A_m^{*t}, which excludes all the detections in $\mathbf{D}_{t-\tau}^t$. Hence, when writing $\mathbf{A}^{*t-1} = \{A_1^{*t-1}, A_2^{*t-1}, \dots\}$, we mean all the clean detections that can be found in target's trajectories, while excluding all the associated detections in the interval $[t-\tau, t]$, with $A_m^{*t-1} = \{d_i \in A_m^{t-1} \mid t_i \notin [t-\tau, t]\}$.
- Otherwise, detections inside the sliding window are referred to as loose detections, and they are susceptible to move from an association to another. Since each iteration brings new detections and since those detections that get past the sliding window become clean detections, this operation of associating loose detections to clean detections is called augmentation.

Summing up, we want to find sub-sets of $\mathbf{D}_1^t$ that coherently correspond to single targets and that are as large as possible. Hence, the tracking task is defined as to solve the following problem

$$(\{\mathscr{I}_1^t, \mathscr{D}_1^t\}, \mathbf{D}_{t-\tau}^t, \mathbf{A}^{t-1}) \rightarrow \mathbf{A}^t \tag{2.1}$$

at each time frame t, where $\mathbf{A}^t$ represents the set of tracked targets, or objectives, up to t. As it appears in this formulation, we make use of the previous associations, made at $t-1$, to infer the new ones at t.

The tracking algorithm needs to solve the data association problem between the existing targets and all the detections found in the sliding window $[t-\tau, t]$, associating them to existing targets or forming new ones.

2.2 Data Association

The data association problem is applied only over loose detections $\mathbf{D}_{t-\tau}^t$ that are not yet associated to existing targets, hence the gain on performance over a global optimization approach: the set of detections to be processed is kept rather small and manageable, just like the simple case shown on Figure 2.3. In order to solve problem (2.1), we use the same strategy proposed by Wongun Choi, using efficient inference with a Conditional Random Field that evaluates many different groupings or combinations through the minimization of an energy function $E(\mathbf{A}^{*t-1}, \mathbf{H})$ that evaluates how good is a specific partitioning "$\mathbf{H}$" on $\mathbf{D}_{t-1}^t$ to be associated to the current set of clean detections $\mathbf{A}^{*t-1}$.

Here, we will present the steps needed prior to the inference itself: building a selection of partitions without testing every possible combination. Details behind the energy function and the inference process are deeply studied on Chapter 3.

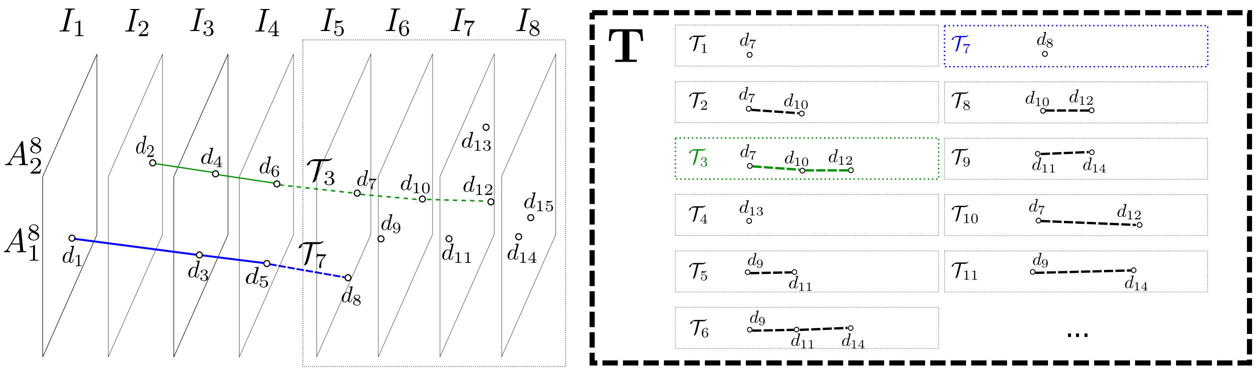

I_1 I_2 I_3 I_4 I_5 I_6 I_7 I_8
A_2^8
A_1^8
$\mathcal{T}_3$
$\mathcal{T}_7$
$\mathbf{T}$
$\mathcal{T}_1$ d_7
$\mathcal{T}_2$ d_7 d_{10}
$\mathcal{T}_3$ d_7 d_{10} d_{12}
$\mathcal{T}_4$ d_{13}
$\mathcal{T}_5$ d_9 d_{11}
$\mathcal{T}_6$ d_9 d_{11} d_{14}
$\mathcal{T}_7$ d_8
$\mathcal{T}_8$ d_{10} d_{12}
$\mathcal{T}_9$ d_{11} d_{14}
$\mathcal{T}_{10}$ d_7 d_{12}
$\mathcal{T}_{11}$ d_9 d_{14}
...

Figure 2.3: Visual depiction of the incoming detections $\mathbf{D}_1^8$ over a frame sequence with $t = 8$ and $\tau = 4$, and the tracking output. Here, we depict one association $A_1^8 = \{d_1, d_3, d_5, d_8\}$ in blue and another $A_2^t = \{d_2, d_4, d_6, d_7, d_{10}, d_{12}\}$ in green. Dotted lines connect loose detections, whilst solid color lines connect clean detections. Hence, $A_1^{*8} = \{d_1, d_3, d_5\}$ and $A_2^{*8} = \{d_2, d_4, d_6\}$, which are augmented by Tracklets $\mathcal{T}_7$ and $\mathcal{T}_3$, respectively. Note that the Tracklets shown here belong to the set $\mathbf{T}$ which is populated with many possible partial trajectories, which could belong to targets 1, 2 or to newer targets appearing during the interval $[t-\tau, t]$. Here tracklets without any common detections with existing trajectories, such as $\mathcal{T}_6$, could be good candidates for new targets.

2.2.1 Overview

Our NOMT Tracker performs the steps defined on Algorithm 2, which will be briefly described on what remains of this chapter. As input data, the algorithm receives:

- $\{\mathscr{I}_1^t, \mathscr{D}_1^t\}$: The pair of color and depth image sequences from which detections were extracted. Through 3D re-projection, we can form a point cloud that belongs to a single bounding-box for each detection. To simplify the notation, we omit these vectors representing the point clouds, as they are implicitly referenced by the set of detections.

- $\mathbf{D}_1^t$: The set of all N detections from time-stamp 1 to t. The total number of frames is also size($\mathbb{T}$), since we mentioned on Section 2.1.1 that we use the integer t as a reference to the most recent time-stamp.

- $\mathbf{A}^{t-1}$: The set of M existing associations created in the previous iteration. Each of them will contain an ordered list of detections associated to a unique target in $D_1^{t-\tau-1}$, along with the ordered list of detections that are bound to change their memberships as they lie in $D_{t-\tau}^t$. Note that M may vary from an iteration to the following one.

As output data, the algorithm gives the new set of associations $\mathbf{A}^t$. Data association needs to build a partitioning $\mathbf{H}$ on detections inside the sliding window and to make sure that the partitions do not share any detection, as it would not make sense to associate a detection with more than one target. Naively, we could build every possible partitioning and test each combination of detections, considering that they are fixed on their frames (ordering by time-stamp) and that partitioning is the same as assigning a label to each of them (with a unique target identifier: new or existing). Each time, we have M existing labels and N new detections and, in the worst case, all the N detections are considered not to belong to already observed targets and coming from separate trajectories. Then, we have a maximum of $N + M$ labels to assign to this set of N detections of fixed orderings. Since labels can be repeated between different frames but not within the same frame (two detections cannot belong to the same target at the same time-stamp), we would need to test a total of $\prod_{i=t-\tau}^{t} P(N+M, \text{size}(\mathbf{D}_i^i))$ cases, where this number is the product of permutations without repetition on a single frame, with $\mathbf{D}_i^i$ the number of detections in frame i.

This is a huge space: imagine a case with 2 detections in each frame, and a sliding window of size 10. This makes $N = 20$. Also suppose that we have 2 existing targets, hence $M = 2$, and the number of all possible partitions is exactly $10 \cdot P(22, 2) = \mathbf{4620}$. Yet, most of them are very unlikely. If we have a way to measure how similar each detection is from one to another in a different frame, we could favor partitions in which consecutive detections should be similar. This intuition leads us to find better ways to test fewer combinations for data association.

2.2.2 Tracklet Generation

The proposed way of partitioning detections in $\mathbf{D}_{t-\tau}^t$ is to build tracklets containing detections similar enough, up to a threshold, using a measure function $a(d_i, d_j)$ that indicates how compatible is a pair of RGB-D detections. This similarity measure is presented in depth on Chapter 4 and is designed to return a real value $r \in [-1, 1]$ where $r = 1$ when detections are very likely to correspond to the same objective. Then we test if resulting chains are all compatible enough with existing targets for them to be associated with, or are more likely new targets. This results in a tracklet set : a set of grouped loose detections that comply with the rules presented in Section 2.1.2.

In order to group detections in a Tracklet, we sample the configuration space of associations and pick the best in terms of overall similarity. To do this, the approach is to greedily build Tracklets as depicted on Algorithm 2, from lines 1 to 18, by picking frames $[t-\tau, t]$ one by one, in increasing order, and then, for each safe detection ($\forall d_i \in \mathbf{D}_{t-\tau}^t$ s.t. $s_i > 0$), it creates a new tracklet $\mathcal{T}_i = \{d_i\}$ and proceeds to find a detection d_k in the following frames through greedy maximization such that the index k maximizes an inner tracklet similarity:

$$k = \underset{\substack{k \,\in\, \text{timestamps}\{\mathbf{D}_{t-\tau}^t \setminus \mathcal{T}_i\} \\ t_k \,>\, t_j \,=\, \max(\text{timestamps}\{\mathcal{T}_i\})}}{\arg\max} a(d_j, d_k) \tag{2.2}$$

The algorithm goes on by repeatedly choosing the most similar detection up to a minimum threshold ρ of similarity, until frame t is reached. The selection of this threshold depends on how good and accurate is the similarity measurement: even though it is expected that all measurements in the range $(0, 1]$ are an indication of similarity, sometimes yields between detections of the same target end up not in this range[3], even to the point of getting better similarity against another target's detections. A bad choice of ρ makes this step prone to introduction of mistakes by not trimming correctly the configuration space of associations since this greedy approach relies only on similarity, but copes well enough for cases when detections of a particular target are not distributed consecutively over frames. In practice, we use $\rho = 0.1$ as it works well with our measuring strategy.

[3]due to the target's dynamic pose, occlusions and changing observable surface.

2.2.3 Non-Maximal Suppression

Most detection mechanisms are susceptible to finding the same target more than once in a given frame due to searches and different image scales used when performing object detection algorithms. Hence a technique called Non-Maximal Suppression (NMS) is performed to keep the best detection in terms of its score s_i when referring to a single target d_i. This process is simple: having a set of detections at frame t_i, it picks a detection with the highest score and tests how much overlap it has with every other detection. When the intersection area is big enough, this detection is discarded. This process is repeated by taking the second highest scored detection and so on until all redundant detections with lower scores are purged. Such process is assumed to have been applied on detections $\mathbf{D}_{t-1}^{t}$ when supplied to the Tracking pipeline.

This same premise applies to Tracklets generated using the previous greedy algorithm, as it tends to generate many Tracklets that are just smaller pieces of larger ones. A NMS on $\mathbf{T}$ is performed similarly to [38], using track bounding-cylinders of radius r to evaluate intersection areas instead.

Here we define the overlap between two single detections through Intersection over Union, a ratio defined as:

$$\mathbf{IoU}(d_i, d_j) = \frac{A_{int}(d_i, d_j)}{(\pi\ d_i[r]^2 + \pi\ d_j[r]^2 - A_{int}(d_i, d_j))} \tag{2.3}$$

where

$$A_{int}(d_i, d_j) = \begin{cases} 0 & \ell \geq d_i[r] + d_j[r] \\ \pi\ d_j[r]^2 & d_i[r] \geq d_j[r] \text{ and } \ell \leq d_i[r] - d_j[r] \\ \pi\ d_i[r]^2 & d_j[r] \geq d_i[r] \text{ and } \ell \leq d_j[r] - d_i[r] \\ \frac{1}{2}\Big[d_j[r]^2[\theta - \sin\theta] + d_i[r]^2[-\phi - \cos\phi]\Big] & \text{otherwise} \end{cases} \tag{2.4}$$

$$\phi = 2\arccos\left[\frac{\ell^2 + d_i[r]^2 - d_j[r]^2}{2\ell d_i[r]}\right] \qquad \theta = 2\arccos\left[\frac{\ell^2 + d_j[r]^2 - d_i[r]^2}{2\ell d_j[r]}\right]$$

and $\ell = \sqrt{(d_i[x] - d_j[x])^2 + (d_i[y] - d_j[y])^2}$ is the distance between the centers. All these equations are deduced from the geometry of two intersecting circles of different radius $d_i[r]$ and $d_j[r]$, with centers $< d_i[x], d_i[y] >$ and $< d_j[x], d_j[y] >$, respectively.

Since Tracklets are a collection of detections, we generalize **IoU** to Tracklets by evaluating:

$$\mathbf{overlap}(\mathcal{T}_i, \mathcal{T}_j) = \frac{\sum_{f=t-\tau}^{t} A_{int}(d(\mathcal{T}_i, f), d(\mathcal{T}_j, f))}{\sum_{f=t-\tau}^{t} \pi r_1^2 + \pi r_2^2 - A_{int}(d(\mathcal{T}_i, f), d(\mathcal{T}_j, f))}, \tag{2.5}$$

where $d(\mathcal{T}, f)$ indicates a Tracklet detection at time-stamp f, considering that such detection could not exist for shorter Tracklets and those with frame-skips. It reads as the total area of disk intersection divided by the total area, considering all detections in Tracklet $\mathcal{T}_i$ and $\mathcal{T}_j$. Intersection areas are then evaluated using equation (2.4).

To apply NMS on Tracklets, we still need to define a sorting other than a detection score. Here we consider the aforementioned similarity function $a(d_i, d_j)$ to find how good is the inner similarity in a Tracklet. Formally, the score is now:

$$\mathrm{s}(\mathcal{T}) = \sum_{f_k}^{F} a(d(\mathcal{T}, f_k), d(\mathcal{T}, f_{k+1})), \tag{2.6}$$

where F is the ordered set of time-stamp frame indices where $\mathcal{T}$ has a detection. Frame f_{k+1} indicates the next nearest detection inside $\mathcal{T}$ after frame f_k (up to $k+1 \leq \tau$), to allow frame-skips. This inner Tracklet similarity score favors longer tracklets. In the case of $\mathcal{T}$ having all its τ tracks and with all its contiguous detections perfectly similar, the result of (2.6) would be $\tau - 1$, as it is also the number of detection pairs that can be evaluated, and as the similarity score of $a(d_i, d_j) \in [-1, 1]$.

All being defined, NMS is summarized in Algorithm 1.

```
Algorithm 1 Tracklet Non-Maximal Suppression (NMS)
Input: $\mathbf{T}$ sorted by decreasing score using (2.6). $\nu$ overlap threshold.
Output: $\mathbf{T}'$
1: $\mathbf{T}' = \emptyset$
2: for all $\mathcal{T}_i$ in $\mathbf{T}$ do
3:     $\mathbf{T}' = \mathbf{T}' \cup \mathcal{T}_i$
4:     for all $\mathcal{T}_j$ in $\mathbf{T} \setminus \mathbf{T}'$ do
5:         if overlap$(\mathcal{T}_i, \mathcal{T}_j) > \nu$ then
6:             $\mathbf{T} = \mathbf{T} \setminus \mathcal{T}_j$
7: return $\mathbf{T}'$ purged of redundant Tracklets.
```

2.2.4 Hypotheses Generation

Tracklets provide plausible partitions since they keep a minimum similarity constraint on their inner detections but they do not exclude overlapping partitions i.e., $\bigcap_k T_k \neq \emptyset$. Even after Non-Maximal Suppression, some Tracklets could share one or more detections. Yet, by construction, we do obtain groupings for every detection in the sliding window i.e., $\bigcup_k T_k = \mathbf{D}_{t-\tau}^{t}$, meaning that there is a subset in $\mathbf{T}$ where partitions do not share detections. Still, we do not make any assertion on whether a Tracklet T_k belongs to an existing target's trajectory or is a new target entering the scene.

To tackle the association problem, we need a fast way to check whether a Tracklet is similar enough to existing targets. For each clean trajectory in $\mathbf{A}^{*t-1}$, the algorithm first generates candidate hypothetical trajectories through spatial prediction over those detections from frames $[1, t-\tau-1]$ using Linear Least-Squares Fitting (LLSQ) to find a polynomial with coefficients $\hat{\beta}$ that best fits the world's $\mathbb{XY}$ position of the detected targets[4]. Considering that trajectories in $\mathbf{A}^{*t-1}$ tend to be either linear or parabolic, we fit them using a quadratic polynomial and since a moving target tends to change its direction and speed, we only use $\kappa = 10, 20$ or 30 and detections in those last κ frames $\{d_i \in A_m^{*t-1} | t_i \in [t-\tau-\kappa, t-\tau]\}$).

[4]which is simpler than fitting 3D coordinates, as the $\mathbb{Z}$ coordinate tends to stay constant in the context of human tracking.

A 2-degree polynomial $y(x) = \beta_1 x^0 + \beta_2 x^1 + \beta_3 x^2$ can be estimated using those $\kappa = \text{size}(A_m^{*t-1})$ detections as input data. If we wanted to find coefficients β_m, such that the curve passed through all pairs $\{(d_i[x], d_i[y])\}_1^\kappa$, the following would need to be true

$$\mathbf{X}\beta_m = \mathbf{Y}, \tag{2.7}$$

where:

$$\mathbf{X}^T = \begin{bmatrix} 1 & d_1[x] & d_1[x]^2 \\ 1 & d_2[x] & d_2[x]^2 \\ \vdots & \vdots & \vdots \\ 1 & d_\kappa[x] & d_\kappa[x]^2 \end{bmatrix} \qquad \beta_m = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} d_1[y] \\ d_2[y] \\ \vdots \\ d_\kappa[y] \end{bmatrix},$$

which provides a system of κ equation with 3 unknowns, rendering the system over-determined, and not having a solution in general. LSQ attempts to find a coefficient vector $\hat{\beta}$ that minimizes $\|\mathbf{Y} - \mathbf{X}\beta_m\|^2$ in (2.7).

By defining a residual $\mathbf{r} = \mathbf{Y} - \mathbf{X}\beta_m$ that is to be minimized, the least squared error minimization is done on $\|\mathbf{r}\|_2 = \sqrt{\sum_{p=1}^{\kappa}(r_p)^2}$. With $r_p = \mathbf{Y}_p - \beta_1\mathbf{X}^0 - \beta_2\mathbf{X}^1 - \beta_3\mathbf{X}^2$, it turns $\mathbf{Y}_p$ into a linear combination of the columns in $\mathbf{X}$, and to minimize the norm of $\mathbf{r}$, an orthogonal projection of $\mathbf{Y}$ over the span$\{\mathbf{X}^0, \mathbf{X}^1, \mathbf{X}^2\}$ would be a solution. By conditioning the residual to $\mathbf{r} \perp \text{span}\{\mathbf{X}^0, \mathbf{X}^1, \mathbf{X}^2\}$, the same is expressed as $\mathbf{X}^T\mathbf{r} = 0$. By substituting the residual $\mathbf{r}$ with its expression, the fitting problem becomes:

$$\mathbf{X}^T(\mathbf{Y} - \mathbf{X}\hat{\beta}_m) = \mathbf{X}^T\mathbf{Y} - \mathbf{X}^T\mathbf{X}\hat{\beta}_m = 0.$$

Hence the best approximation of the optimal linear least squares coefficients for any set of points given by a target's trajectory is obtained by solving:

$$\hat{\beta}_m = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}. \tag{2.8}$$

Once $\hat{\beta}_m$ of A_m^{*t-1} is known, the full tracklet set $\mathbf{T}$ is tested against predicted positions to make a selection of candidates, creating in the process multiple target hypotheses[5] $\mathbf{H}_m^t = \{\emptyset, H_{m,2}^t, H_{m,3}^t, \dots\}$ where $\emptyset$ (empty hypothesis) represents the termination of target's trajectory and each $H_{m,k}^t$ indicates a candidate set of detections in $[t-\tau, t]$ that can be associated to a target. Every tracklet in $\mathbf{T}$ is considered as a possible new entering target, associated or not, so that the inference process is capable of deciding on whether a particular track is from a new target or from an existing target. For hypotheses on newly entering targets, we define their clean trajectory as an empty set $A_m^{*t-1} = \emptyset$ until the time window passes it, through augmentations.

Each $H_{m,k}^t$ may contain from 0 to τ detections (in one single frame, there can be 0 or 1 detection associated to target m). Given the set of hypotheses for all the existing and new targets, the algorithm finds the most consistent set of hypotheses for each target using a graphical model that will be described in the next chapter.

[5]Note that earlier we introduced $\mathbf{H}$ as an undefined partitioning over $\mathbf{D}_{t-\tau}^t$. Later we built many partitions through a selection of Tracklets, now referred to as hypotheses, which are in essence a better defined partitioning.

Algorithm 2 NOMT: Single iteration over sliding window $[t-\tau, t]$

```
Input: ${\mathscr{I}_1^t, \mathscr{D}_1^t}$, $\mathbf{D}_1^t$, $\mathbf{A}^{t-1}$, $\gamma = 0.4$, $\omega = 0.1$
Output: $\mathbf{A}^t$
 1: define clean targets $\mathbf{A}^{*t-1} = \{d_i \in \mathbf{A}^{t-1} \mid t_i \notin [t-\tau, t]\}$
 2: for all $A_m^{*t-1}$ in $\mathbf{A}^{*t-1}$ do
 3:     for all $d_i \in \mathbf{D}_{t-\tau}^t$, s.t. $s_i > 0$ do
 4:         Set $\mathcal{T}_i = \{d_i\}$
 5:         repeat
 6:             $s^* = 0$
 7:             $k^* = 0$
 8:             for all $d_k \in \mathbf{D}_{t-\tau}^t \backslash \mathcal{T}_i$ do
 9:                 for all $d_j \in \mathcal{T}_i$ do
10:                     $s = a(d_j, d_k)$
11:                     if $s > s^*$ then
12:                         $k^* = k$ and $s^* = s$
13:             $\mathcal{T}_i = \mathcal{T}_i \cup d_{k^*}$
14:         until $s^* < \gamma$ or size$(\mathcal{T}_i) = \tau$
15:         $\mathbf{T} = \mathbf{T} \cup \mathcal{T}_i$
16: Apply Non-Maximal Suppression on $\mathbf{T}$.
17: for all $\mathcal{T}_i$ in $\mathbf{T}$ do
18:     Create NEW target $m+1$ with $A_{m+1}^{*t-1} = \emptyset$
19:     $\mathbf{A}^{t-1} = \mathbf{A}^{t-1} \cup A_{m+1}^{*t-1}$
20: for all $A_m^{*t-1}$ in $\{\mathbf{A}^{*t-1} | A_i^{*t-1} \neq \emptyset\}$ do
21:     Initialize $\mathbf{H}_m^t$ with null hypothesis $\{\emptyset\}$
22:     Using LLSQ obtain $\hat{\beta}_m$ for $k$-degree polynomial fit and predict positions in $[t-\tau, t]$.
23:     for all $\mathcal{T}_i$ in $\mathbf{T}$ do
24:         if $\mathbf{IoU} > \omega$ between any predicted positions and those in $\mathcal{T}_i$ at any frame then
25:             Add $\mathbf{H}_m^t = \mathbf{H}_m^t \cup H_i^t$ where $H_i^t = \mathcal{T}_i$.
26: Solve $\hat{x} = \arg\min_x E(\mathbf{A}^{*t-1}, \mathbf{H}^t(x), \mathbf{D}_{t-\tau}^t, \{\mathscr{I}_1^t, \mathscr{D}_1^t\})$
27: for all $A_m^{*t-1}$ in $\mathbf{A}^{*t-1}$ do
28:     Augment $A_m^t \leftarrow A_m^{*t-1} \cup \mathbf{H}_m^t(\hat{x}_m)$
29:     if $A_m^t$ is NEW and did not augment i.e., $A_m^t = \emptyset$ then
30:         drop target $m$
31: return augmented clean targets as $\mathbf{A}^t$
```

2.2.5 Data Association and Target Augmentation

Having a set of clean trajectories $\mathbf{A}^{*t-1}$, the set of hypotheses $\mathbf{H}^t$ describing configurations of detections $\mathbf{D}_{t-\tau}^t$ within the time window with their corresponding RGB-D frames, the data association introduced in equation (2.1) is reformulated as the following energy minimization

$$\hat{x} = \arg\min_x \; E(\mathbf{A}^{*t-1}, \mathbf{H}^t(x), \mathbf{D}_{t-\tau}^t, \{\mathscr{I}_1^t, \mathscr{D}_1^t\}). \tag{2.9}$$

The variable x is an integer state vector indicating which hypothesis (i.e., which index in $\mathbf{H}_m^t$)

is chosen for each target m. $\mathbf{H}^t$ is the union set of all hypotheses sets per target $\{\mathbf{H}_1^t, \mathbf{H}_2^t, \dots\}$ and $\mathbf{H}^t(x)$ is a subset of selected hypothesis $\{H_{1,x_1}^t, H_{2,x_2}^t, \dots\}$ where x_m is the index of a particular hypothesis in $\mathbf{H}_m^t$.

After finding the solution $\hat{x}$, we get the set of selected hypothesis $\mathbf{H}^t(\hat{x})$ which provides an optimal configuration or partitioning on detections $\mathbf{D}_{t-\tau}^t$ that can be used to augment clean trajectories $\mathbf{A}^{*t-1}$. New trajectories are obtained from existing targets by connecting detections in the chosen hypothesis and previous clean trajectories[6]:

$$A_m^t = A_m^{*t-1} \cup \mathbf{H}_m^t(\hat{x}_m). \tag{2.10}$$

In practice, all the targets managed by the tracker keep a label indicating whether they are NEW, DELETED, OCCLUDED or just OBSERVED, which is taken into consideration when cleaning targets that were created during step 20 of Algorithm 2, as all Tracklets are treated as if they could belong to a newly entering target. If after the process of inference these temporary targets did not get augmented (still inside the sliding window) they are discarded as shown on step 30.

Details on the aforementioned energy minimization framework and its solution are now presented in Chapter 3 which dives into the process of efficient inference and how it fits our problem.

[6]Note that at the next iteration, depending on how many frames the window moved, some detections in $\mathbf{H}_m^t(\hat{x}_m)$ become clean tracks in A_m^t whilst the rest are required to be recalculated, hence, susceptible to change as new detections are available, causing possibly new partitioning in $\mathbf{D}_{t-\tau+\epsilon}^{t+\epsilon}$.

Chapter 3

Inference with CRF and Application to Tracking

The data association problem in Near-Online Multiple Object Tracking is handled as an inference problem in a Conditional Random Field (CRF). Hence, we will first do a brief review of concepts of probability and inference in graphical models. Readers familiar with this topic can skip the first sections of this chapter up to Section 3.7..

3.1 Inference over Graphical Models

Inference is the process of reasoning and making decisions by taking into account prior knowledge (evidence) and uncertainty inside a probabilistic model. A mechanism is required so that we can perform analysis over evidence and quantify our choices to reach an optimal solution for our interests.

3.1.1 Probability theory

The term probability is used to imply a certain degree of confidence that an event of uncertain nature could happen. When all possible outcomes for this event are known (hence finite) and if we can assign to them a probability, we refer to such event as a discrete random variable.

Capitalized letters are used in this document to denote random variables i.e. X, whilst lower case letters represent possible mutually exclusive outcomes or states of a variable i.e. $x_1, \ldots, x_n$. To indicate that a random variable has been observed, evaluated, or resulted in one of its possible states we write $X = x$. Since a random variable is not precisely a set of states, we will refer to the set of all its possible sates as $Val(X) = \{x_1, ..., x_n\}$.

A set of random variables will be represented by bold capital letters, i.e. $\mathbf{X} = \{X_1, \ldots, X_n\}$. Along with this, we introduce the concept of an assignment of $\mathbf{X}$ which is the same as indicating that every random variable in X resulted in a specific state $\{X_1 = x_1, \ldots, X_n = x_n\}$. For short, we use a bold lowercase letter $\mathbf{x} = \{x_1, \ldots, x_n\}$, hence used here as $\mathbf{X} = \mathbf{x}$. Again, we will reutilize function Val to refer to the set of all possible assignments: $Val(\mathbf{X}) = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ where it should be noted that $m = \prod_{X_i \in \mathbf{X}} \#Val(X_i)$ i.e. the product of all the number of possible states of each random variable in $\mathbf{X}$.

Formally, probability is defined as a function $P(X) : Val(X) \to \mathbb{R}^+$. Therefore, the probability of a state is represented by a positive real number $P(X = x) = c \in \mathbb{R}^+$, abbreviated as $P(x)$. When writing $P(X)$, we represent the distribution of X, which is the set of probabilities $P(X) = \{P(x_1), \dots, P(x_n)\} = \{c_1, \dots, c_n\}$ which must satisfy $\sum_i c_i = 1$. This constraint indicates the certainty we have that X will always take one of its possible states.

Summarizing, our certainty is "distributed" over each state, and this is the basis of most studies in a system of reasoning commonly known as theory of probability that considers uncertainty following this set of rules:

- If A is a random variable, $P(a)$ is a real number in the interval $[0, 1]$.
- If $a \in A$ is certain, then $P(a) = 1$.
- If A and B are mutually exclusive, then $P(A \text{ } or \text{ } B) = P(A) + P(B)$.

The last rule presents two different random variables whose probabilities are summed depending on the relationship between them.

When A and B are independent (i.e. $A \perp\!\!\!\perp B$), the outcome of A does not affect the probability of occurrence of any outcome of B. On the opposite case, when there is dependency, the evaluation is modelled with the help of conditional probabilities; which are expressed as $P(A = a \mid B = b) = x$ and can be read as: "given the evidence of B resulting in b and if any other information to handle is irrelevant to A, the probability of A resulting in a is x". This means that $P(A \mid B) = P(A)$ when $A \perp\!\!\!\perp B$, since no evidence or outcome of B will change the odds of the result of A.

When working with two random variables, we may also want to calculate their joint probabilities i.e., the measurement of the odds of A resulting in a and B resulting in b at the same time, denoted as $P(A = a, B = b)$. In the case of independent variables, since A and B share no conditional probabilities, the following equality holds:

$$P(A, B) = P(A) \; P(B). \tag{3.1}$$

For the evaluation in the non-independent case, we introduce the fundamental rule for probability calculus, which is the product rule:

$$P(A, B) = P(A \mid B) \; P(B), \tag{3.2}$$

which combines conditional probabilities for individual variables to define joint probabilities.

3.1.1.1 Bayes Theorem

It is easy to find that the joint probability of variables A and B is equal to:

$$P(A, B) = P(A \mid B) \; P(B) = P(B \mid A) \; P(A), \tag{3.3}$$

from which the Bayes Theorem can be easily deduced:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{3.4}$$

Typically, we are interested in A, and we begin with a prior knowledge $P(A)$ for our belief about A, and then we observe B. The Bayes theorem tells us that our revised belief for A, the posterior probability $P(A \mid B)$, is obtained by multiplying the prior $P(A)$ by the ratio $\frac{P(B \mid A)}{P(B)}$, which is a function of varying values of A for a fixed value for B, called likelihood of A. This relationship is sometimes expressed as $P(A|B) \propto P(A)P(B|A)$, by omitting the constant $P(B)$, which normalizes the resulting probability.

3.1.1.2 Chain Rule

The previous theorem is actually a special case of the chain rule or general product rule which allows the calculation of a joint distribution over a set $\mathbf{X} = \{X_1, \dots, X_n\}$ of random variables. It is expressed as:

$$P(X_1, \dots, X_n) = P(X_1)P(X_2 \mid X_1) \dots P(X_n \mid X_1, \dots, X_{n-1}). \tag{3.5}$$

The main challenge in the inference process is to evaluate efficiently the joint probability of a selection of values in a complex set of random variables. Evaluating this distribution of probabilities by programmatically applying the Bayes rule can quickly become intractable for many real-life applications, even more when response times are critical and when the calculus needs to be done instantly.

As we have seen, concepts introduced so far can be translated to larger sets of random variables and are the foundation for most techniques used to ease the computation of the probabilities.

3.1.2 Graphical Models

Graphical models provide an intuitive way to understand conditional probabilities and dependencies in a set of random variables. Graph theory provides a data structure called graph defined as $G = (V, E)$ where V is a set of vertices and E is a set of edges. A graphical model uses such a structure to encode the nature of the relationships among a set of variables $\mathbb{X}$ by associating a random variable from $\mathbb{X}$ to each vertex, and by drawing edges between them to indicate a dependence. Note that $\mathbb{X}$ is a special case of a set of random variables, as it represents the complete set of variables of a graphical model. Often, most definitions use $\mathbf{X}$ which is assumed to be $\mathbf{X} \subseteq \mathbb{X}$ unless stated otherwise.

More properties are given to graphs allowing for many useful contraptions, such as the directed graph (DG), which has its edges associated to a direction. The direction is implied by the order of vertices in the edge i.e. $e_a = (v_i, v_j)$, meaning that it begins on v_i and ends on v_j. With this definition, the edges e_a and $e_b = (v_j, v_i)$ are distinct and both valid.

DGs give a notion of cause-effect between random variables, as their edges and orientations represent influence (although not always of a causal nature). Take edge an $e = (v_i, v_j)$ for instance, then v_i is the parent of v_j, and v_j is the child of v_i. By grouping all adjacent edges for a vertex v, its parent set is denoted by $pa(v)$ and the children set is denoted by $ch(v)$. A vertex is then generated by its parents, as any evidence or observation made in any of those random variables in its parent set will change the odds of the child vertex. Any other vertex that is not a parent in any level of depth will thereby be independent from v, as dictated by the model. Graphical models are built by design, to represent such relationships.

3.1.3 Bayesian Networks

Bayesian networks are instances of graphical models that facilitate the calculation of conditional probabilities, giving a way to compute them over variables of interest, given data or evidence at hand. They provide:

- a data structure that represents the joint distribution of a variable set in a compact way and factorized in terms of conditional and prior probabilities,
- a compact representation for a set of conditional independence assumptions about a distribution.

A Bayesian network is a Directed Acyclic Graph (DAG) whose structure defines a set of conditional independence properties. To each vertex is associated a conditional probability distribution over a single random variable, the conditioning being done on the parents of the node in the graph, $P(X|pa(X))$. The joint density over the set of all variables $\mathbb{X}$ in the graph is then given by the product of such terms over all nodes:

$$P(\mathbb{X}) = \prod_{X_i \in \mathbb{X}} P(X_i|pa(X_i)). \tag{3.6}$$

This is called a recursive factorization, simplifying the joint distribution calculation by applying independence assumptions depicted according to the DAG. At this point, part of the process of inference must take into account the data observed on some of the random variables, if any. This involves calculating marginal probabilities conditioned on the observed data, using the Bayes theorem, which is diagrammatically equivalent to reversing one or more of the Bayesian network arrows.

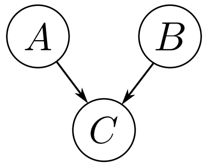


Figure 3.1: On this simple Bayesian network with random variables A, B and C, the relationship modeled here is the dependence between C to its parents A and B, whilst these two are independent of each other. By the chain rule, we could calculate the joint distribution as $P(A, B, C) = P(A)\ P(B \mid A)\ P(C \mid A, B)$. Since A and B are independent, it becomes $P(A, B, C) = P(A)\ P(B)\ P(C \mid A, B)$, which is also the result of applying formula 3.6 on the graph.

Many real-world problems can be represented with Bayesian networks, as there is an intuition of the relationship between entities, and as the outcomes of these may affect the outcomes of other components, meaning that the models can be generative and also have independent variables. However, sometimes, real-world relationships do not behave this way, they are less parent-child-like, meaning that the dependence between two random variables is bi-directional: observing either of the outcomes may affect the outcome of the other. This situation is impossible to represent with a Bayesian network, and requires for a different kind of graphical model.

3.2 Markov Networks

As Bayesian Networks are represented by Directed Graphs, Markov Networks are represented by Undirected Graphs. Many problems involve entities that do not influence directly others, as sometimes a change on either side might results in an alteration of the other. Markov Network models are useful in describing these interactions between variables.

Nodes in a Markov Network also represent random variables, and edges indicate now a notion of direct interaction between neighboring variables that is not mediated by any other variable in the network. This means that one cannot represent the distribution over one node given others. Rather, we use a more symmetric parametrization by representing the affinities between related variables.

With this in mind, we introduce the concept of factor, a.k.a potential, which is a function $\phi(\mathbf{U}) : Val(\mathbf{U}) \to \mathbb{R}^+$ where $\mathbf{U} \subset \mathbb{X}$ is a subset of variables which are being affected by each other, called the scope of ϕ. The value associated to a particular assignment of $\mathbf{U}$ denotes an affinity: the higher the value, the higher the compatibility of the single assignments embedded in $\mathbf{u}$.

When designing the Markov Network, we create a set of factors $\mathbf{\Phi} = \{\phi_1, \ldots, \phi_f\}$ to explain all the relationships between variables in $\mathbb{X}$. Each factor works with its corresponding scope: $Scope[\phi_i] = \mathbf{U}_i$. The number of subsets and factors f depends on the modeled dependencies, hence one can play with the scopes of factors and still obtain the same distribution, but with a different modelling. Some models may be easier to parametrize than others and this is a problem that will be addressed later in this section.

After establishing a way to measure dependent variables, we need to join their local interactions into a global model. Just like in Bayesian Networks, factors are multiplied to combine their contributions and estimate a probability distribution. Formally, the factor product is defined as:

Definition 3.2.1 Let $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ be three disjoint sets of variables, and let $\phi_i(\mathbf{X}, \mathbf{Y})$ and $\phi_j(\mathbf{Y}, \mathbf{Z})$ be two factors. The product of $\phi_i \times \phi_j$ results in a new factor $\psi : Val(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) \to \mathbb{R}$,

$$\psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) = \phi_i(\mathbf{X}, \mathbf{Y}) \times \phi_j(\mathbf{Y}, \mathbf{Z}). \tag{3.7}$$

So it can be derived that the full joint distribution is the product of all factors over the subsets:

$$\tilde{P}_{\mathbf{\Phi}}(\mathbb{X}) = \prod_{\phi_i \in \mathbf{\Phi}} \phi_i(\mathbf{U_i}). \tag{3.8}$$

The function $\tilde{P}_{\mathbf{\Phi}}$ gives values that are not necessarily normalized, as it is not a probability[1], and just weights all possible selections of the variables involved. This means that we could obtain a constant Z known as the partition function to define a legal distribution by normalizing $\tilde{P}_{\mathbf{\Phi}}$

$$Z = \sum_{\mathbf{x} \in Val(\mathbb{X})} \tilde{P}_{\mathbf{\Phi}}(\mathbf{x}). \tag{3.9}$$

This finally gives the joint distribution of the Markov Network as:

$$P_{\mathbf{\Phi}}(\mathbb{X}) = \frac{1}{Z} \tilde{P}_{\mathbf{\Phi}}(\mathbb{X}), \tag{3.10}$$

where $P_{\mathbf{\Phi}}(\mathbb{X})$ is a probability distribution, hence satisfying that the sum of all probabilities must be 1. Then again, being a "legal" distribution, this parametrization is not as intuitive as that of Bayesian Networks, since factors do not correspond to either probabilities or conditional probabilities. Parameters are not intuitively understandable, since the representation is undirected by nature and the influences of variables over others might not be symmetrical. However, thanks to factors, we achieve the flexibility to have both the notion of a distribution and a conditional probability distribution.

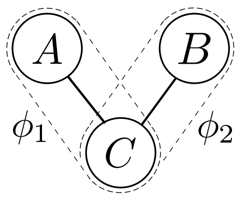


Figure 3.2: This simple Markov network with variables A, B and C presents two undirected edges between C and its parents. To model their interactions we would define factors $\phi_1(A, C)$ and $\phi_2(B, C)$. To measure the potential or affinity of a particular selection in this set we multiply the factors: $\tilde{P}_{\Phi}(a, b, c) = \phi_1(a, c)\ \phi_2(b, c)$. This way, the distribution of $\tilde{P}_{\mathbf{\Phi}}(A, B, C)$ complies with equation 3.8.

When defining the factors, we mentioned the set of variables $\mathbf{U}$ as a set of random variables from which we want to model their relationships, as they are a subgraph in the Markov Network. We would connect the variables in $\mathbf{U}$ with the corresponding undirected edges. Any edge implies the need of a factor. Consider Figure 3.2 where two factors where created. They both share variable C but if A and B had a direct relationship (i.e. connected) we could have defined yet another factor. The distribution could have been $\Phi(A, B, C) = \phi_1(A, C)\phi_2(B, C)\phi_3(A, B)$ in that case.

Remember that we are multiplying the factors to join their contributions, hence joining the common parts. In this case, the contributions of A, B and C. This is actually a special case of pairwise Markov network where factors are over single variables or pairs of variables only.

[1]Since we are no longer working with such random variables in which the outcome is uncontrollable, rather we don't fully understand the effects of selections in a complex model. Otherwise we could sample a large number of times and get a result with some frequency to obtain a statistical probability, but now the focus is more about measuring the affinity of a selection as if it was a probability.

3.2.1 Parametrization and factors

We defined a factor as a function $\phi(U) : Val(U) \rightarrow \mathbb{R}^{+}$. To define a factor, we need to assign a real value to each element of $Val(U)$. How many parameters are needed? It equals the product of how many choices each random variable in U has, i.e. $m = \prod_{X_i \in \mathrm{U}} \#Val(X_i)$, as presented at the beginning of this chapter.

When defining factors, the selection of their scopes is important. Take this example: If A, B and C were random variables, with connections $A - B - C$ but none between A and C, each with 3 possible states. We could come up with one factor $\phi(A, B, C)$ that would have $3 * 3 * 3 = 27$ parameters to be evaluated, whilst if we had only two factors, $\phi(A, B)$ and $\phi(B, C)$ then it would have only $(3 * 3) + (3 * 3) = 18$ parameters to assign. In this case, if A and C had been connected, any choice of factors and scopes would need 27 parameters to be assigned.

Without losing generality, we could reduce the number of factors by defining clique potentials. A clique C is a complete subgraph[2] found in the graphical model over $\mathbb{X}$. This can be interpreted as a group of fully related random variables, and a single factor could be designed to consider the affinity of all the assignments in C and would require heavier parametrization for every possible assignment. Since in practice most cliques end up being of a very small scope due to the full connectedness requirement, the size of m ends up smaller, which is what happened on the scenario $A - C - B$ where the only valid cliques are $A - C$ and $C - B$.

If we associate a factor only to a clique, we are not violating the independence assumptions introduced by the network structure. So we get a reduced number of evaluations in our parametrization if we only allow factors for maximal cliques[3].

In the context of computer vision, Markov networks are typically referred to as Markov Random Fields (MRF) and are widely used for many visual processing tasks such as image segmentation, removal of blur and noise, stereo reconstruction, object recognition and many more.

3.2.2 Factor Graphs

A Markov Network structure might not reveal all the information on the nature of some dependencies, as it can actually be used to emulate those relationships as depicted on a Bayes Network. One cannot tell from the graph structure whether the factors in the parametrization involve maximal cliques or subsets. Though Markov Networks are not redundant in terms of conditional independences, since no matter how small are factors scopes, no additional independences are implied in the distribution; yet this grouping is significant on the number of parameters needed.

A factor graph $\mathscr{F}$ provides an alternative and richer representation that makes the structure of factors in the network explicit. $\mathscr{F}$ is an undirected graph with two types of nodes: one type is the usual random variables, denoted by ovals, and the other corresponds to factors over the variables, denoted by squares. There are only edges between variable nodes and factor nodes. Thus $\mathscr{F}$ is parametrized by a set of factors, where each factor node V_ϕ is associated with precisely one factor

[2]i.e. all nodes are connected directly to all the other nodes, meaning the subgraph is fully connected.

[3]cliques that can't be augmented by including another adjacent vertex, hence the name maximal.

ϕ, whose scope is the set of variables neighbors of V_ϕ in the graph. A distribution P factorizes over $\mathscr{F}$ if it can be represented as a set of factors of this form.

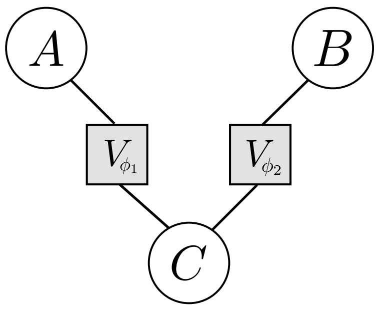


Figure 3.3: A factor graph for $A - C - B$ where factors are explicitly drawn as square nodes V_{ϕ_1} and V_{ϕ_2} This representation of a Markov Network is very useful when working on a programming language: factor nodes can be associated with its neighboring variable vertices and probability tables with evaluations on all possible assignments, encoding everything that conforms the model in a single structure.

Factor graphs might not change the independence assertions made by the model, but are very useful in the context of inference, both for hand-coded models and learning.

3.3 Variable Elimination

Up to this point, we have introduced the main tools used for tackling large sets of random variables and their relationships in a model. Each graphical tool we have described permits for the representation of different types of dependencies in their variables and gives a way to calculate the joint density. Yet, typically we require to perform inference over a set of random variables $\mathbb{X}$ given the joint density formula. We want to make a query for inferring the distribution of a subset $\mathbf{X}_Q \subset \mathbb{X}$, called the query variables, whilst considering an observed assignment: the evidence $\mathbf{e}$ from a subset of variables $\mathbf{X}_E \subset \mathbb{X}$, called the evidence assignment. Thus the objective is to evaluate the following distribution, called conditional probability query:

$$P(\mathbf{X}_Q|\mathbf{X_E} = \mathbf{e}) = \frac{P(\mathbf{X}_Q,\ \mathbf{e})}{P(\mathbf{e})}. \tag{3.11}$$

Focusing on the numerator $P(\mathbf{X}_Q,\ \mathbf{e})$, remember that $\mathbf{e}$ is a fixed assignment of outcomes in the set $\mathbf{X_E}$. Then we need to evaluate all possible assignments of the query variables i.e. $Val(\mathbf{X}_Q) = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$, which can be computed by summing out all entries for the joint distribution that correspond to assignments consistent with $\mathbf{q}$ and $\mathbf{e}$. Let $\mathbf{X}_Y = \mathbb{X} - \mathbf{X}_Q - \mathbf{X}_E$ be the remaining random variables, that are neither query nor evidence. Then for each $\mathbf{q}$ (a possible value for $\mathbf{X}_Q$),

$$P(\mathbf{q}, \mathbf{e}) = \sum_{\mathbf{y} \in \mathbf{X}_Y} P(\mathbf{q}, \mathbf{e}, \mathbf{y}). \tag{3.12}$$

Note that the 3-tuple $\mathbf{q}$, $\mathbf{e}$, $\mathbf{y}$ is actually a set of assignments in the full set of random variables and so it is an entry in the joint distribution. $P(\mathbf{e})$ could be evaluated likewise afterwards, to save up

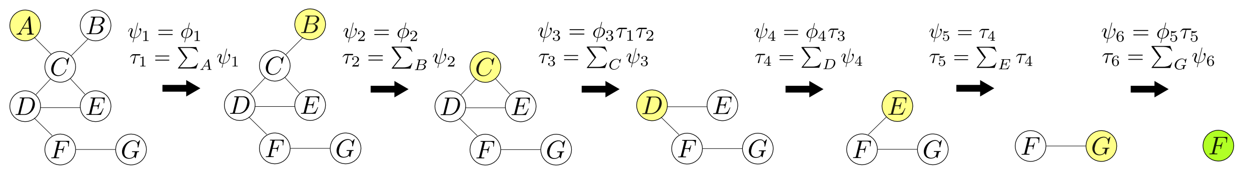


Figure 3.4: Consider an undirected graph containing nodes A to G and a set of factors: $\Phi = \{\phi_1(A,C), \phi_2(B,C), \phi_3(D,C,E), \phi_4(D,F), \phi_5(F,G)\}$. If we queried for the distribution of F, i.e. $P(F)$, we could choose to eliminate variables in the order A, B, C, D, E, G by applying the formulas (3.14) and (3.15) repeatedly and normalizing the final factor $\psi_7(F) = \tau_5(F) \cdot \tau_6(F)$.

some calculations as

$$P(\mathbf{e}) = \sum_{\mathbf{q}} P(\mathbf{q}, \mathbf{e}). \tag{3.13}$$

This way, equation 3.11 is calculable. In the end, we search for the most likely assignment and that is the output of inference by variable elimination, called this way since the effect of summation that removes first the set of assignments of the variables in $\mathbf{X}_Y$ and later, by Bayes rule, the variables in $\mathbf{X}_E$.

Note that this translates likewise to factors on Markov Networks. Remember that the set of factors Φ over a set of variables $\mathbb{X}$ are calculated as in equation (3.10) to build the joint distribution, and normalized to give the notion of a legal distribution. We name the variable elimination as factor marginalization. It is formally defined as:

Definition 3.3.1 Let $\mathbf{X}$ be a set of variables, and $Y \notin \mathbf{X}$ a single variable to be eliminated. First we compile every factor where the variable Y belongs to their scope. This is a set $\mathbf{\Phi'} = \{\phi \mid Y \in Scope[\phi]\}$. Define:

$$\psi(\mathbf{X}, Y) = \prod_{\phi \in \mathbf{\Phi'}} \phi(\mathbf{X}, Y). \tag{3.14}$$

Then $\psi(\mathbf{X}, Y)$ is a new factor (also called potential) where Y has been completely compiled. Factor marginalization of Y in ψ, denoted $\sum_Y \psi$, then becomes a factor τ over $\mathbf{X}$ only, such that:

$$\tau(\mathbf{X}) = \sum_Y \psi(\mathbf{X}, Y). \tag{3.15}$$

Variable elimination works by performing factor manipulation directly. Each step of the process creates a potential ψ by multiplying existing factors, and then marginalizing it to create a new factor τ in which a variable has been "eliminated". This process, though not studied in depth here, is illustrated in Fig. 3.4 and is the basis for designing better inference algorithms.

The problem of selecting the best order to perform variable elimination is *NP*-hard. Also, if we should change the query variable set to ask for something different, we shall re-evaluate products and summations. To make this more affordable, dynamic programming is possible to introduce thanks to the fact that many evaluations appear repeatedly and can be stored in memory to reduce the computational burden.

3.4 Efficient inference

Variable elimination requires to be re-computed entirely whenever there is a different query, and the ordering in which we go eliminating variables also changes the effort needed. Each time we eliminate successfully a variable, a factor τ_i is generated. So this gives the intuition of a "message" being passed for another elimination that creates a new ψ_j and then another τ_j.

Junction Tree Algorithms are a generalization of the Variable Elimination technique that avoids the multiple computations problem by compiling the density into a data structure that is capable of supporting many types of queries. To reach such a convenient graphical model, we study the evolution of graphs used in the inference process.

3.4.1 Cluster graph

Cluster graphs $\mathscr{U}$ are undirected graphs that allow the visualization of the process of factor-manipulation, just like a flowchart. It works with a set of factors $\mathbf{\Phi}$ over the set of random variables $\mathbb{X}$. Each node in $\mathscr{U}$ is associated with a subset $\mathbf{C}_i \subseteq \mathbb{X}$ called cluster[4], which are connected when their intersections is non-empty, much like factor graphs of section 3.2.2. An illustration is given in Fig. 3.5.

The key difference relies in the association of each factor $\phi \in \mathbf{\Phi}$ to a cluster $\mathbf{C}_i$, expressed as $\alpha(\phi) : \mathbf{\Phi} \to \mathbf{C}$ in which a factor works, such that $Scope[\phi] \subseteq \mathbf{C}_i$. As we said, each undirected edge between a pair of clusters $\mathbf{C}_i$ and $\mathbf{C}_j$ is associated with a shared set of variables called separate set or sepset $S_{i,j} \subseteq \mathbf{C}_i \cap \mathbf{C}_j$. This expression $S_{i,j}$ is interpreted as the set of variables that are passed from cluster i to cluster j, giving the sense of direction.

The usefulness of this contraption arises when performing variable elimination, as the whole execution can be described with a cluster graph: We have a cluster for each factor ψ_i that is created in the computation of equation (3.14). Then ψ_i is associated with the set of variables $\mathbf{C}_i = Scope[\psi_i]$. We have an edge between $\mathbf{C}_i$ and $\mathbf{C}_j$ if the marginalized variable message τ_i from ψ_i is used in the computation of τ_j.

In summary, for a determined elimination order, it depicts the order in which the contributions of variables are compiled until reaching the distribution of query variables.

3.4.2 Junction Tree

When variable elimination is executed and a cluster graph is obtained as a consequence, one must note that each intermediate factor τ_i is used at most once to create ψ_j, and so the factor ϕ_i used when marginalizing a variable is left aside and never used again for any calculation. This means that a factor generates another, giving a sense of directionality.

Even if the cluster graph $\mathscr{U}$ is defined as undirected, the process of variable elimination that it represents behaves necessarily as a tree, as it shows the passing of messages τ_i up until a last

[4]Not to be confused with a single random variable C or a clique C. As it is a set, we use a bold $\mathbf{C}$, but with its items being associated to one or more factors we give it the name of cluster.

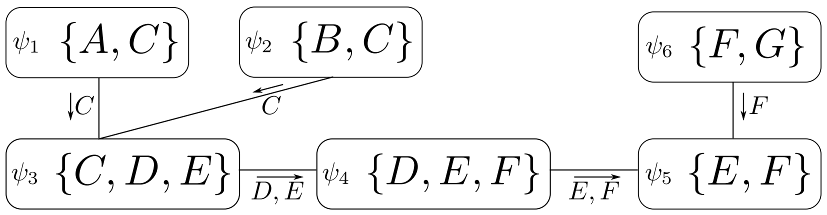


Figure 3.5: The cluster graph induced by the process of variable elimination presented in the previous section. Nodes contain the clusters of scopes of each ψ_i generated and edges present the sepset that is passed as a message for the next operation, meaning that the edges contain the scopes of τ_i. The graph will change its structure when choosing a different order of elimination.

node where the final result is obtained, which we can call the root of such tree. We can conclude that variable elimination also induces a junction tree with a cluster graph as a basis. Although not demonstrated here, it can be studied in depth in [15].

We introduce junction trees, which are able of caching computations, allowing multiple executions of variable elimination to be performed much more efficiently than by simply performing each one separately. Formally:

Definition 3.4.1 A junction tree $\mathscr{T} = \{V_{\mathscr{T}}, E_{\mathscr{T}}\}$, sometimes called join tree or cluster tree, is a tree over the set of factors $\mathbf{\Phi}$ over $\mathbb{X}$ composed of a set of vertices and edges that satisfy the running intersection property, which implies that whenever there is a variable X such that $X \in \mathbf{C}_i$ and $X \in \mathbf{C}_j$, then X is also present in every cluster of the (unique) path in $\mathscr{T}$ between $\mathbf{C}_i$ and $\mathbf{C}_j$.

This property holds in general for any cluster tree derived from variable elimination, since any variable that is not in the query variable set $X \notin \mathbf{X}_Q$ appears in every factor from the moment it is introduced when compiling for ψ_i until it is summed out. The query variables remain part of the message τ_j generated, as their scope is precisely $\mathbf{C}_i \cap \mathbf{C}_j$.

3.4.3 Clique Tree

Up to this moment, the junction tree clusters do not involve any restriction on the factors associated to them. When a junction tree involves operations over clusters that are cliques only, it is called clique tree. We have introduced the concept of clique in section 3.2. We reach this data structure to obtain a solid tool for the process of variable elimination, and a model that maintains a shorter set of factors $\mathbf{\Phi}$ by keeping maximal cliques only and a smaller footprint on parametrization. The following analysis on the steps of variable elimination over clique trees is also true for junction trees in general, but we will drop the term cluster in favor of clique.

After this long exploration on the tools needed to perform inference, we are ready to make the following summary:

If $\mathrm{C}_1, \ldots, \mathrm{C}_k$ are the cliques in a Markov Network, we can parametrize its corresponding joint distribution $P_{\mathbf{\Phi}}$ using a set of clique potentials $\psi_1(\mathrm{C}_1), \ldots, \psi_k(\mathrm{C}_k)$ and do inference using the variable elimination process and model it using a clique tree.

Now, we proceed to describe the steps involved in the basic usage of a clique tree.

3.4.3.1 Initialization

In general, any junction tree whose factors do not scope a whole clique can be converted into this form simply by assigning each factor to a clique ($a(\phi) \rightarrow \mathrm{C}$) that contains its scope $Scope[\phi] \subseteq \mathrm{C}_{a(\phi)}$ and multiplying all of the factors assigned to each clique to produce a clique potential (also referred to as initial potential)

$$\psi_i(\mathrm{C}_i) = \prod_{\phi \,:\, a(\phi)=i} \phi. \tag{3.16}$$

Note that this is formula achieves the same as equation (3.14), by concentrating clusters in the clique C_i, which has variables $\mathbf{X}_Y$ that will be eliminated by applying (3.15).

$$\tau_j(\mathrm{C}_i \backslash \mathbf{X}_Y) = \sum_{\mathbf{X}_Y} \psi_i(C_i). \tag{3.17}$$

Hence this initializes our first sepset as $S_{i,j} = \mathrm{C}_i \backslash \mathbf{X}_Y$ for leaf nodes.

3.4.3.2 Message Passing

We create a clique potential ψ to isolate the contributions of a certain variable for its elimination, resulting in a message τ. If a particular clique C' requires a message from a neighbor C, it must wait until the computation of C is finished. On a clique tree, messages are passed through all clique nodes, until reaching a root node that contains query variables. Messages move through the tree, and since the root is the last element, it needs the messages from its neighbors to be ready.

Definition 3.4.2 Let $\mathscr{T}$ be a clique tree. We say that C_i is ready to transmit to a neighbor C_j when C_i has messages from all its neighbors except from C_j.

The same happens for its neighbors as they too need other messages from their neighboring nodes, excluding the one that asked first. We note that the first messages to be actually sent are those from leaf nodes in the tree.

We define the following concepts:

- The root clique is referred to as C_r, where r is the index of such clique, and it is included in the set of all cliques.
- The neighbors of any clique C_i are represented as $nb[i]$ and they form the set of all indices that have an edge with C_i.
- Any $\mathrm{C}_{i|i\neq r}$ necessarily shares a unique path to the root element (thanks to definition 3.4.1) and that particular neighbor that sends its message to the root is called the upstream neighbor of i, represented as $p_r[i]$. Note that when any clique node C_i has received all the incoming messages, it can send its message to $\mathrm{C}_{p_r[i]}$, until the root is reached.

First we have the tree initialized by running equations (3.16) and (3.17) to create the first clique potentials on leaf cliques.

The passing of messages from a node to another can be represented with a recursive formula. We introduce the following function:

$$\delta_{i \to j} = \sum_{\mathrm{C}_i \backslash S_{i,j}} \left[\psi_i \cdot \prod_{k \in (nb[i] \backslash \{j\})} \delta_{k \to i} \right], \tag{3.18}$$

where $\delta_{i,j}$ is the sum-product message passing, that reads as "the marginal contribution sent from cliques i to j". It compiles all the messages received from neighboring ready cliques[5]. All contributions are multiplied just like we did before with Bayesian Networks and Markov Networks. Then all variables are summed out with the exception of those in the sepset

$$\tau(S_{i,j}) = \sum_{C_i \backslash S_{i,j}} \psi_i(C_i). \tag{3.19}$$

This gives a marginal clique potential $\tau(S_{i,j})$ where every sepset that is passed to a non-leaf node is $S_{i,j} = \mathrm{C}_i \cap \mathrm{C}_j$, advancing messages through the cluster tree until the root node is reached.

All contributions are combined with the initial potential at the root $\psi_r(\mathrm{C}_r)$. So to begin all calculations in the clique tree, we run the following equation until C_r is ready by evaluating $\delta_{i \to pr[i]}(\mathbf{S}_{i,pr[i]})$ on all ready cliques[6], recursively until reaching the root.

The result of this is a last potential on the root called belief and denoted as:

$$\beta_r(\mathrm{C}_r) = \psi_r \cdot \prod_{k \in nb[\mathrm{C}_r]} \delta_{k \to r}. \tag{3.20}$$

Note that this is the same as the unnormalized distribution of the root clique $\tilde{P}_{\mathbf{\Phi}}(\mathrm{C}_r) = \sum_{\mathbb{X} - \mathrm{C}_r} \prod_{\phi} \phi$, which contains our query variables.

3.4.3.3 Calibration

To perform inference on a clique tree is to perform a query, consider evidence, choose a root node and calculate its belief using equation (3.20). We mentioned that clique trees have the convenience of performing multiple queries, so if we evaluate belief over all nodes in the clique tree, we can achieve this goal.

This task has many redundant calculations, as we mentioned in section 3.3 on the variable elimination process. The reason is simple: Consider two neighboring cliques C_i and C_j. When evaluating the belief on any other node[7] of the clique tree, the passing of messages can only occur either on C_i side or on C_j side. Messages sent from a particular direction will always be the same,

[5]with the exception of leaf nodes, where the recursion ends by returning $\delta_{i \to j} = \tau_j(\mathrm{C}_i - \mathbf{X}_Y)$

[6]All leaf nodes are always ready to pass their initial potential.

[7]including i and j.

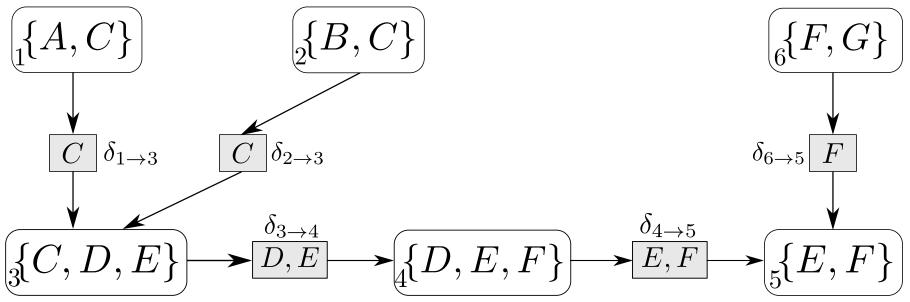


Figure 3.6: The junction tree formed with previous cluster graph has nodes for the corresponding sepset, storing generated messages on both directions to allow for many different queries. Note this tree has 6 cliques, meaning there are $2(6-1) = 10$ possible messages, which only 5 of them are shown as those are the ones used to obtain query variable F. If we needed to query for G, we would use the same set of messages except for $\delta_{6\to5}$, using $\delta_{5\to6}$ instead, which would have been computed already.

just like the sepset doesn't change. Thus, for any clique tree, each edge has two messages associated: $\delta_{i\to j}$ and $\delta_{j\to i}$.

The advantage of the clique tree is that all posterior probabilities are calculated for all variables in the graphical model using only twice the computation of the upward pass in the same tree. If the clique tree has c cliques, there are $c-1$ edges in the tree, and therefore $2(c-1)$ messages to compute.

When we are done calculating all messages, the clique tree is said to be calibrated. As a consequence, the following definition holds:

Definition 3.4.3 Two adjacent cliques C_i and C_j are said to be calibrated if

$$\sum_{\mathrm{C}_i \setminus \mathbf{S}_{i,j}} \beta_i(\mathrm{C}_i) = \sum_{\mathrm{C}_i \setminus \mathbf{S}_{i,j}} \beta_j(\mathrm{C}_j) := \mu_{i,j}(\mathbf{S}_{i,j}). \tag{3.21}$$

A clique tree $\mathscr{T}$ is calibrated if all pairs of adjacent cliques are calibrated. We use the term clique belief for the term $\mu_{i,j}(\mathbf{S}_{i,j})$.

3.4.4 Junction Tree Algorithm

As we have seen all the basic elements involved in the nature of the clique tree, we present one algorithm to summarize all the steps for performing inference.

Algorithm 3 begins with the initialization of each clique in the tree, obtaining clique potentials. Then it propagates all the messages between pairs of clusters in both directions. Note that the loop is finite as leaf nodes are always ready to send their message, which consequently makes their neighbors ready, hence evaluating $2(c-1)$ messages $\delta_{i,j}$. Finally, all messages are used to build clique beliefs for every node in the tree $\mathscr{T}$, which makes it calibrated.

Algorithm 3 Upward pass of variable elimination in clique tree

```
Input: $\mathbf{\Phi}$, $\mathscr{T}$, $a$, $\mathrm{C}_r$
Output: $\beta_r$
1: for all $\mathrm{C}_i \in \mathscr{T}$ do
2:     Calculate initial potentials using (3.16)
3: while there exists $i, j$ such that $i$ is ready to transmit to $j$ do
4:     Evaluate $\delta_{i,j}(\mathbf{S}_{i,j})$ using (3.18).
5: for all clique $\mathrm{C}_i$ do
6:     Calculate belief $\beta_i$ using (3.20).
7: return Set of clique beliefs $\mu_{i,j}(\mathbf{S}_{i,j})$ that satisfy equation (3.21)
```

Better and more efficient algorithms exist for the evaluation of a junction trees. Though not studied here, they are designed considering all the basic theory that we have presented on probability and graphical models to perform inference.

We can finally go back to the problem of Tracking, which is modeled and solved with these tools.

3.5 Building the Junction Tree

We have been working with clique trees by taking for granted that this structure was obtained through variable elimination. But now we will study a method called the Hugin Algorithm, that performs the following steps to turn a graphical model, be it a Bayesian or Markovian Network, into a clique tree.

Given a graphical model $\mathscr{H}$, with a set of modeled dependencies (edges) and random variables (nodes), we do the following:

- Moralization. We only apply this step when $\mathscr{H}$ is a Bayesian Network (or a DAG). First we add undirected edges to all co-parents (nodes sharing a common child) which are not currently joined, thus marrying them. Then we drop all directions in the graph obtained. The resulting graph is called a moral graph. The moral graph is undirected, and the addition of extra edges in the moralization (and triangulation in the following step) processes makes it impossible to keep all of the conditional independences of the original network.

- Triangulation. Once we have a Markov Network (undirected graph), we want to build a chordal graph $\mathscr{H}^*$ through triangulation. Here is why.

  Whilst the moralization of a graph is unique, triangulation might result in many alternative graphs. For optimal computation, we would like to have a graph where the largest clique has a minimal size to facilitate its parametrization, as the state space will be smaller. Finding such triangulation is not easy and is also an NP-hard problem. In practice, we use heuristic algorithms, because available exact methods have a computational cost exponential in the size of the largest clique in the graph and end up being too costly.

Since a chordal graph is one in which links between nodes have no cycle (i.e. closed paths) of length 4 or more distinct nodes without a short-cut through any other node, it is good for clique assignation.

The point is that, after moralization and triangulation, there exists for each node-parent set at least one clique which contains it, and thus a clique potential can be formed on the cliques of the triangulated graph.

The addition of edges to the moral graph does not stop a clique from being a complete subgraph, then for each clique in the graph there is at least one clique in the triangulated graph which contains it, meaning that the original model is still true after the additional edges.

- Clique extraction. Identify maximal cliques of the triangulated graph and join them together to form a set of cliques and build factor functions over them, hence constructing a cluster graph.

  Finding a set of cliques on chordal graphs is quite easy. One can perform a maximum cardinality search over the resulting chordal graph and collect maximal cliques generated in the process.

- Build the clique tree. We only need to find the edges between clique nodes to complete the cluster graph. We can achieve decent results and somewhat efficiently via a maximum spanning tree like one obtained with the Kruskal algorithm. The resulting cluster graph can be treated as a clique tree.

3.6 Conditional Random Fields

Markov Networks are a great tool for encoding the joint distribution over the set $\mathbb{X}$. As we have seen above, we can also use it to encode a conditional distribution $P(\mathbf{Y} \mid \mathbf{X})$, where $\mathbf{Y}$ is a set of target variables and $\mathbf{X}$ is a disjoint set of observed variables (evidence).

A Conditional Random Field is an undirected graph $\mathscr{H}$ whose nodes correspond to $\mathbf{Y} \cup \mathbf{X}$. Parametrized the same way as a Markov Network by a set of factors, the key difference lies in the distribution it represents, since it encodes $P(\mathbf{Y} \mid \mathbf{X})$ rather than $P(\mathbf{Y}, \mathbf{X})$. To do so, it disallows factors that involve only variables in the observed set $\mathbf{X}$, hence the factors are $\phi_1(\mathbf{U}_1), \ldots, \phi_f(\mathbf{U}_f)$ such that each $\mathbf{U}_i \nsubseteq \mathbf{X}$. This makes the encoded distribution as:

$$P(\mathbf{Y} \mid \mathbf{X}) = \frac{1}{Z(\mathbf{X})}\hat{P}(\mathbf{Y}, \mathbf{X}) \tag{3.22}$$

$$\hat{P}(\mathbf{Y}, \mathbf{X}) = \prod_{\phi_i \in \mathbf{\Phi}} \phi_i(\mathbf{U}_i) \tag{3.23}$$

$$Z(\mathbf{X}) = \sum_{\mathbf{Y}} \hat{P}(\mathbf{Y}, \mathbf{X}). \tag{3.24}$$

Any pair of variables will be connected by an undirected edge whenever they appear together in the scope of some factor. The only difference to the standard Markov network definition is the normalization used in the partition function $Z(\mathbf{X})$ which now is not constant, as it induces a different value for every assignment $\mathbf{x}$ to $\mathbf{X}$.

3.7 Inference on NOMT

We are ready to tackle the problem of data association on tracking presented in Eq. (2.9). To reduce the size of equations from now on, we omit the terms $\mathbf{D}_{t-\tau}^t$ and $\{\mathscr{I}_1^t, \mathscr{D}_{t-\tau}^t\}$. This leaves us with:

$$\hat{x} = \arg\min_x E(\mathbf{A}^{*t-1}, \mathbf{H}^t(x)), \tag{3.25}$$

where

- the set of clean tracks is $\mathbf{A}^{*t-1} = \{A_1^{*t-1}, \dots, A_m^{*t-1}\}$, with A_i^{*t-1} representing a single target whose trajectory is being tracked, with a sliding-window of size τ that is used to separate detections into "finalized partitions" (or clean tracks) A_k^{*t-1} and unassigned (or loose) tracks,
- the built set of hypotheses is $\mathbf{H}^t = \{\mathbf{H}_1^t, \mathbf{H}_2^t, \dots\}$, where each $\mathbf{H}_k^t$ is the set of hypotheses for the k-th target, denoted by $\mathbf{H}_k^t = \{\emptyset, H_{k,1}^t, H_{k,2}^t, \dots\}$. The null hypothesis $\emptyset$ represents the termination of said target due to occlusion or because it is leaving the scene.

Summing up, we describe the data association problem as the problem of associating each existing target A_k^{t-1} to a single hypothesis H_{k,x_k}^t, where x_k is the index of the hypothesis in $\mathbf{H}_k^{t-1}$. We search for the overall hypothesis assignation which is the most likely to form the real trajectories. Hence, the notation $\mathbf{H}^t(x)$ indicates a potential solution (or inferred partitioning) for detections in $\mathbf{D}_{t-\tau}^t$, where the indexing is given by x. The optimal solution is $\hat{x}$ and it will be used to augment A_k^{*t-1} and create the new trajectory A_k^{*t}. This problem can be solved using a graphical model.

3.7.1 Model representation

We can build an undirected graphical model, namely a Markov Network $\mathscr{H}$, where each node represents a single target. The set $\mathbb{X}$ corresponds to A^{*t-1} (the clean tracks that we will not revise from now), and the states associated to each k-th variable will be a (discrete) value for their corresponding possible hypotheses, to be chosen within $\mathbf{H}_k^t$. Edges connecting the nodes will be added between dependent variables. Among the strongest dependencies: Any hypothesis inside a set $\mathbf{H}_k^t$ contains a set of detections that can never be shared with another target. Hence, if any hypothesis in $\mathbf{H}_i^t$ shares a single detection with another hypothesis in $\mathbf{H}_j^t$, the value that A_i^{*t-1} can take depends on the value that A_j^{*t-1} can take (as some pairs of values are incompatible). Then, their corresponding targets (variables within the graphical model) are considered as dependent random variables.

When building the graphical model as described above, we end up with some nodes being disconnected and others rather connected together, or forming cliques, depending on how much their hypothesis overlap and share tracks (see for example Fig. 3.7). To take advantage of this graphical model with possibly many independent nodes, first, a Connected Component Analysis is applied to find and separate the original model into smaller and more manageable problems, as depicted on Algorithm 4, which can be done as a breadth-first search, such as shown in [11].

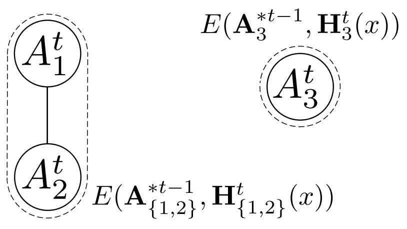

$E(\mathbf{A}_3^{*t-1}, \mathbf{H}_3^t(x))$
A_1^t
A_2^t
A_3^t
$E(\mathbf{A}_{\{1,2\}}^{*t-1}, \mathbf{H}_{\{1,2\}}^t(x))$

Figure 3.7: Consider an example of 3 objectives, two of them are dependent since their trajectories are similar enough for the tracker to have hypothesis sharing some tracks. Choosing any outcome for the node A_3^t will not affect any other targets, hence it is not connected and can be evaluated independently from targets A_1^t and A_2^t.

Algorithm 4 Connected Component Analysis

```
Input: Graph $\mathscr{H}$ with each node marked with “visited” flag as false.
Output: Set of components or sub-graphs $[\mathscr{C}]_1^C$
1: Add an empty component $\mathscr{C}_0$ without nodes and edges.
2: Set the current component $\mathscr{C}' = \mathscr{C}_0$.
3: Create an empty queue of nodes $\mathscr{Q} = \emptyset$.
4: for all node $A_k^{*t-1}$ in $\mathscr{H}$ do
5:    if $A_k^{*t-1}$ not visited then
6:       Mark $A_k^{*t-1}$ as visited.
7:       Push $A_k^{*t-1}$ to $\mathscr{Q}$.
8:       while $\mathscr{Q} \neq \emptyset$ do
9:          for all edge out of node in front($\mathscr{Q}$) do
10:            Remove edge from $\mathscr{H}$ and add to $\mathscr{C}'$.
11:            Move to node $A_j^{*t-1}$ connected by edge.
12:            if $A_j^{*t-1}$ not visited then
13:               Mark $A_j^{*t-1}$ as visited.
14:               En-queue $A_j^{*t-1}$ to $\mathscr{Q}$.
15:         Remove front node from $\mathscr{Q}$ and add it to $\mathscr{C}'$.
16:      Create new component $\mathscr{C}_c$ and set as current $\mathscr{C}' = \mathscr{C}_c$
17: return $[\mathscr{C}_c]_1^C$
```

For sub-graphs consisting of a single node A_k^{*t-1} that end up being independent, the “best” hypothesis can be directly picked in $\mathbf{H}_k^t$, whilst for nodes that depend on others, and to make sure that the best solution is chosen jointly for all of them, we pick the Maximum A Posteriori (MAP) using the junction tree algorithm presented in the previous sections, which turns the problem into a factor product of smaller configuration sets.

Note that the evaluation of the energy function $E(\mathbf{A}^{*t-1}, \mathbf{H}^t(x))$ results in real numbers, with large positive values for bad choices in the assignments of x and moderate negative values for plausible choices. In order for this to be used as a probability distribution, data is normalized as in equation (3.10) to obtain a Gibbs distribution, though this is done per-component and considering the following conventions.

3.7.2 Clique potentials as energy functions

In many fields of sciences such as thermodynamics and statistical mechanics, energy minimization is applied like in our formulated problem (3.25), and the function $E(x)$ is interpreted as the energy of a configuration x. Often, clique potentials are chosen with the form $\phi(x) = \exp(-E(x))$, which is called a Gibbs distribution.

The energy assigned by E is an indicator of the likelihood of corresponding relationships within the clique, with higher energy corresponding to lower probabilities and vice-versa. Rewriting our function and considering the partitioning done on the original graph, we evaluate per component:

$$\tilde{E}(\mathscr{C}, \mathbf{H}^t(x_{\mathscr{C}})) = \frac{1}{Z}\exp[-E(\mathscr{C}, \mathbf{H}^t(x_{\mathscr{C}}))] \tag{3.26}$$

$$Z = \sum_{x \in \mathbf{X}} \exp[-E(\mathscr{C}, \mathbf{H}^t(x))] \tag{3.27}$$

$$E(\mathscr{C}, \mathbf{H}^t(x_{\mathscr{C}})) = \prod_{\mathrm{C} \in \mathscr{C}} E(\mathrm{C}, \mathbf{H}^t(x_{\mathrm{C}})) \tag{3.28}$$

where each target in the clique $\mathrm{C} = \{A_i^{*t-1} \mid i \in \mathrm{C}\}$ in the component $\mathscr{C}$ compiles the contributions of the states $x_{\mathscr{C}}$ involved. Finding such cliques is done as explained in section (3.5) about the process of building a junction tree.

After calculating $\tilde{E}(\cdot)$ we simply pick the configuration or hypothesis set that gave the MAP. Solving this for every component gives the final best configuration, hence, solving the data association problem. Of course, this inferred solution is completely dependent on how well the energy function reflects good/bad assignments. As we made adjustments to the original proposal, we obtained good results for RGB-D data that will be presented on Chapter 6.

All we need now is a formal definition for how such dependencies in the clique components apply and for measuring how good is the compatibility of any given assignment for variables $\mathbf{X} = x$. The model for this should encode an evaluation similar to what a human takes into account when keeping track of a person or of an objective within sight, by considering its appearance and its motion to decide on the objective identity.

3.7.3 Choi's energy function

The definition of factors formulated by Choi [6] describes the joint distribution of the (in-)compatibility of x as an energy:

$$E(\mathbf{A}^{*t-1}, \mathbf{H}^t(x)) = \sum_m \Psi(A_{m,x_m}^{*t-1}, H_{m,x_m}^t) + \sum_{m,l} \Phi(H_{m,x_m}^t, H_{l,x_l}^t). \tag{3.29}$$

This original formulation considered a similarity measure called the Aggregated Local Flow Descriptor (ALFD), which operates over the 2D image, finding interest point color patches on detected rigid bodies. We first revisit potentials that are involved on the evaluation of E and re-interpret their calculation for the domain of RGB-D data:

- The function $\Psi(.)$ is the Single Target Consistency that measures the likelihood of assigning H^t_{m,x_m} to a target A^{*t-1}_m:

$$\Psi(A^{*t-1}_{m,x_m}, H^t_{m,x_m}) = \sum_{i \in H^t_{m,x_m}} \psi_u(A^{*t-1}_m, d_i) + \sum_{\substack{(i,j)\in H^t_{m,x_m} \\ i \neq j}} \psi_p(d_i, d_j) \quad + \quad \psi_h(A^{*t-1}_m, H^t_{m,x_m}) \tag{3.30}$$

which is the sum of three terms:

- ψ_u is an Unary Potential which takes every detection d_i in the target hypothesis H^t_{m,x_m} and tests its compatibility/similarity with A^{*t-1}_m as denoted by the use of the min function:

$$\psi_u(A^{*t-1}_m, d_i) = \min\Big(\ \mu_A(A^{*t-1}_m, d_i)\ ,\ \mu_T(A^{*t-1}_m, d_i)\ \Big) - s_i. \tag{3.31}$$

The min will mainly choose the term of appearance similarity $\mu_A < 0$, unless we are certain of its membership to the trajectory by a dynamics prediction test $\mu_T < 0$. This potential will result in $-s_i$ (s_i being the score of detection d_i), whenever $A^{*t-1}_m = \emptyset$, hence favoring assignments to trajectories that have a history past the sliding window, and giving the smallest sway to new targets.

  * μ_A is an Affinity Metric:

$$\mu_A(A^{*t-1}_m, d_i) = -\sum_{\Delta t \in \mathscr{N}} a_A(d(A^{*t-1}_m, t_i - \Delta t), d_i), \tag{3.32}$$

where $\mathscr{N} = \{1, 2, 3, 4, 5, 10, 20\}$ is a fixed set of frame distances, defining a time neighborhood from the current detection d_i to the previous detections in A^{*t-1}_m, $d(A^{*t-1}_m, t_i)$, which gives the associated detection of A^{*t-1}_m at time step t_i. The function a_A is the similarity measure whose evaluation will be studied in depth on Chapter 4. This measure is designed to tell us how similar are two detections, by considering geometry and appearance, resulting in a value between $[-1, 1]$, respectively indicating mismatch or match. In the most optimistic case in which every single detection in the clean track A^{*t-1}_m exists in the neighborhood at time t_i for d_i and they are perfectly similar, the sum results in $-\text{size}(\mathscr{N})$ and $+\text{size}(\mathscr{N})$ on the worst case. This means that the selection on how big the neighborhood should be impacts the contributions of this potential and that longer tracks are favored.

  * μ_T is a Target Dynamics function:

$$\mu_T(A^{*t-1}_m, d_i) = \begin{cases} \infty & \text{if } o^2(p(A^{*t-1}_m, t_i), d_i) < 0.5 \\ -\eta^{t_i - f(A^{*t-1}_m)} o^2(p(A^{*t-1}_m, t_i), d_i) & \text{otherwise,} \end{cases} \tag{3.33}$$

where o is an overlapping measure and $p(A, t)$ a predictor for the position at t of the target corresponding to clean track A. The output that takes an infinite value (first case above) will never happen because of the aforementioned min whenever the overlap between track d_i and its predicted position $p(A^{*t-1}_m, t_i)$ using LSQ as presented in 2.2.4, is inferior to a threshold. We make mostly use of the appearance,

unless the overlap indicates a very good match. The function $f(A_m^{*t-1})$ denotes the last associated frame of the target and the scalar $\eta = 0.98$ is a decay factor that discounts long term prediction. Hence, the further frames separate t_i from the clean track, the less contribution to μ_T from these frames.
In the context of 3D Point Clouds, since we use bounding-cylinders to represent targets, we evaluate o^2 as the squared IoU by measuring the area of intersection between circumferences using the formula (2.3), requiring an IoU of 0.7 to pass the threshold.

Note that this potential translates to the intuition that "whenever we are unsure of the appearance being good enough, we give a chance of still minimizing with a negative weighted overlap value rather than penalizing with a positive value due to low similarity... only if said overlap is very large."

- ψ_p is a Pairwise Potential that measures the inner appearance consistency of hypothesis H_{m,x_m}^t through:

$$\psi_p(d_i, d_j) = \begin{cases} -a_A(d_i, d_j), & \text{if } |d_i - d_j| \in \mathcal{N} \\ 0 & \text{otherwise} \end{cases} \tag{3.34}$$

with the same set $\mathcal{N}$ of allowed frame distances. It is very similar in nature to Equation (3.32), which measures the similarity between clean tracks with respect to loose tracks. This potential intends to measure how good is the similarity between pairs of detections, but inside the hypothesis itself. Again, this potential scales with the number of possible pairs, favoring long trajectories in hypothesis H_{m,x_m}^t with high similarity, and vice versa. Although not mentioned by Choi's paper on NOMT, this potential should also not be giving a sway whenever $A_m^{*t-1} = \emptyset$, which gave better experimental results, specially on cases in which an entering target is evaluated against another one with non-null history. Both end up getting the same pairwise potential, whilst having no history should not be as important as an observed trajectory that has been tracked for a while. Summing up, this potential allows to "prefer long self-similar trajectories for an existing target with a past, rather than shorter or dissimilar trajectories with no past.".

- ψ_h is a High Order Potential that penalizes abrupt motion and low similarity over time:

$$\psi_h(A_m^{*t-1}, H_{m,x_m}^t) = \gamma \sum_{i \in H_{m,x_m}^t} \xi(p(A_m^{*t-1}, t_i)\ ,\ d_i) + \epsilon \sum_{\substack{(i,j) \in A_m^{*t-1} \cup H_{m,x_m}^t \\ i \neq j}} \theta - K(d_i, d_j). \tag{3.35}$$

where γ, ϵ and θ are constant values. Differently from Choi, we operate with 3D coordinates rather than 2D bounding boxes, hence, here, ξ simply measures the normalized euclidean distance between predicted positions with $p(A_m^{*t-1}, t_i)$. We use only clean tracks rather than $A_m^{*t-1} \cup H_{m,x_m}^t$, as that could lead to wrong fittings when the hypothesis is nowhere near the target. Using only clean tracks helps giving out higher penalization for such cases in which the hypothesis is not near A_m^{*t-1}, and this potential will not give sway whenever $A_m^{*t-1} = \emptyset$. The second term considers every pair in the whole trajectory augmented with the considered hypothesis, adding up $K(d_i, d_j) \in [0, 1]$ which is the intersection kernel between histograms of d_i and d_j, giving 1 if the histograms are identical, 0 otherwise. More details on the calculation of these histograms will be given in Chapter 4. The constant values we used are $(\gamma, \epsilon, \theta) = (20, 1, 0.8)$. The high order potential could be interpreted as "preferring assigning hypothesis that fall right inside the predicted or extrapolated path of existing long target trajectories, also, favoring high similarity. Otherwise, penalties are high".

- $\Phi(.)$ is a Mutual Exclusion term that gives huge penalties when two targets try to use a pair of hypothesis that have any overlap or that share common tracks:

$$\Phi(H^t_{m,x_m}, H^t_{l,x_l}) = \sum_{f=t-\tau}^{t} \alpha \cdot o^2(d(H^t_{m,x_m}, f), d(H^t_{l,x_l}, f)) + \beta \cdot \mathrm{I}(d(H^t_{m,x_m}, f), d(H^t_{l,x_l}, f)), \quad (3.36)$$

  where α and β are constant parameters and $d(H^t_{m,x_m}, f)$ gives the associated detection of H^t_{m,x_m} at time f (if none, $\emptyset$ is returned), $o^2(d_i, d_j) = \mathrm{IoU}(d_i, d_j)^2$ and $\mathrm{I}(d_i, d_j)$ is an indicator function that results in 1 whenever H^t_{m,x_m} and H^t_{l,x_l} share any common track (i.e., same detection and frame). This potential is a key for avoiding undesirable configurations by penalizing too much overlap between hypotheses and having duplicate assignments of detections, which will overshadow any good sway on Single Target Consistency. We use $(\alpha, \beta) = (0.5, 100)$. Hence, this potential kind of forces a "probability" of almost 0 on the conditional random field for inconsistent assignments x.

3.8 Our proposed Energy for RGB-D NOMT

After studying how the energy function was designed and after having experimentally tested and observed the behavior of the previously presented energy, we realized that the original model might suffer from scaling problems in data and does not make it clear on what happens when certain configurations occur. Though it has provided good results on the adaptation to RGB-D data as is, we noticed some issues that can be dealt with differently.

3.8.1 Data association term

To begin explaining our proposed changes, we re-organize each term in Single Target Consistency function for an overview as follows:

$$\begin{aligned}&\Psi(A^{*t-1}_m, H^t_{m,x_m}) = \\ &\sum_{i \in H^t_{m,x_m}} \min\Big(\Big[-\sum_{\Delta t \in \mathcal{N}} a_A(d(A^{*t-1}_m, t_i - \Delta t), d_i)\ \Big]\ ,\ \Big[-\eta^{t_i - f(A^{*t-1}_m)} \cdot o^2(p(A^{*t-1}_m, t_i), d_i)\Big]_{o^2<0.5?o^2:\infty}\Big) - s_i + \\ &\sum_{\substack{(i,j) \in H^t_{m,x_m} \\ |i-j| \in \mathcal{N}}} -a_A(d_i, d_j) + \gamma \sum_{i \in H^t_{m,x_m}} \xi(p(A^{*t-1}_m \cup H^t_{m,x_m}, t_i), d_i) + \epsilon \cdot \sum_{(i,j) \in A^{*t-1}_m \cup H^t_{m,x_m}} \theta - K(d_i, d_j).\end{aligned} \quad (3.37)$$

We make the following observations:

1. On the first line, the first term in the min operator is a detection-to-detection similarity score accumulated over the target's existing trajectory (indices Δt) and over the candidate tracklet

detections (indices i); the second term is a spatial likelihood term that measures the closeness (overlap) of the detection considered in the tracklets to the predicted position p of the target for the detection timestamp t_i; this term importance is weighted by a factor $-\eta^{t_i - f(A_m^{*t-1})}$ that exponentially decreases with the time span between the last timestamp present in the target and the detection timestamp. The scalar s_i is the detector confidence (score) in the detection d_i. This whole line remains unchanged for our design.

2. The terms in the second line measure the appearance consistency within the hypothesis (terms in $a_A(d_i, d_j)$), the spatial consistency (by measuring the distance between the detections and the interpolated trajectory using both the target and the tracklet data) and, finally, another appearance consistency $K(d_i, d_j)$.

3. a_A was the ALFD evaluation from Choi, i.e. an appearance similarity score. It works by training some model weights that encode 'overlap' with GT data. What it measures, is a voting or consensus on whether a pair of Interest Points (FAST patches inside the bounding box) match within a particular pyramidal grid. It is said to measure "local flow" and give values in the interval $[-1, 1]$. It just accounts for color patches that were found on the same places, but not the general appearance of the target.

This energy function relies heavily on a_A for the Single Target Consistency (3.37). However, the last term $(\theta - K(\cdot, \cdot))$ seems to be included in a ad-hoc way to appeal for measuring what ALFD does not use: the pyramidal LAB histograms between detection d_i and d_j. In some way, it appears a bit redundant since we made our similarity measure to encode this information already from point-cloud processing.

3.8.2 Hypothesis consistency term

Mutual exclusion (3.36) is the only part of the potential function that encodes a relationship between a pair of "outcomes" in the network, resulting in an edge between targets whenever hypotheses share any track. This potential has proven to correctly avoid such decisions that are impossible in the context of tracking, and does not seem to require any adjustment. As said above, it acts as a prior on the distribution of possible assignments.

The left term penalizes the overlap between detections made in a pair of different target-tracklet pair, at the same time instant, while the right term adds a huge penalization if two detections are the same and are both associated to different targets.

3.8.3 Changes

- Considering what is mentioned above, and since our own similarity function already encodes geometry and color appearance rather than a local flow of patches, we think that the last term in Equation (3.37) could be discarded, as it will just be kind of redundant with the first term of the min operator and with the second term (pairwise-potential). Hence:

$$\epsilon \cdot \sum_{(i,j) \in A_m^{*t-1} \cup H_{m,x_m}^t} \theta - K(d_i, d_j) \Longrightarrow \emptyset. \tag{3.38}$$

- We observe that the third term in equation (3.37), or high order potential, uses a polynomial fitting over the set of detections in the union $A_m^{*t-1} \cup H_{m,x_m}^t$. We found that this choice artificially penalizes those targets with $A_m^{*t-1} = \emptyset$. For the targets with non-empty clean tracks, the fitting with the hypothesis, when the tracks are quite distant or far from the actual trajectory, tends to minimize the adjustment error on the overall tracks but does not measure the real error on the observed trajectory and the tested hypothesis. Hence we changed the term to just:

$$\gamma \sum_{i \in H_{m,x_m}^t} \xi(p(A_m^{*t-1} \cup H_{m,x_m}^t, t_i), d_i) \Longrightarrow \gamma \sum_{i \in H_{m,x_m}^t} \xi(p(A_m^{*t-1}, t_i), d_i). \tag{3.39}$$

- Trough experimentation, we observed that some situations such as occlusions are not directly well encoded in the formulation and it is noted that ALFD is vulnerable to them since the absence of some interest points does not help in the voting that it uses for similarity. Also, the original NOMT, even if it does fix bad associations when the occlusion occurs inside the sliding window, does not perform well when the occlusion lasts longer[8] and when the predicted movement now does not fit the missing target's new real trajectory.
  - We could think of an occlusion as a situation where a target or whatever 3D object affects the outcomes of other targets. In front of a RGB-D sensor, targets nearer to the sensor retain most re-projected points and those behind get occluded, losing density in their points. Should we make a counter-weight for the High Order Potential, such that when similarity is good enough but movement was not so smooth, it is less penalizing (remember that it will not do anything other than penalizing, even when normalized).
  - We should make our similarity function better and more resistant to the loss of points and to the partial view of the detection, also so that configurations with detections from the same objective will stand out, no matter the separation in time provoked by the occlusion (non-smooth movement).

3.8.4 Occlusion Potential

We attempt to encode the following statements:
- "If an observed trajectory appears to be under occlusion, we should penalize less a movement that was not too smooth."
- "Otherwise, when no occlusion is evident, we let non-smooth movement to be completely penalized."
- "Also, as a safeguard, when occlusion is possible for the first case, we require that, though non smooth movement is allowed, appearance similarity must be good enough, meaning that when H_{m,x_m}^t is not similar enough to A_m^{*t-1}, then it is more likely for this tracklet to be a new target."

[8]Though the original energy function could be capable of correctly re-activating again a target that went missing as long as T_{active} holds, when the occlusion lasts longer than the size of the sliding window, the target movement and the appearance might not be good enough for it to beat the bet of starting a new tracked target rather than continuing the old trajectory. This could have been achieved by Single Target Consistency, but only if a_A was really good to match two detections separated by the time the occlusion occurred.

We designed the following equation to be a representation of the above (to replace Choi's High Order Potential):

$$\gamma \sum_{i \in H^t_{m,x_m}} \xi(p(A^{*t-1}_m, t_i), d_i) \Longrightarrow \Omega \sum_{i \in H^t_{m,x_m}} \omega(A^{*t-1}_m, d_i) \cdot \xi(p(A^{*t-1}_m, t_i), d_i) \tag{3.40}$$

where

$$\omega(A^{*t-1}_m, d_i) = \begin{cases} 1 - \dot{\omega}(A^{*t-1}_m, d_i) & \text{if } \bar{\mu}_A(A^{*t-1}_m, d_i) \geq \gamma \geq 0 \\ 1 & \text{otherwise.} \end{cases} \tag{3.41}$$

It encodes the idea that when similar, we penalize less, proportionally to the occlusion, and when not similar, we penalize as usual. The term $\bar{\mu}_A(A^{*t-1}_m, d_i)$ is the average original unary potential (average similarity toward the observed trajectory):

$$\bar{\mu}_A(A^{*t-1}_m, d_i) = -\frac{1}{\text{size}(\mathscr{N})} \sum_{\Delta t \in \mathscr{N}} a_A(d(A^{*t-1}_m, t_i - \Delta t), d_i). \tag{3.42}$$

It yields a value between $\gamma \leq \bar{\mu}_A \leq 1$, where 1 means that d_i is very similar. So, being similar implies a chance to reduce penalization when occluded. We also use the notation $\dot{\omega}(A^{*t-1}_m, t_i) \in [0, 1]$, with $\dot{\omega}_{t_i} = 1$ when occlusion is sure and $\dot{\omega}_{t_i} = 0$ when there is a priori no reason to think the target A^{*t-1}_m is undergoing an occlusion at time t_i.

Still, this is not a so simple problem of deciding whether a target is occluded or not at time t_i. The idea of using depth points in the person cloud makes a lot of sense, even more after down-sampling is performed. Indeed, nearer in the range of depth sensors, points are more saturated, and further, the density of points per surface is inferior. Our down-sampling is made by limiting a single point per sphere of a given radii. This makes D_{t_i} keep an almost consistent number of 3D points, no matter whether the target is further or nearer. What may influence this number of points are self occlusions, self-enlargement of the target (not by getting closer but rather by having more surface) and other target occlusions, even by disappearing from the frame. Intuitively, we think of a target having a number of points in time to behave like a normal distribution, having a mean amount of points over time and a variance indicating how much self-occlusions occur or changes in its surface. So, a significant reduction or increase in the number of observed points will be an indication of occlusion by another target / obstacle or bigger target, even disappearance. So we define:

$$\dot{\omega}(A^{*t-1}_m, d_i) = \begin{cases} e^{-\frac{(n_d(\mathcal{P}_i) - \mu_{n_d})^2}{2\sigma^2_{n_d}}} & \text{if } n_d(\mathcal{P}_i) < \mu_{n_d} \\ 0 & \text{otherwise} \end{cases} \tag{3.43}$$

$$\mu_{n_d} = \frac{\sum_i n_d(\mathcal{P}_i)}{\# A^{*t-1}_m} \tag{3.44}$$

$$\sigma^2_{n_d} = \frac{\sum_i n_d(\mathcal{P}_i)^2}{\# A^{*t-1}_m - 1} - \mu^2_{n_d}. \tag{3.45}$$

This will consider observed points in the clean tracks, and make a valuation of how occluded is the given detection, that already needs to be similar thanks to definition of ω. This is for it to make it more sensitive to how much changes in the number of points are perceived over time for a target.

This would allow for targets that show sudden non-smooth changes in their movement, in direction or even speed, and avoid the need of thinking a trajectory belongs to a new target rather than an already observed target, as long as they are similar in appearance.

3.8.5 Interpretation of our RGB-D NOMT energy function

Rewriting the updated energy function, and changing the ALFD descriptor $a_A(d_i, d_j)$ by our own point-cloud descriptor, $a_R(d_i, d_j)$, we move around some terms for a convenient study of what is 'encoded':

$$E'(\mathbf{A}^{*t-1}, \mathbf{H}^t(x)) = \sum_m \Psi(A_m^{*t-1}, H_{m,x_m}^t) + \sum_{m,l} \Phi(H_{m,x_m}^t, H_{l,x_l}^t) \tag{3.46}$$

$$\begin{aligned}&\sum_m \Psi(A_m^{*t-1}, H_{m,x_m}^t) = \\ &-\sum_m \sum_{i \in H_{m,x_m}^t} s_i - \sum_m \sum_{\substack{(i,j) \in H_{m,x_m}^t \\ |i-j| \in \mathcal{N}}} a_R(d_i, d_j) + \Omega \sum_m \sum_{i \in H_{m,x_m}^t} \omega(A_m^{*t-1}, d_i) \cdot \xi(p(A_m^{*t-1}, t_i), d_i) + \\ &\sum_m \sum_{i \in H_{m,x_m}^t} \min\Big(\big[-\sum_{\Delta t \in \mathcal{N}} a_R(d(A_m^{*t-1}, t_i - \Delta t), d_i) \ \big] \ , \ \big[-\eta^{t_i - f(A_m^{*t-1})} \cdot o^2(p(A_m^{*t-1}, t_i), d_i) \big]_{o^2 < 0.5 ? o^2 : \infty} \Big)\end{aligned} \tag{3.47}$$

$$\sum_{m,l} \Phi(H_{m,x_m}^t, H_{l,x_l}^t) = \sum_{m,l} \sum_{f=t-\tau}^{t} \alpha \cdot o^2(d(H_{m,x_m}^t, f), d(H_{l,x_l}^t, f)) + \beta \cdot \mathrm{I}(d(H_{m,x_m}^t, f), d(H_{l,x_l}^t, f)) \tag{3.48}$$

Remember that x_m is what varies in this sum for m, meaning that it "toggles" a certain selection of H_{m,x_m}^t for A_m^{*t-1}. This means that in a "simple" case with 3 targets, each with two hypothesis, then $3^2 = 9$ combinations of x exist, some of which are easily discarded by mutual exclusion. But some plausible assignments are differentiated with this potential Ψ, and it is the key for choosing the best configuration.

- The first term in (3.47) is simply the sum of all the detection scores inside the active hypothesis. It means that null hypotheses do not contribute. Supposing that every detection is perfect, the highest value that can be obtained here is when the configuration to be chosen actually gives, for all targets, a non-null hypothesis that covers the whole interval $[t - \tau, t]$. Also, longer hypothetical trajectories are favored, which is actually what we want. It does not make any assertion on the appearance nor on the kind of movement.
- The second term in (3.47) is a pairwise potential that only applies to tracklets. It does not even consider the history or the observed trajectory. Again, null hypotheses do not help and longer hypothesis are better. This potential is just the sum of similarities between neighboring detections in the hypothesis. It can be said to consider appearance since it solely depends on

what a_R encodes. It introduces a penalization whenever a_R is good enough to indicate that two different targets are dissimilar, giving negative values.

- The third term in (3.47) is the occlusion-friendly high order potential, and is a "penalize only" potential, since it is the sum of distance errors between a predicted position in time and the observed position at that time. So, any hypothesis that follows/falls in the predicted path will give minimal errors, tending to be at best 0. Here, a special case occurs with the null hypothesis, which are benefited, as they cannot be tested here and obtain zero penalization. This potential will now penalize less strongly when the appearance of the hypothesis and the observed trajectory is good enough and when there is a possibility that the target is being occluded (i.e., when it has fewer points in its cloud than usual).
- The last term in (3.47) is a bit tricky, as the min function works as a switch for choosing between two potentials: One is similar to the pairwise potential but using also A_m^{*t-1} (it seems to clash with second term, or at least duplicate contributions), which is used most of the times and the other potential that is only used when the overlap between predicted and seen detection is really high (it seems to complement the third term). It encodes the sense of a last chance for a favorable potential: if similarity was not good enough, but if the movement is just fitting very well, it allows a contribution instead of a penalization for not being similar enough.

Mutual exclusion (3.48) remains the same. Now with this we are ready to use (3.46) as our factor definition for individual cliques in our sub-graphical models obtained with Algorithm 4. The energy from each independent clique is combined per component, using Algorithm 3, and finally each component's joint distribution is combined as explained in equation (3.28). After the full joint distribution of the entire model is available, we can just choose the x that is the best overall association of trajectories for each target.

Having fully defined the efficient inference process with a model that takes advantage of RGB-D data for better occlusion resistance, we now present our formulation of $a_R(d_i, d_j)$ to appeal for geometry and color on non-rigid objectives.

Chapter 4

RGB-D Similarity

4.1 Point-Cloud Descriptor

A key difference between the context in which W. Choi proposed his original ALFD descriptor [6] and the needs in our approach is that a RGB-D descriptor for our system needs to encode the motion of the detected person and to track its changing geometry, form and color, since a walking human is by essence not a rigid object and is hence very different from a car that gets closer to the camera as in the aforementioned work.

Most approaches, including Choi's one [6], extract interest points on the 2D space of the image, inside the rigid body's bounding box, whilst we wish to track a person that could be walking, moving its limbs and occluding itself, making the extraction of image interest points less useful a priori.

To get a better grasp on each person's geometry and position, we work on the full 3D point-cloud information rather than on 2D interest points. The cloud data gives the possibility of applying more interesting 3D techniques such as body part segmentation, pose estimation (skeleton) and estimation of the local surface orientations, to give a few examples.

However, let us recall that RGB-D sensors provide very dense point-clouds of the observed scenes, which may produce a large overhead in the processing pipeline. To alleviate this burden, we consider that any detection d_i lives in a vertically-oriented 3D "bounding-cylinder", centered on 3D coordinates $< d_i[x], d_i[y], d_i[z] >$ and with a circumference of diameter $d_i[r]$ and a height $d_i[h]$. At this point, we do not take into account the scenario in which the target moves, and we are only interested in the point-cloud data inside this bounding-cylinder. Using down-sampled RGB-D data $\{\mathscr{I}_1^t, \mathscr{D}_1^t\}$ and said bounding cylinder, we extract individual point-clouds for each detection, which will be referred to as $\mathscr{P}_i$. An example of $\mathscr{P}_i$ is given in fig. 4.1.

Even after downsampling, the raw point-cloud data is still full of redundant information and

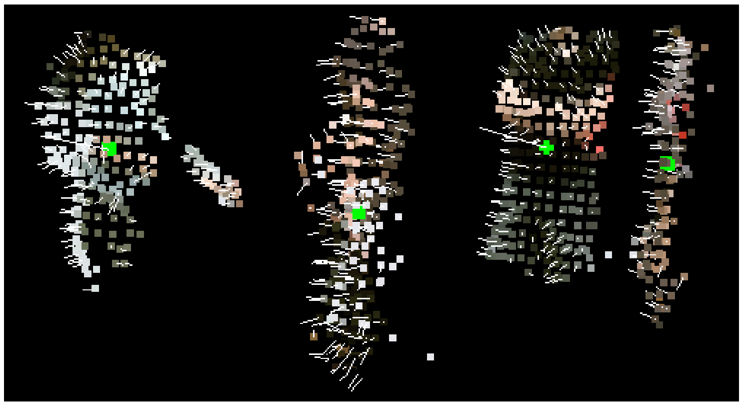


Figure 4.1: Using the Point Cloud Library (PCL) [28] and our simulation software, we are able to extract and down-sample each person's point cloud $\mathcal{P}_i$, and to perform an analysis on its geometry and color content. In this picture, we show the estimation of the normal vectors on the surface of the point cloud. These normal vectors are used to create a descriptor associated to each detection.

needs to be converted into a more compact representation, easier to process and to compare to others. This is what we call a descriptor. There are many approaches and choices to compose a descriptor from raw data: histograms, bit strings and bag of words models, among others.

4.2 Geometry and surface estimation

Most descriptors over point-clouds require to make an analysis over the neighborhood of a point to find geometric parameters such as the local curvature or the surface normal at this point. This needs to be done while avoiding the problems caused by outlier points. This is why many descriptors are local features, as they characterize a single point by considering its k closest neighbors, including noisy data. Many feature descriptors have been designed to be translation-, rotation-, moment- and volume-invariant, and as least sensitive to noise as possible. The selection on how big the neighborhood should be is very important, since the descriptor will describe the underlying surface along which this neighborhood is positioned. A small k could result in large errors in the feature estimation, whilst a large k might suppress small details that could have been helpful for characterization (as a low-pass filter would do). An illustration is given in Fig. 4.2. Once a particular point and its neighbors are picked, then in order to find the surface normal, an approximation using Principal Component Analysis (PCA) is common and performs rather efficiently.

PCA is a statistical technique used to describe multidimensional data in terms of a new set of, if possible, non-correlated variables called components, which are focused on capturing the variance of the original data. PCA results naturally in a projection process, which leads to a reduction on the data dimension and on the complexity of the data. It may also allow a better understanding on which are the most informative variables in the sample, down to the least. In the context of point-clouds, PCA applies a linear transformation over the set of the k nearest neighbors $[p_i]_1^k \subset \mathscr{P}_i$ of a query point p_q, each of which described with $m = 3$ dimensions $< x, y, z >$. Intuitively, since

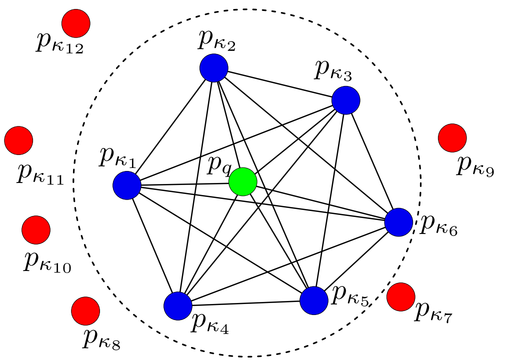


Figure 4.2: Influence region diagram for a query point p_q (in green) with its k neighbors, within a 3D sphere of radius r (in blue). All the points inside the neighborhood are turned into a fully connected mesh, that is used to describe all the point pair relationships between normal orientations. This process has a complexity of $O(k^2)$.

points $[p_i]_1^k$ are distributed roughly as a surface due to their origination from a RGB-D sensor observing objects, they will tend to be mostly varying along two directions, corresponding to their common surface, with the least variance along the normal of such surface. Hence, PCA should output a base of 3 vectors, the first two (largest components) capturing the variations on the surface, while the smallest component will most likely be similar in direction to the real surface normal.

Formally, PCA takes input data gathered as $\mathbf{D} \in \mathbb{R}^{k\times 3}$, i.e. a matrix such as

$$\mathbf{D} = \begin{bmatrix} d_1[x] & d_1[y] & d_1[z] \\ d_2[x] & d_2[y] & d_2[z] \\ \vdots & \vdots & \vdots \\ d_k[x] & d_k[y] & d_k[z] \end{bmatrix}, \tag{4.1}$$

where each column can be considered as a set of samples of random variables $[D_X, D_Y, D_Z]$, respectively. A simple method for calculating the principal components is to build a correlation matrix $\mathbf{C} = [r_{ij}] \in \mathbb{R}^{3\times 3}$ where:

$$r_{ij} = \mathrm{corr}(D_i, D_j) = \frac{\mathrm{cov}(D_i, D_j)}{\sqrt{\mathrm{var}(D_i)\mathrm{var}(D_j)}} \equiv \frac{\sum_k (d_k[i]-\mu_i)(d_k[j]-\mu_j)}{\sqrt{\sum_k (d_k[i]-\mu_i)^2 \sum_k (d_k[j]-\mu_j)^2}}, \tag{4.2}$$

where μ_j gives the mean of the data contained in column j.

$\mathbf{C}$ is a symmetrical matrix that, in general, can be diagonalized: $\mathbf{C} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}^{-1}$ with $\mathbf{\Lambda}$ being a diagonal matrix whose elements are the variances or eigenvalues:

$$\sigma(\mathbf{C}) = \{\lambda_i \mid \mathbf{C}v = \lambda_i v \quad \forall i = 1,2,3\}, \tag{4.3}$$

and $\mathbf{P}$ is the matrix whose columns are the eigen-vectors of $\mathbf{C}$, that span the eigen-space associated to each λ_i in the same order established by $\mathbf{\Lambda}$. Hence, they form a kernel of the matrix $(\mathbf{C} - \lambda_i\mathbf{I})$:

$$\mathbf{P} = [v_1, v_2, v_3], \quad v_i \in \ker(\mathbf{C} - \lambda_i\mathbf{I}) \quad \forall i = 1,2,3. \tag{4.4}$$

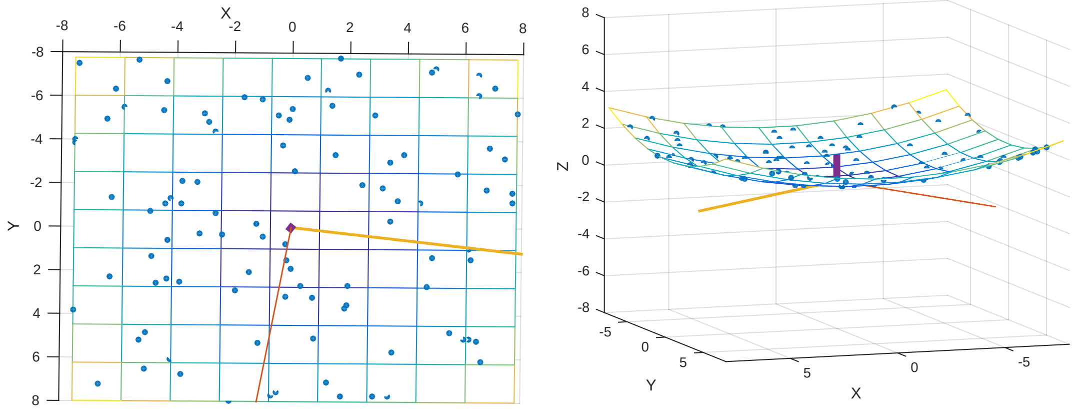


Figure 4.3: As a demonstrative exercise, we sampled values $X, Y \sim \mathrm{U}(-8, 8)$ and $Z = \frac{1}{40}[X^2 + Y^2]$ to create a curved surface similar to what RGB-D data could provide. Applying PCA over 100 samples, we obtained the eigen-vectors drawn in the figure, with the length of their corresponding eigen-value. In this controlled case, the largest eigen-vectors tend to align and live in the XY plane as the data has bigger variance along those dimensions. The smallest variance occurs along Z, which results in the smallest eigen-vector as our approximation of the normal in the origin.

Up to this point, we take the smallest eigen-vector in terms of its eigen-value, here identified with v_0, which can be either v_1, v_2 or v_3 when PCA is applied. This smallest eigen-vector will correspond to our approximation of the surface normal. An illustration is given in Fig. 4.3.

A potential problem is the direction in which the normal should point. An easy test for verifying and correcting these directions in $\mathscr{P}_i$ is to use the angle θ formed between the camera view-point z_c, translated to point p_i, and the estimated normal n_{p_i}. This angle must be between $[\pi/2, \pi]$ for it to be correctly oriented, since the camera should be in principle located in the exterior of the shape. Hence, we can simply rely on the vectors inner product formula to calculate the value of $\cos\theta$. We evaluate:

$$\frac{\langle z_c - p_i, n_{p_i} \rangle}{\|z_c - p_i\| \|n_{p_i}\|} = \cos\theta. \tag{4.5}$$

If $\cos\theta \geq 0$, we make $n_{p_i} = -n_{p_i}$. Otherwise, the normal n_{p_i} is pointing towards the viewpoint, which is correct. This test is done over all points $p_i \in \mathscr{P}_i$. In Fig. 4.4, we give an illustration of this process.

Summing up, the computation of surface normals in the point-cloud is shown in Algorithm 5.

4.2.1 Point Feature Histogram (PFH)

Histogram-based descriptors are easy to compute and encode the neighborhood geometrical properties much better than other representations. Moreover, they are invariant to scale and pose changes. In this section, we focus on the descriptor FPH as presented in [27] and [29].

Having the cloud $\mathscr{P}_i$ and its associated surface normals, we can obtain a generalization of the mean surface curvature at a given point p using histograms: We make a count of the occurrences of specific normal values, with the goal of providing an informative signature, invariant to the pose

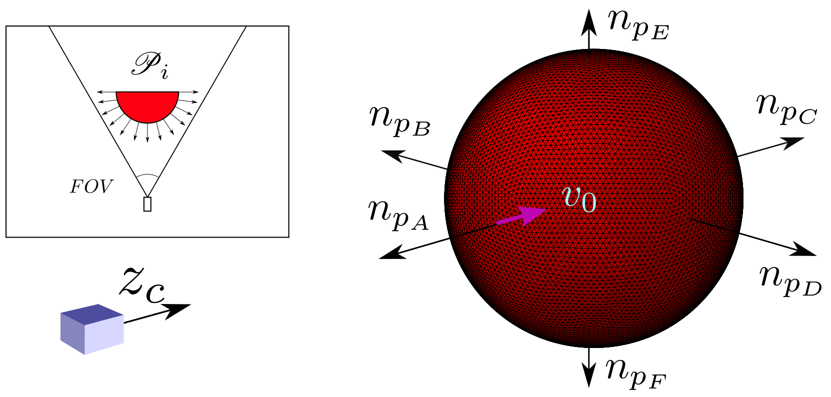


Figure 4.4: Suppose that the surface in the previous figure belongs to a sphere that is being observed by a RGB-D sensor right on its front. Hence, its viewpoint is aligned to the real normal illustrated as n_{p_A}. The smallest eigen-vector v_0 (in pink) of the previous solution is oriented backwards. Using (4.5), we can correct the orientation of this vector and repeat this process for every normal in the observed surface, up to every perpendicular such as n_{p_B}, n_{p_D}, n_{p_E}, n_{p_F}, which marks the end of observable surfaces. Any real surface whose normal is faced on the opposite orientation up to perpendicular normals is considered as non-observable, such as n_{p_C}.

Algorithm 5 Estimation of Surface Normals

```
Input: $k$ - neighborhood size
    $z_c$ - current view-point
    $r$ - sphere radius for neighbor search
    $\mathscr{P}_i$ - set of 3D points within a detection $d_i$.
Output: $[n_{p_i}] \quad \forall p_i \in \mathscr{P}_i$.
1: for all $p_i$ in $[p_i]_1^k$ do
2:   if $p_i$ is missing its surface normal $n_{p_i}$ then
3:     Find a $k-$neighborhood $[p_i]_1^k$ of $p_i$ enclosed by a sphere of radius $r$ around $p_i$.
4:     $n_{p_i} = v_0$ where $v_0$ is the eigen-vector corresponding to the smallest eigen-value using PCA
       over the k-neighborhood coordinates.
5:   if $\frac{\langle z_c - p_i, n_{p_i} \rangle}{\|z_c - p_i\|} < 0$ then
6:     $n_{p_i} = -n_{p_i}$
7: return $[n_{p_i}] \quad \forall p_i \in \mathscr{P}_i$.
```

of the underlying surface and resistant to resolution loss during down-sampling.

A Point Feature Histogram (PFH) intends to encode local geometrical properties between a point and its k-neighborhood and their estimated surface normals and mean curvature around the point, using a multidimensional histogram of values. The idea is to take into account all the interactions between the directions of the estimated normals.

For every pair of points p_i and p_j where $j < i$ in the k-neighborhood of the query point p_q, with their estimated normals n_i and n_j, we select among them a source p_s and a target p_t, with the source being the one having the smaller angle between the associated normal and the line

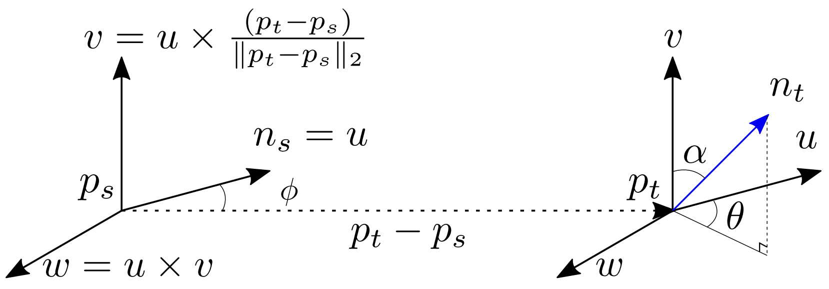


Figure 4.5: The coordinates systems around n_s and n_t allow to evaluate their relative orientation transform.

connecting the points:

$$\text{IF}\Big[\text{acos}(\langle n_i, p_j - p_i\rangle) \leq \text{acos}(\langle n_j, p_i - p_j\rangle)\Big] \text{ THEN } p_s = p_i,\ p_t = p_j \text{ ELSE } p_s = p_j,\ p_t = p_i. \quad (4.6)$$

Then, we define a fixed coordinate frame u, v, w with the origin at source point as shown in Figure 4.5 to compute the relative difference between their associated normals n_s and n_t. Using this coordinate system, the difference between both neighborhoods can be expressed as a set of angular features as follows:

$$d = \|p_t - p_s\|_2, \quad (4.7)$$
$$f_\alpha = v \cdot n_t, \quad (4.8)$$
$$f_\phi = u \cdot \frac{p_t - p_s}{d}, \quad (4.9)$$
$$f_\theta = \arctan(w \cdot n_t\ ,\ u \cdot n_t). \quad (4.10)$$

The four features are a measure of the angles between the points normals and the distance vector between them. Because f_α and f_ϕ are dot products between normalized vectors, they are in fact the cosines of the angles α and ϕ between the 3D vectors, thus their value is between -1 and 1, and 0 if they are perpendicular. Similarly, θ is the arctangent of the angle that n_t forms with w if projected on the plane defined by $u = n_t$ and w, so its value is between $-\pi$ and $+\pi$, and 0 if they are parallel.

The 4-uple $< f_\alpha, f_\psi, f_\theta, d >$ is computed for each pair of points in the k-neighborhood (hence, for a total of $\frac{k(k-1)}{2}$ pairs), hence reducing the 12 ($= 4 \times 3$) values of 3D positions with normal information of any pair to only 4. The set of all quadruplets is then binned into a histogram specific for the query point. The binning process divides each feature value range into b subdivisions and counts the number of occurrences in each sub-interval. Since three out of the four features presented above are a measure of the angles between normals (or their cosines), their values can be easily normalized within the same interval. In Fig. 4.5, we depict the angles defined above.

A binning example is to divide each feature interval into the same number of equal parts, and therefore create a histogram with b^4 bins in a fully correlated space. In this space, an histogram bit increment corresponds to a point having certain values for all its 4 features. In some cases, the fourth feature d does not have an extreme significance for 2.5D data-sets, usually acquired in robotics, as the distance between neighboring points increases with the viewpoint (it is not an invariant feature). Therefore, omitting d has proven to be beneficial, and this lets us with b^3-bins histograms.

Most implementations of PFH use a histogram of b^3 bins. The PFH descriptor uses $b = 5$, which gives a histogram of 125 bins.

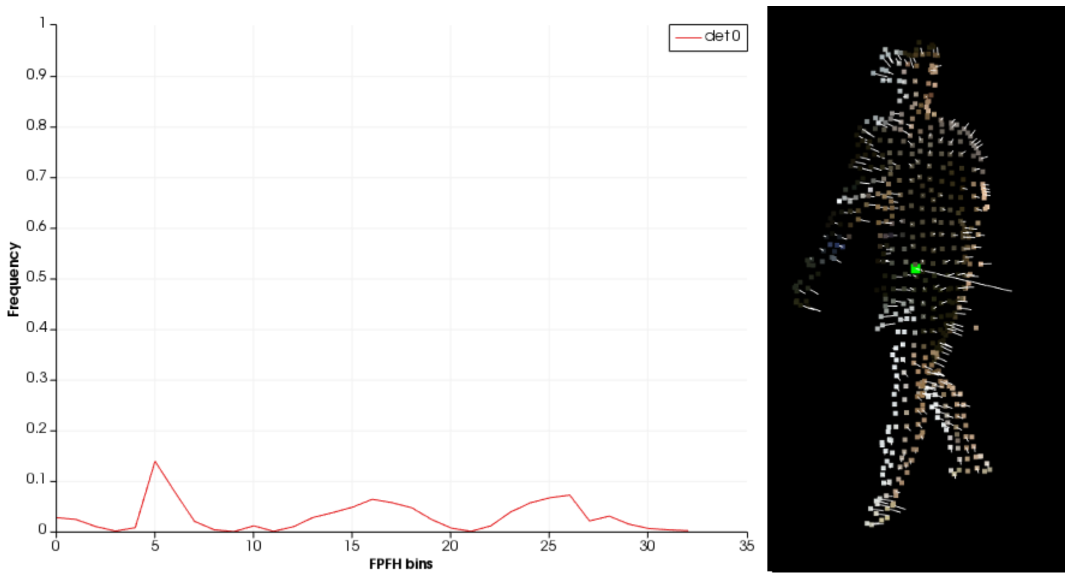


Figure 4.6: On the left, evaluation of the normalized FPFH histogram for the query point (in green) in the sample point-cloud on the right, using 33 bins to encode a fingerprint of the orientations of normals around the query point.

4.2.2 Fast Point Feature Histogram (FPFH)

The main disadvantage of using PFH lies on its complexity: theoretically, for a given point-cloud $\mathcal{P}_i$ with $n_d = \text{size}(\mathcal{P}_i)$ points and when considering all the possible pairs, the complexity is $O(n_d k^2)$, where k is the number of neighbors for each point p_i in $\mathcal{P}_i$. The computation of PFH in real-time or near real-time applications with dense neighborhoods turns into a notorious bottleneck.

A simplification of PFH, called Fast Point Feature Histograms [26], illustrated in Fig. 4.6, reduces the computational complexity to $O(n_d k)$, while keeping the discriminative qualities of PFH. To evaluate a FPFH, we follow the steps:

1. For each query point p_q, a set of tuples f_α, f_ψ, f_θ between p_q and its direct neighbors are computed as in Equation (4.7). As opposed to using a fully connected graph (all pairs), here only k pairs values are considered. Creating a histogram with these data is called the Simplified Point Feature Histogram (SPFH).

2. Then, for each point in the previous step, the k neighborhood is re-determined, and the neighboring $SPFH$ values are used to weight the final FPFH histogram of p_q as follows:

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_i}\cdot SPFH(p_i), \tag{4.11}$$

where ω_i represents a distance between the query point p_q and a neighbor point p_i in some given metric space, thus scoring the (p_q, p_i) pair, but it could just as well be selected as a different measure, if necessary. Figure 4.7 illustrates the influence region diagram for a k-neighborhood set centered at p_q.

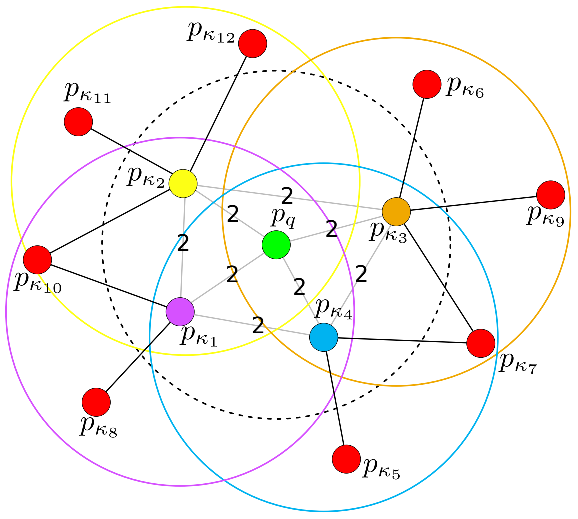


Figure 4.7: Influence region diagram for a query point p_q with connections only to its direct k neighbors (gray edges), within a 3D sphere of radius r. Each direct neighbor is connected to its own neighbors and the resulting SFPH histograms are weighted together with the histogram of the query point to form the FPFH. Connections in gray get a weight of 2 as they contribute twice as much in the final histogram.

Another important difference between PFH and FPFH is the binning, since PFH was using b^3 bins. The problem of this is that many of the bins end up being empty as this cube in 3 dimensions is a fully correlated feature space, leading to redundancy in the histogram space and not helping in differentiating histograms. To avoid this, each feature is processed on separate histograms and then concatenated in a single one. Here we use 33 bins, where the first 11 bins belong to α, the next 11 to ψ and the last 11 to θ.

Note that the classic PFH models a precisely determined surface around p_q, while FPFH includes additional point pairs outside the sphere of radius r, though at most $2r$, and due to the re-weighting scheme, FPFH recaptures some of the neighboring value pairs.

Just as PFH, Fast Feature Histograms are unaffected by the surface orientation, and they form a local descriptor for points in $\mathcal{P}_i$. Considering a person in movement, the captured geometry by this descriptor will tend to indicate surfaces to be mostly shaped as cylinders, and by itself, it may not help that much in differentiating humans, unless they have uncommon body shapes or are even carrying specific objects. To make the FPFH descriptor less local, we also evaluate an average, overall histogram as will be explained later.

4.3 Measuring color and appearance

Settling with FPFH for describing the local geometry of our targets, we still need to take into account the colors observed on these targets, which should be our best bet for differentiating detections of a same target from another one, as geometry alone won't be enough. Considering that our input data is a point-cloud $\mathcal{P}_i$ and that we need a cheap strategy in terms of computing, we will be using color histograms. Thanks to depth data combined with detections, we do not need to worry about background removal as in classic 2D histogram evaluation, for all the points inside the bounding cylinder should be already free[1] of pixels that do not belong to the target, thanks to depth data used during re-projection.

4.3.1 HSV Color space

Working directly with RGB data is usually not a good idea in terms of comparing how a color is similar to another. In the field of computer vision, color representations such as HSV and CIELAB are preferred for they are designed to make more sense of the distance when comparing two different colors. We chose the HSV color representation for its simplicity and easy calculation. HSV stands for hue (in degrees), saturation and value (both from 0 to 1, and re-scaled depending on the application), and the color transformation from RGB is performed as:

$$MAX = \max(R, G, B) \tag{4.12}$$

$$MIN = \min(R, G, B) \tag{4.13}$$

$$H = \begin{cases} 60^\circ \times \frac{G-B}{MAX-MIN} + 0^\circ & \text{if } \substack{MAX=MIN \\ MAX=R \\ G\geq R} \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 360^\circ & \text{if } \substack{MAX=R \\ G<B} \\ 60^\circ \times \frac{B-R}{MAX-MIN} + 120^\circ & \text{if } \scriptstyle MAX=G \\ 60^\circ \times \frac{R-G}{MAX-MIN} + 240^\circ & \text{if } \scriptstyle MAX=B \\ \text{undefined} & \text{otherwise} \end{cases} \tag{4.14}$$

$$S = \begin{cases} 0 & \text{if } MAX = 0 \\ 1 - \frac{MIN}{MAX} & \text{otherwise.} \end{cases} \tag{4.15}$$

$$V = MAX. \tag{4.16}$$

Using the HSV color-space, comparing a pair of color values yields a distance that is more related on how similar are the colors to human vision, which does not happen with the RGB color space, where the jump in the values are in general not proportional to the eye perception.

[1]This is true up to some extent, since depth sensors provide certain precision, when taking color and depth images, removing distortion and registering images for a perfect match of RGB and D pairs still leaves some pixels incorrectly falling on the wrong 3D position in the world, specially for pixels further from the center of the image.

4.3.2 Color Histogram

Color histograms are maybe the most classic and easiest technique used for object classification and identification in computer vision. However, they are not so reliable for such applications, as color by itself is just used as a frequency and many objects could get the same color histogram, since they do not account for how colors are physically located.

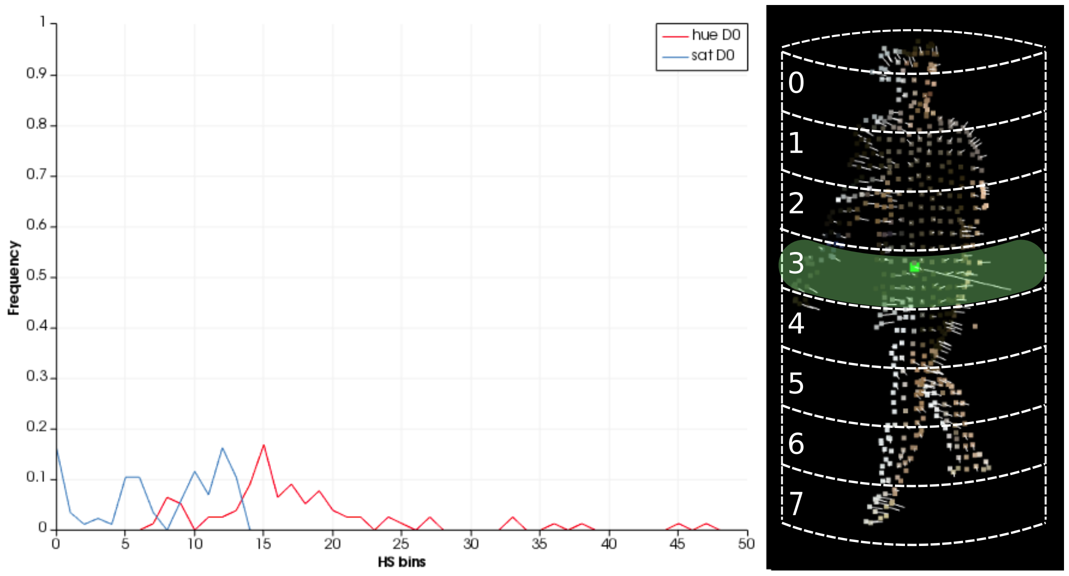


Figure 4.8: Hue and Saturation histograms using the color information stored at points that fall inside bin #3 (green highlight). The binning is done by partitioning the point-cloud $\mathcal{P}_i$ by its detection height, dividing it into 8 equally-sized bins along the z axis. For points inside each bin, the color histograms are extracted and each point FPFH histogram is averaged.

In spite of its drawbacks, a motivation for using this weak appearance measure, apart of its low computing complexity, is the advantage given by RGB-D data, as background elimination in targets is freely granted and as classification with color histograms typically assumes the background to be constant or previously removed for an efficient enough classification. We are able to use RGB data in our point-cloud, turn them to HSV values and build a histogram, though, for making it more useful, we also introduce some information about the spatial distribution of the colors by using a binning in $\mathcal{P}_i$ as will be explained below.

4.4 Histogram Comparison and Metrics

It is clear now how the geometry in point-clouds is encoded within the histograms that are built. For geometry, we have a FPFH per $p \in \mathcal{P}_i$ and we have seen that they describe only a small patch (p's neighborhood). Also, for color, we have explained how color histograms are a rather global descriptor that could fail because color positioning does not play part in making the descriptor more discriminant. Note that both types of histograms are agnostic of the target orientation and position, so we decided to build average histograms that are spatially restricted to a binning that did not require knowledge on the target actual orientation or position of its limbs. We use a horizontal binning along the z axis of the target, since human movement occurs more on the plane rather than along its height. We are able to restrict the position of the points in the target to a disk, as shown in Figure 4.8, hence dividing the point-cloud into 8 pieces: $\{\mathcal{P}_i^0, \mathcal{P}_i^1, \cdots, \mathcal{P}_i^7\}$. Each bin b will then have a $\mu-$FPFH$[\mathcal{P}_i^b]$, a H-hist$(\mathcal{P}_i^b)$ and S-hist$(\mathcal{P}_i^b)$ histogram, to encode global information of the point-cloud into 24 histograms so that we can describe the whole target detection d_i at frame t_i.

Finally, we need to define a way to evaluate how similar are two detection's point-clouds. Having them described by 24 histograms each, we need a way to measure the distance between any pair of histograms. We review different distance metrics for histograms. For simplicity we will refer to a pair of histograms $\mathbf{G}$ and $\mathbf{H}$, with bin values $[g_i]_1^b$ and $[h_i]_1^b$ in the following formulas.

The most common distances are the Manhattan (L1) and Euclidean (L2) norms:

$$\text{Manhattan(L1)} = \sum_{i=1}^{b} |g_i - h_i| \qquad \text{Euclidean(L2)} = \sqrt{\sum_{i=1}^{b} (g_i - h_i)^2}. \tag{4.17}$$

Another distance that is similar to the L2 norm, but more sensitive to differences in smaller bins, is the Jeffries-Matusita(JM) metric, also known as Hellinger distance:

$$\text{Jeffries-Matusita (JM)} = \sqrt{\sum_{i=1}^{b} (\sqrt{g_i} - \sqrt{h_i})^2}. \tag{4.18}$$

The Bhattacharyya distance is widely used in statistics to measure the statistical separability of spectral classes:

$$\text{Battacharyya (B)} = -\ln \sum_{i=1}^{b} \sqrt{g_i - h_i}. \tag{4.19}$$

Other popular measures for histogram matching in literature are Chi-Square(χ^2) divergence and the Kullback-Leibler (KL) divergence:

$$\text{Chi-Square}(\chi^2) = \sum_{i=1}^{b} \frac{(g_i - h_i)^2}{g_i + h_i} \qquad \text{KL divergence} = \sum_{i=1}^{b} (g_i - h_i) \ln \frac{g_i}{h_i}. \tag{4.20}$$

Findings from [29] indicate that the KL divergence gives good results for computing differences between histograms. To make use of the KL-divergence, denoted as $D_{KL}(\mathrm{G}, \mathrm{H})$, we first need to apply a strategy since it gets undefined when $h_i = 0$.

KL measures the divergence between probability distributions. If we think of histograms as discrete distributions then $\sum_{i=1}^{b} g_i = 1$ and $\sum_{i=1}^{b} h_i = 1$. Both P and Q have the same "outcomes" which are the binning and frequencies obtained, but some end up being 0, which could be interpreted as an "impossible" outcome. If for any g_i to be measured against a corresponding h_i, the first is possible and the second is impossible, it means that the distributions are different, hence the divergence is infinite.

To avoid these statistical interpretations to occur in our context, and since it is valid for a histogram to have many zeroes, we derive a valid probability distribution by smoothing the histograms. An easy technique we use is called absolute discounting, that gives every impossible event a small chance proportional to how many events like this need to become possible, and removing those odds from events that weren't impossible. Details can be seen on Algorithm 6.

Algorithm 6 Absolute discounting

```
Input: $G = \{g_i\}_1^b$ - discrete distribution (histogram)
       $\epsilon$ - a small constant, like 0.00001.
Output: $G'$ - smoothed distribution (histogram).
1: $G' = G$
2: $z = size(\{g_i \mid g_i = 0\}_1^b)$
3: if $z > 0$ then
4:   for all $g_i$ in $\{g_i\}_1^b$ do
5:     if $g_i = 0$ then
6:       $g'_i = \epsilon$
7:     else
8:       $g'_i = g_i - \frac{\epsilon}{z}$
9: return $G'$.
```

4.4.1 Descriptor design

For each bin in the 8-binned bounding cylinder, we calculate the average FPFH histogram, the H-histogram and the S-histogram. Our overall appearance descriptor is a set of 24 histograms that will be the numeric representation of our original point-cloud. Using the Kullback-Leibler divergence, we measure the distances between each histogram, which results in a value from 0 to infinity, where 0 indicates a total match and big numbers indicate a mismatch. This results in a set of 24 distances, which are finally weighted and averaged into a single value:

$$\begin{array}{rcl} a'_R(d_i, d_j) & = & \rho \sum_{b=0}^{7} D_{KL}(\mu - \text{FPFH}(\mathcal{P}_i^b),\ \mu - \text{FPFH}(\mathcal{P}_j^b)) + \\ & & \lambda \sum_{b=0}^{7} D_{KL}(\text{H-hist}(\mathcal{P}_i^b), \text{H-hist}(\mathcal{P}_j^b)) + \\ & & \nu \sum_{b=0}^{7} D_{KL}(\text{S-hist}(\mathcal{P}_i^b), \text{S-hist}(\mathcal{P}_j^b)) \end{array} .$$

The parameters (ρ, λ, ν) are constants that sum 1.0 for us to weight the importance of the parameters. In practice, we use the values $(0.3, 0.3, 0.3)$. Do note that a'_R takes its values in the range $[0, \infty)$. Hence, to change its range to $[1, -1]$ for using it in our RGB-D NOMT framework, we just evaluate:

$$a_R(d_i, d_j) = 2 * \exp\{-a'_R(d_i, d_j)\} - 1, \tag{4.21}$$

which we use instead of a_A.

Chapter 5

Developed Software

In this chapter, we give technical details about the implementation of the algorithms we described in the previous chapters.

5.1 Overview

A tracking system, on its simplest form and core, consists on the following elements:

- INPUT: A set of detections in time. A detection is modelled as a data structure containing an id, a time-stamp, and since we work with RGB-D data, we also include the 3D coordinates of the points inside the detected bounding-box, with their corresponding colors. This information is provided by a Detector software module, which might work with a different data structure for detections, so we might need to perform some extra pre-processing for its translation into a format compatible with the tracker.
- TRACKER: As the main component of the system, for each set of detections at a given time-stamp, it performs data association to generate the output by using the algorithms we have described in this thesis.
- OUTPUT: A set of labels assigned to the detections. This way, each detection that belongs to the same individual has the same tracked target label. This information could be used in many applications, which most commonly is the process of predicting trajectories, the analysis of group motion and the generation of subsequent actions.

All of these elements have been built to operate on top of the Robotics Operative System [25] using the C++ Programming Language. Software in the form of ROS modules are able to work independently of the detections provider and of the tracking consumer software.

5.2 About ROS

The Robotics Operative System (ROS) [25] is a collection of open-source C++ and Python implementations of the most popular robotics algorithms, libraries and tools. Its aim is to ease the task of creating complex and robust robot behaviors and platforms. Achieving such a construction is hard, even for "simple" things like a single camera acquisition module, as it needs to send the images to a computer and this data transfer by itself brings difficulties like maintaining the camera models, firmwares, networking properties and formats needed to receive and to understand the data. Also the data load, the varying frequencies at which the camera produces images and the rate at which the computer can process incoming images are variables to deal with. Now, most systems require more than one camera and involve more than one computer (they work as distributed networks). Modern systems must be capable to cope with these heterogeneous schemes and pipeline the results to perform actions such as detection, tracking, prediction and navigation.

ROS operates just like an operative system, with nodes (processes) and topics (exchange of data through ports and sockets). This design allows for realistic simulations since the nodes work in a standalone fashion and operate at their own frequencies, meaning that data consumption and generation might occur completely asynchronously. The ROS robustness comes from the possibility of running independent nodes that wait for incoming data, even if the node responsible of producing such data is down, has not been initiated yet or fails during its execution. This way, the system can keep working and can restart its modules in case of failure.

For further understanding of the concepts used in this chapter, we give the following definitions, which are important in the context of ROS:

- roscore: As the main process of ROS, it represents the kernel of the operative system, since it manages the ports and the running processes and handles the topics and data exchange through local and remote sockets. It keeps a set of global parameters and must always be running for the correct execution of the other processes.
- nodes: A term used for any executable program that runs under the control of roscore. It can consume and output data through topics.
- topic: A buffer or pool of data that is referred by this name or "topic". Nodes might consume this queue of data by a process called subscription, and might post data into it by a process called publishing. Data is formatted as a text, called a message.
- messages: It is a common name for any data structure that defines a text representation for the data, with an automatic conversion to object code in C++ or Python. It allows for an easy exchange of data between nodes and topics.
- launch: Since most of the time, we need to run multiple nodes and set up a pipeline of topics for data exchange, launch files are YAML-formatted files that give an easy way to define a set of calls, run-time parameters and variables for initializing and running a whole system. In this thesis, we have written launch files for easy testing of experiments and data-sets.

- modules & packages: A ROS code project that provides definitions of messages, executable nodes, launch files, libraries and dependencies. It can be compiled using the ROS compiler: "catkin".

5.3 Simulating RGB-D Detections

In order to evaluate our tracking algorithms, we need to process the RGB-D data within a dataset and read its ground-truth detections so that we can create repeatable experiments and evaluate numerically the performance of these algorithms.

We have built ROS utilities to process pairs of RGB and depth images, to perform the rectification of these images (i.e. to be able to unambiguously associate a depth pixel with a RGB pixel), to perform registration and point-cloud reconstruction, along with reading and displaying ground-truth detections stored in datasets in the form of image sequences and annotations.

In order to turn datasets into what the actual Kinect Sensor provides under the ROS System, we have created a ROS node that we called dataset_publisher, with its corresponding launch files for testing both the EPFL and the Spinello datasets. To come to an understanding of what is needed for the construction of the INPUT of our Tracking system, we need to publish the following topics:

- /depth_registered/points: The set of 3D points expressed in the world frame, with their associated RGB information. Corresponding with D_{t_i} at time t_i, this is the complete observed scene, with a desired level of density of points, through down-sampling. If many depth sensors are used, this should correspond to the merged views of all points now on a single coordinate system.

- /rgbd/detections: The 3D locations at which the targets were detected, also expressed in the world frame, corresponding to the set D_{t_i}. Agnostic of how detections were made, we place our bounding box using this information. Again, when many sensors are involved, the detections made in their respective viewpoints are combined to match the newly given depth images, so we can successfully obtain each d_i target cloud $\mathcal{P}_i$.

- /tf: Transforms topic. This is a special topic in ROS, which is used for defining every coordinate system in the form of a tree: a root coordinate frame would be tf_world, and every sensor has a tf_optical_frame from which the z axis is oriented as the camera's viewpoint direction. At first, every detection and re-projected points will live in the 3D world of the camera frame, and the tf tool library provides an easy way to transform coordinates from a coordinate system to another.

In order to obtain the point cloud and detections input streams, depending on how much pre-processed is the data, coming either from read data-sets or from direct sensor output, ROS provides many pre-build "nodelets" that help in turning the information to the following steps needed to get world frame data as we need it. Mainly, these are:

- image_proc. The ROS module that removes the geometrical distortion from the raw camera stream, and that is also able to apply rectification, resizing and other color transformations, if needed.
- depth_image_proc. The ROS module that provides the basic processing for depth images, such as registration, re-projection, to create the 3D RGB point clouds.
- camera_info topics. This message contains the calibration matrix data, the camera distortion model and the projection matrices. It is a key element at each step in the pre-processing of the images and is used with the detector and the tracking software.

A representation of this pipeline is shown in Figure 5.1

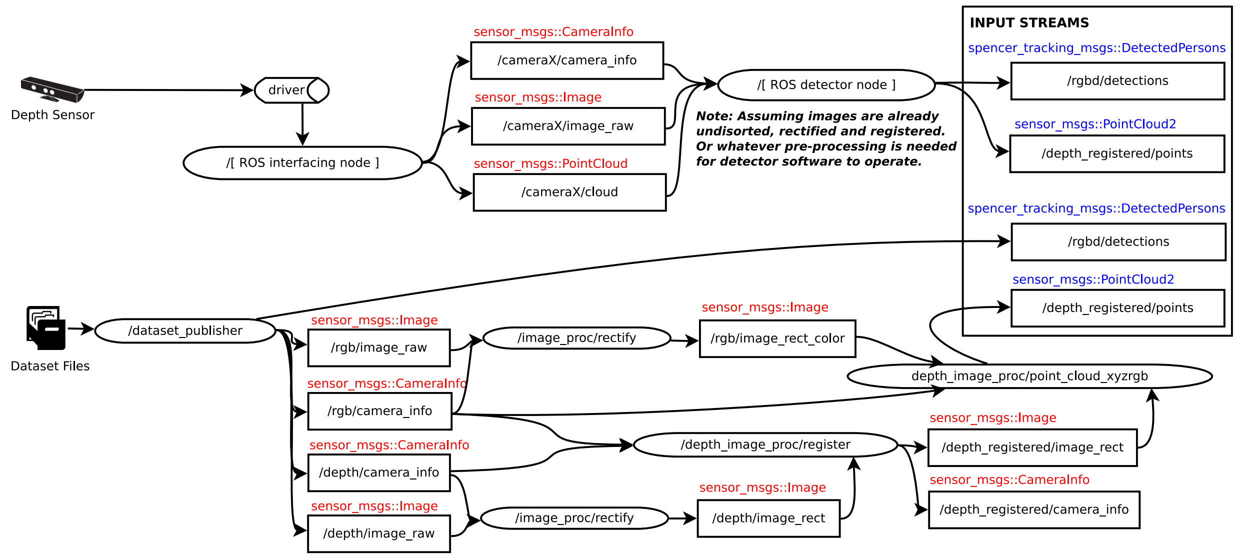


Figure 5.1: A ROS Graph with nodes as ovals and topics as rectangles. It shows two situations in which data can be collected. First, using a physical depth sensor which has its own data output formats and its corresponding software drivers. Most sensors are supported on the ROS platform through specialized nodes. Another case is when the data is read from files, with annotations on detections, used camera configurations and coordinate systems involved. Sometimes the data is stored without any kind of needed pre-processing such as the images un-distortion, rectification or registration, which are key steps for most computer vision applications.

5.4 Multiple Sensor Support

There is an ever growing need for using more than one sensor. Single sensors have limitations in their range that make it more useful to use data collected with several RGB-D sensors, each providing depth and color info, in the coordinate system around the sensor. Also, consider the fact that for each sensor, the stream of data is broadcast at certain frequencies and generates a large load of incoming data that may easily saturate the receiver host hardware. Having more than one sensor rises the need to perform aggregation of incoming data to prepare it for the Tracking software.

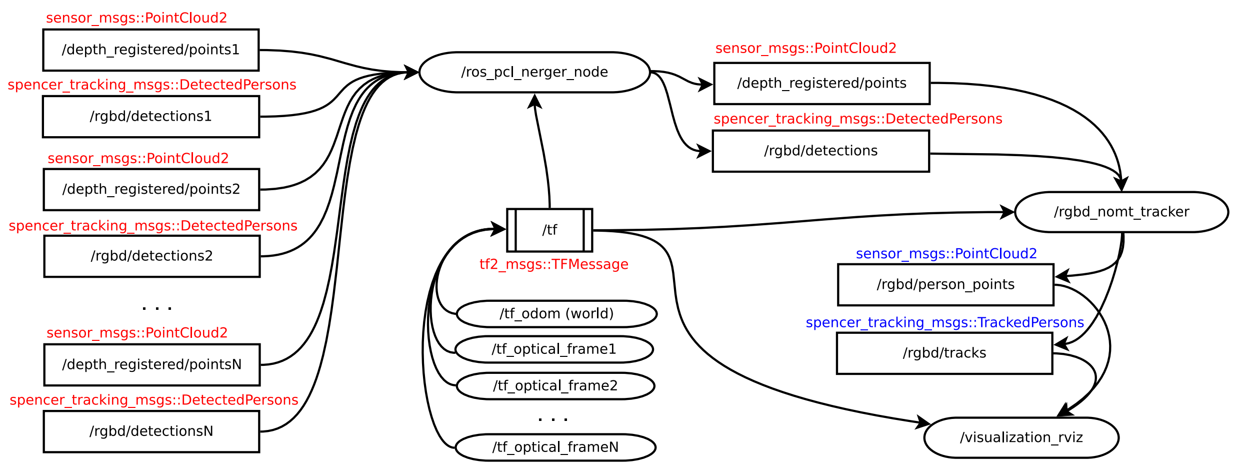


Figure 5.2: A ROS Graph with nodes as ovals and topics as rectangles, considering the input of N sensors, each providing data in terms of their optical frame and each with their own resolutions and frequencies. The ros_pcl_merger_node subscribes to each of their outputs: point clouds and detections, and by transforming them on to a common world frame, the point-cloud is sub-sampled to reduce the computing burden. Then, detections are synchronized by approximate time to match the point clouds and the positions in which persons where detected. rgbd_nomt_tracker is the tracking process that matches detections to consistent trajectories in the 3D space, outputting a special type of message that is visualized on RViz, the ROS data visualization tool.

ROS allows an easy integration of many models and brands of sensors, and a transparent reception of the input RGB-D data through ROS topics. We have built a simple ROS node that depends on the usage of the PCL Library [28] for usage with incoming point-clouds. This node down-samples and transforms points from their original sensor coordinate system to a main world frame in which the tracker operates. The ros_pcl_merger_node node requires a subscription to the corresponding camera coordinate systems in order to calculate the transformation of the points into the world frame. A depiction of this pipeline is shown on Figure 5.2. The Tracker runs here under the name rgbd_nomt_tracker and outputs its results in the form of associated trajectories and the current target point-cloud for visualization and possible further operations.

5.5 Tracking Pipeline

For inspiration on the design of our system and pipeline, we took a look at the Spencer Multi-Modal People Detection & Tracking Framework from [17]. It also contains visualization tools as plug-ins for ROS's RViz that are helpful for passing data between the ROS nodes and for visualizing the tracking experiments. Particularly, we use the following items:

- spencer_tracking_msgs. A Ros module that contains definitions of messages DetectedPerson, which contain a detection id, its score or confidence and its 3D pose, and TrackedPerson, which has a target id, flags for current status, age and 3D pose history (observed trajectory).
- spencer_tracking_rviz_plugin. It has meshes and models for the visualization of the DetectedPerson objects, in the form of bounding boxes, even if in the actual processing of the tracker,

detections are treated as bounding cylinders, as explained in Chapter 2. Also, TrackedPerson messages can be visualized as a human figure positioned on the latest position of a tracked target, and their pose history can be used to visualize trajectories.

Our tracking software operates on a sliding window and begins its execution with a set value for the window size τ, in seconds, which is used to separate incoming detections in the form of DetectedPersons messages over frames, by looking at their timestamps. It receives the first set of detections $D_{t_0}^{t_1} = \{d_1, d_2, ..., d_k\}$. The Tracker assumes that detections within this set might not have the exact same time-stamp, only that they lie in the temporal range $[t_0, t_1]$. Also, on subsequent calls or iterations, the tracker keeps receiving detection sets $D_{t_1}^{t_2}, ..., D_{t_i}^{t_j}, D_{t_j}^{t_k}$. These detection sets are stored in memory using an internal format PCLDetections that, in addition to pose, score and id, keeps a reference to the extracted point-cloud inside the bounding-cylinder and to other PCL related structures.

It should be noted that, sometimes, due to problems in the communication channels between the Detection process and the Tracking process, like network latency and bandwidth saturation, the received messages may arrive un-sorted along their time-stamp. By essence, window-based trackers can be more resistant to this situation: if messages arrive late but within the time window period, they can be still used for generating new hypothesis and can be then integrated with the final trajectory.

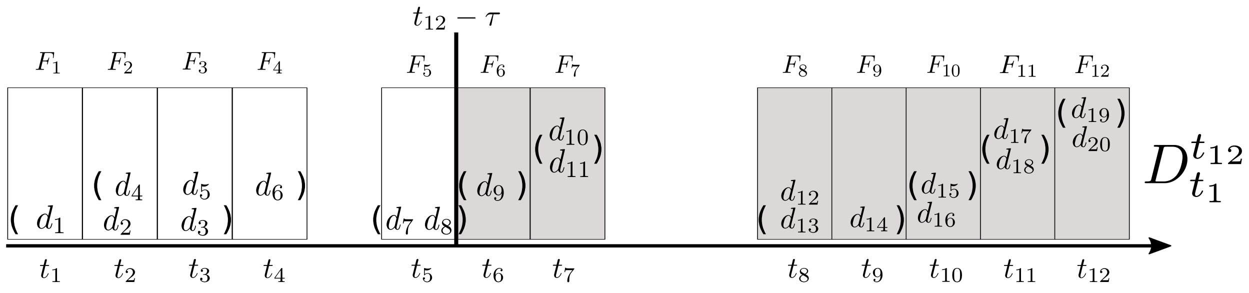

$t_{12} - \tau$
F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8 F_9 F_{10} F_{11} F_{12}
d_1 … d_{20}
$D_{t_1}^{t_{12}}$
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12}

Figure 5.3: Depiction of a possible set of the first 9 iterations, showing the incoming detections given as groups of varying size. Each message is represented by parentheses, sometimes received with older timestamps. The Tracker automatically manages frames by sorting distinct time-stamps and assigning detections with the exact same t. The Tracker also keeps the record of which frames fall inside or outside the sliding window (grayed out). By design, older detections are only taken into account by the tracking as long as they fall inside the sliding window. Any trajectory outside it is considered as final and untouchable.

The Tracker does not begin the data association process until the sliding window is full (i.e., τ seconds have passed). Once this is accomplished, the Tracker begins with Tracklet and Hypotheses generation, Data Association and lastly Target Augmentation and clean up for the next iteration.

Particularly for Data Association, as mentioned in Chapter 3, we build a graph model and perform its connected component analysis using the C++ Boost Graph Library [32]. Then, we solve the inference problem using the Joint Tree algorithm over all the components with more than one node, for which we also used the libDAI C++ library [21].

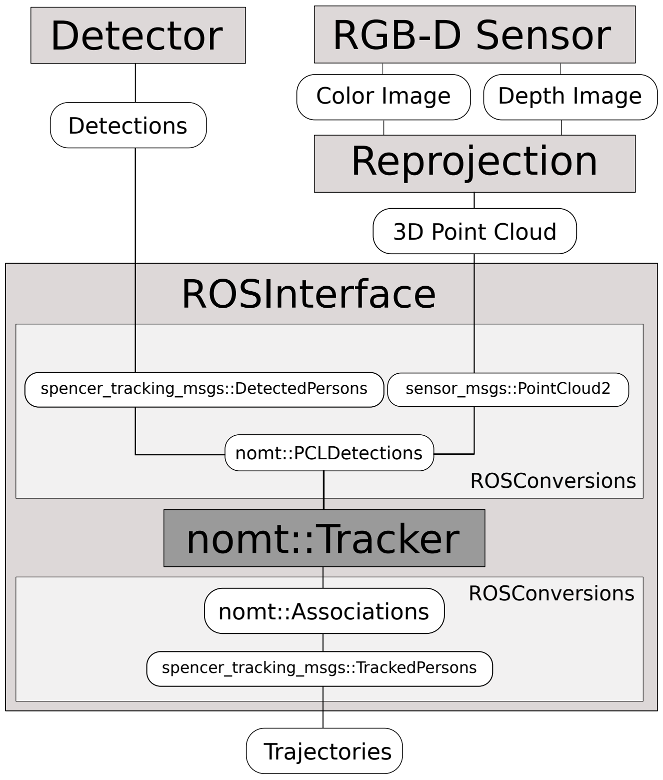


Figure 5.4: Our NOMT Library is used on top of the ROSInterface, as a node. Using the reprojections of 3D point clouds and the detections 3D positions, it translates them into instances of the Detection class, as an input. Each iteration of the Tracker outputs a group of Associations, which contains the resulting Targets labels and the observed trajectories.

5.6 The NOMT Library

We have created an easy-to-use general purpose tracking software designed as a ROS Package called nomt. It can be used either as a dependence for other ROS Packages or as a standalone shared library called libnomt.so. It includes the following classes:

- nomt::Tracker: The main object, designed using the singleton and factory programming pattern. This object is designed to be the only one responsible of creating each instance of the other mentioned classes. Each of them are stored as a list of pointers:
  - $D_{t_0}^t$ (m_tracks) is a list of pointers to each Track, from t_0 to t.
  - T^t (m_T) is the list of all Tracklets created at time t. These objects will be destroyed and recreated at every iteration, except for the Tracklets referenced by Associations.
  - H_1^t (m_H) is the list of all Hypotheses created at time t. These objects will be destroyed and recreated at every iteration.
  - A_1^M (m_A) is the list of all Associations, which are the output of the Tracking software: associations between tracks in the trajectory $[t_0, t-\tau]$ and Tracklet $[t-\tau, t]$.

This class is intended to be used, through inheritance, to define new Tracking strategies, without worrying too much about frame and detection management, sliding window control and target augmentation. This leaves developers with the sole task of implementing the Data Association step. We provide our own implementation, named RGBDTracker, as a class that performs data association as presented in this document.

- nomt::Detection: This is the representation of a detection made by the detector. For RGB-D data, it includes its 3D coordinates, the down-sampled point-cloud inside the bounding box and any other useful information that could be given by the Detector that could help to differentiate a Detection from another and measure their similarity.

- nomt::Track: This is the smallest item in the Tracking software, intended to keep a smart pointer to its associated Detection and keep a unique id. There is only one track for each Detection, i.e., this it is a one-to-one relationship. Most operations are done through this structure in order to avoid the movement of the huge load of memory that a Detection could represent. A Track represents one point with its associated info, and, when grouped in an ordered sequence, it becomes a trajectory.

- nomt::Tracklet: A Tracklet is a list of ordered Tracks, much similar to the concept of trajectory. The difference is that, whilst we call trajectory the final path that a tracked target has been walking on in the frames $[t_0, t-\tau)$, a Tracklet is still bound to modify its ordered set of Tracks since they are still in the frames $[t-\tau, t]$. This allows for the Tracker to correct mistakes. The length of the Tracklets is limited to the size (in seconds) of the sliding window. Also, a Tracklet keeps a count on how many times it is referenced by a Hypothesis.

- nomt::Hypothesis: This structure keeps a reference to a Tracklet, and this reference might be shared among other Hypotheses. The difference is that a Hypotheses might belong to different objectives or targets, meaning that it has not been decided to which Target the Tracklet will be assigned to.

- nomt::Association: A representation of the Target, that keeps the unique ID and the final trajectory of the same individual. Also, it keeps a pointer to the last associated Tracklet, completing the trajectory from t_0 to t.

5.7 ROS Interface

This is a package that allows the separation of the nomt library and the ROS framework, as it joins and converts the ROS messages and initiates ROS publishers and subscribers that send data in the right format to the Tracker class.

It comes with the following classes:

- nomt::ROSInterface: The main program or ROS executable node, that initializes and reads ROS Params from the ROS Parameter Server, creates subscriptions to the input topics and to the publishers for output topics. Then, it performs calls on the nomt::ROSConversions instance to translate ROS Messages to feed the Tracker and it gets the results for their publishing.

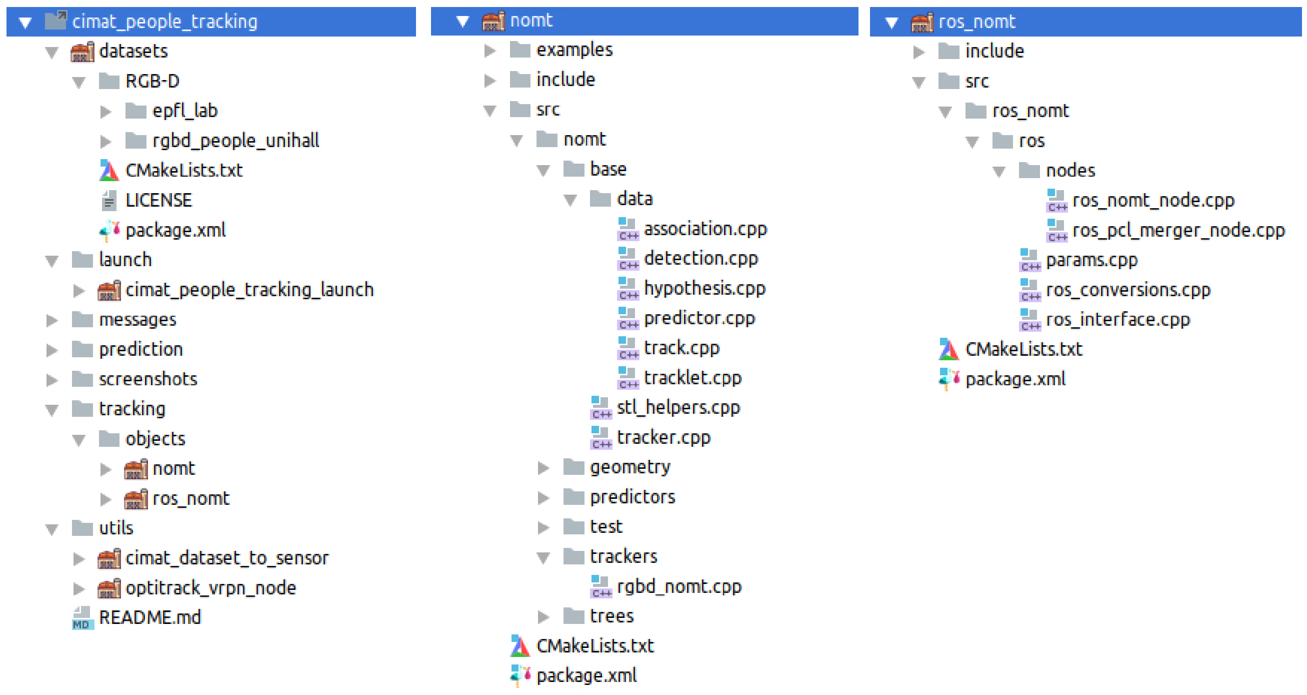
- nomt::ROSConversions: A class dedicated solely to the transformation of formats between ROS Messages and the classes of the NOMT Library. It does mainly the following transforms:
  - spencer_tracking_msgs::DetectedPersons, which are mainly the detections 3D coordinates, and sensor_msgs::PointCloud2, which are the RGB-D data collected, to nomt::-RGBDDetections for generating the input at each cycle of the Tracker instance.
  - nomt::Associations to spencer_tracking_msgs::TrackedPersons, which is the main output of the Tracker instance.
  - tf::TransformListener, to collect information about the coordinate systems world and odom. Our current implementation assumes that the data comes already represented in the world frame, as given by the ros_pcl_merger_node.
  - Many ROS Publishers for visualization on RViz like Tracklets, Downsampled Point Clouds and statistics for benchmarking.
- nomt::Params: a small utility class for handling data from the ROS Parameter Server or those given on launch files.
- nomt::Config: All the constants needed for publishers, subscribers and for the operation of the software, like topic names and queue sizes are hard-coded inside this class to provide default values that can be overridden easily through ROS Parameters on launch.

5.8 Implementing Trackers

The Tracker class was designed to be used with inheritance since it already manages the tasks of processing Detections, arranging Frames and managing common global variables in the Sliding Window. The code is well documented and structured as a ROS Module, containing the structure shown in Figure 5.5.

When creating a new Tracker, the following steps should be taken care of:

1. Extending from the Tracker class. Trackers are intended to be placed under the trackers folder, for other implementations. This grants access to factory methods needed to create Tracks, Tracklets, Hypotheses and Associations. Tracklets and Hypotheses are cleaned up automatically on each call of the processCycle method.
2. Overriding the dataAssociation method: when in this task, the programmer has access to most elements of the NOMT Library. What should be explicitly done at the end of this method is to populate the x_state variable, which indicates the final optimal selection of hypothesis $\mathbf{H}(x_m)^t$ per target A_m^{*t-1}, because the Tracker will use it to automatically perform Target Augmentation.
   - Though the Tracker does not care on how the Data Association is made, it will manage automatically any Tracklet, Hypothesis and Association created using the corresponding factory creation methods. If, for instance, you choose to solve the data association problem using an approach agnostic of the concept of Tracklet and Hypothesis, you could just create a single Tracklet, a single Hypothesis and the x_state variable indicating to pick the only hypothesis, which is the one your method came up with.



```
▼ cimat_people_tracking
    ▼ datasets
        ▼ RGB-D
            ▶ epfl_lab
            ▶ rgbd_people_unihall
          CMakeLists.txt
          LICENSE
          package.xml
    ▼ launch
        ▶ cimat_people_tracking_launch
    ▶ messages
    ▶ prediction
    ▶ screenshots
    ▼ tracking
        ▼ objects
            ▶ nomt
            ▶ ros_nomt
    ▼ utils
        ▶ cimat_dataset_to_sensor
        ▶ optitrack_vrpn_node
      README.md
```

```
▼ nomt
    ▶ examples
    ▶ include
    ▼ src
        ▼ nomt
            ▼ base
                ▼ data
                      association.cpp
                      detection.cpp
                      hypothesis.cpp
                      predictor.cpp
                      track.cpp
                      tracklet.cpp
                  stl_helpers.cpp
                  tracker.cpp
            ▶ geometry
            ▶ predictors
            ▶ test
            ▼ trackers
                  rgbd_nomt.cpp
            ▶ trees
      CMakeLists.txt
      package.xml
```

```
▼ ros_nomt
    ▶ include
    ▼ src
        ▼ ros_nomt
            ▼ ros
                ▼ nodes
                      ros_nomt_node.cpp
                      ros_pcl_merger_node.cpp
                  params.cpp
                  ros_conversions.cpp
                  ros_interface.cpp
      CMakeLists.txt
      package.xml
```

Figure 5.5: Project folder cimat_people_tracking contains every ROS module used for the NOMT Tracking System. In particular, the nomt module contains all the class definitions inside the src/nomt/base directory. It is designed to be used as a dependency or also as a standalone library (libnomt.so) for other modules, such as the ros_nomt module, which has the interfacing classes described earlier for receiving incoming data and publishing results.

3. Create the main ROS executable node like the one provided in ros_nomt_node.cpp, which instantiates a ROSInterface and the custom Tracker objects. To correctly start the cycle, it is needed to call the connect and spin methods.

5.9 Execution & Visualization

An in-depth guide on how to install ROS, create a workspace and check out code for testing cimat_people_tracking package is given. We provide documentation on a README.md at project's root folder. After the code and the environment are set up in place, we make sure the ROS master is up by running:

```
$ roscore
```

For the quick execution of an experiment, we created in the folder cimat_people_tracking_launch/-launch/datasets/ the files publish_epfl_lab.launch and publish_spinello_mensa.launch. Both files start the execution of the required nodes depicted in Figures 5.1 and 5.2, so that simulated input data is ready to be used by the RGBDTracker. Launch files can be ran by using:

```
$ roslaunch cimat_people_tracking [desired dataset].launch
```

The dataset files need to be uncompressed in the folder cimat_people_tracking/datasets/ to be found by the provided launch files, though changing to absolute paths is also possible. Files can be downloaded from either from the official web-sites or as backup files at our code repository.

A successful execution of the launch files will pop up ROS's Vizualization Tool (RViz), a GUI with visual objects already set up, including the Spencer People Tracking RViz Plugins. The latter provides a way to visualize ROS messages from the spencer_people_tracking package, which is the format used for detections and results.

Note that executing directly from datasets requires expensive operations for the processor, even for keeping a low frame-rate of 15Hz. Both launch files also run the ros_pcl_merger_node to transform point-clouds and detections into the world frame and perform down-sampling, which reduces burden for the tracker node. For testing purposes, we have also included additional launch files that operate by playing a ROS bag file, which is a compacted version of all the data streams that were generated using the previous launch files. This allows for already processed data to be used instead of live re-projection of the points.

Once the Tracker input topics /rgbd/detections and /depth_registered/points are getting data published by the pcl_merger_node, we can begin the execution of the tracker itself by using:

```
$ roslaunch cimat_people_tracking pcl_tracker_simulation.launch
```

Doing this begins the recollection of DetectedPersons and downsampled-aggregated depth images, up to τ seconds defined for the sliding window, which begins by creating and publishing Tracked-Persons, now visible on RViz. We have also included some PCL Viewers for histogram and point normal visualization that will be re-drawn on each iteration, for us to visually confirm what the Tracker is taking in for its processing.

For consumption of the Tracker results, a conversion between TrackedPersons and the desired format needs to be provided. Again, if the topic name needs to be changed from /rgbd/tracks, a remap of Topic names is always possible, for existing and further piping of the nodes.

Chapter 6

Obtained Results

In this chapter, we perform an evaluation of our tracking system. We first describe the methodology we followed for the evaluation and then we present results we obtained on standard datasets.

6.1 Evaluation Methodology

In [20], an extensive discussion is presented on how to measure and compare the results of any tracker against the ones of another. This is a very challenging task that should contemplate a lot of aspects:

- Tracking approach: the multiple parameters and considerations that are taken by the tracker itself, requiring training or not, adjusting some models and parameters depending on the situation it is getting tested under, etc. In short, everything that could lead to better or worse results. Also, tracking-by-detection methods rely on the output of an object detector, meaning that a better detector will most likely yield better tracking results. So it is important that the same precise input is given to different trackers if we want to compare them.

- Dataset composition: there is not always a consensus on what the "correct" solution should look like and, for some applications, a rough estimate is enough to release datasets for testing and some goals are achieved more easily than others. This leads to problems like erroneous and ambiguous ground truth annotations. Also, in our specific case, due to the relative novelty of RGB-D sensors, we have observed a lack of standard datasets and we did not find many trackers to compare with.

- Standards and bench-marking: Over the years, new metrics, standards, evaluation protocols and many possible benchmark strategies have been proposed. Each one has been designed to fairly compare tracking implementations, even with the already mentioned problems.

When evaluating the performance of multi-person tracker systems, the most widely used evaluation system is the standard named CLEAR MOT [3], which provides the definitions of some metrics that can measure how good is the tracking when applied to a known public dataset. These metrics are designed to make it fair to compare an implementation to another.

6.1.1 CLEAR MOT

To understand this standard, an over-generalization of what any tracker is expected to do, regardless of its approach, is done by defining the following:

- An object o_i is any real target that appears in the video sequence and that should be subject to tracking, like a ground-truth. This means that, during its lifespan, the tracker should be yielding a result for it at every moment. In our context, this corresponds to the set of all ground truth detections with their track ids.
- For every o_i, the tracker should have an hypothesis h_i (with position, and track id, which is the label given to that particular object), making a mapping (o_i, h_i). In our context, this corresponds to tracks grouped by the association set A^t.
- At any frame in time t, for every visible object, a ground-truth mapping set $M_t = \{(o_i, h_i)\}$ should be given by the tracker as a result if the tracking is perfectly performed. When the tracker yields a different mapping than the ground truth, it is said that errors have occurred.

CLEAR MOT makes the following measurements, which we have described along with our considerations in the context of our approach:

- False Positives (FP). When the tracker yields a hypothesis h_i for an o_i that is not present in the ground truth. The pairings (o_i, h_i) are done by checking whether the location of h_i is close enough to an o_i, up to a distance threshold. FP occur when no o_i is found under these criteria. Due to the design of our RGB-D NOMT which operates by just assigning labels to safe detections, without predicting or creating additional positions, we expect zero mistake of this kind. The lower this count, the better.
- False Negatives (FN). Also called misses, it corresponds to cases when the tracker yields no h_i for an o_i present in the ground truth data, i.e. when it returned a mapping $(o_i, \emptyset)$. In our tracking strategy, we may get FNs on detections as long as they remain inside the sliding window and have not yet been associated with clean tracks outside the sliding window. Once a trajectory gets past the sliding window, the tracker associates all tracks o_i that were false negatives to some hypothesis h_i, fixing the mistake and getting the FN count down.
- Identity Switches (IDS). When the tracker sees the same o_i in consecutive frames, but changes its identity from h_i to h_j, indicating that the identity was discarded in favor of another. This could happen immediately in consecutive frames or could also occur when there is a separation in time provoked by false positives and negatives.
- Mostly Tracked (MT). The ratio of trajectories that have been given an hypothesis h_i for at least 80% of their objects o_i in their lifespan. For instance, if a sequence of detections has 100 ground truth detections for o_i, and 80 of them get an hypothesis, regardless of the identity switches, it is considered to be mostly tracked. Hence $MT = \frac{\#\text{mostly tracked trajectories}}{\#\text{ground truth trajectories}}$. The higher, the better.
- Mostly Lost (ML). The ratio of trajectories that have been given an hypothesis h_i for at most 20% of their objects o_i in their lifespan. For instance, if a sequence of detections has

100 ground truth detections for o_i, and just 20 of them get an hypothesis, regardless of the identity switches, it is considered to be mostly lost. Hence $ML = \frac{\#\text{mostly lost trajectories}}{\#\text{ground truth trajectories}}$. The lower, the better.

- Fragmentation (FM) is a count of how many times the tracking got interrupted, breaking a smooth trajectory or sequence of tracked objects into fragments. This is a count of each time the object o_i went from being tracked to un-tracked, regardless of it being assigned the same or another identity. In our system, this will just be the number of times a mapping went from (o_i, h_i) to $(o_i, \emptyset)$ and then back to (o_i, h_j), no matter if the h_i is the same or not as h_j. The lower, the better.

- Multiple Object Tracking Accuracy (MOTA). It combines false positives (FP), false negatives (FN) and identity switches (IDS):

$$\text{MOTA} = 1 - \frac{\sum_t (\text{FP}_t + \text{FN}_t + \text{IDS}_t)}{\sum_t \text{GT}_t} \tag{6.1}$$

  where t indicates a particular moment in time since the tracking began and $\sum_t \text{GT}_t$ is the total number of objects present in all the frames. The best score is MOTA $= 1$, meaning that no mistake happens. However, MOTA could get negative values if there are more errors than there are ground truth objects.

- Multiple Object Tracking Precision (MOTP). This measures the total error between the estimated positions for the matched object-hypothesis pairs over all the frames, and averages it by the total number of matches. A perfect score is a MOTP $= 0$, meaning that the ground truth is obtained precisely all the time. This value is dependent on the scale in the data. For instance, it could measure the distance between 2D bounding box detections in pixels or centimetres. In our context, since we work with point-clouds, we are able to find the centroid of cloud $\mathcal{P}_i$ and use it instead of the original position given by the dataset. We believe that this is a better and smoother position of the target for its usage in tracking, but still we make the search of points inside the bounding-cylinders, using a radio of 0.3 meters from the ground truth position, hence we expect our MOTP ≤ 0.3 at all times.

As we have mentioned it, the usage of these metrics allows to have a fair comparison of a tracking technique to another. However, the MOTChallenge, from which these metrics originated, and which is the group in charge of defining which data-sets are to be used by anyone interested in comparing its method with every other contestant, has yet not given a place for RGB-D trackers to prove their capabilities. At MotChallenge's 3D MOT 2015, they have provided data-sets with camera calibration parameters to allow for 3D tracking, but still using RGB images only, meaning that there is no way to exploit the full scene depth information as we require. We will still use these metrics to have a way to compare two tracking strategies based on different inference processes, the first one using our proposed energy model (3.46), referred to as NOMT+$\boldsymbol{\Omega}$ and the second one being Choi's vanilla NOMT model (3.29) adapted for RGB-D. This allows us to observe the effects of considering an occlusion model in the evaluation.

6.2 Tested datasets

Needing RGB-D data with annotated ground-truth detections, we found a couple of interesting data-sets, both with particular data structures and circumstances that will help us get an idea of the effects of our proposed occlusion model and also observe the strengths and weaknesses of NOMT's approach.

6.2.1 EPFL-LAB

A first dataset we have used is the EPFL-LAB dataset, presented in [1]. It contains around 1000 pairs of color and depth images with 3000 annotated people instances, both through "camera" coordinates and 2D bounding-boxes in each frame. In this dataset, there are at most 4 persons walking in front of a single sensor, with occlusions that last in general less than a couple of seconds. The difficulty lies on the agents trajectories, which are far from linear, as they tend to make circular and zigzagging movements, which make predictions less accurate.

Be noted that the EPFL-LAB dataset was originally designed for testing detector software, meaning that it is rather unused for tracking purposes and that no MOTA results are available for comparison. Yet, it resulted in a good challenge for evaluating the robustness of our proposed model, since many occlusions occur and since these 4 targets remain close to the sensor and performing motions rather difficult to predict. We observe situations such as:

- Annotations are given in a world coordinate system and very few of them are done incorrectly, meaning that most detections are reliable and it is easy to filter out detections: when extracting a $\mathcal{P}_i$ from d_i, we obtain an empty cloud, then the annotation or ground-truth detection is discarded. Also, we only work with detections that have a minimum number of points, just to obtain a reliable normal estimation and hence a descriptor of what is observable in the detection.
- On playback, we have one target disappearing from the camera FOV and later returning from a different entry point. We expect the tracker to be able to re-identify the same target.
- Another pair of targets move a little bit more around the same area, but they get very close and perform circular movements. This gives a challenge for the tracker as they get occluded in this motion and also have their point-clouds mixed, since the targets hold hands and hug.
- One target remains behind the other aforementioned targets, but shares a rather similar clothing and appearance with respect to another objective. This means that identity switches here have a considerable chance of occurrence.
- Finally, since all the targets remain close and inside the sensor FOV, the depth measurements are very accurate, so we expect little to no incorrect point-cloud extractions from the detections world coordinates.

Figure 6.1a presents ground-truth trajectories of these four targets, with the original label shown as a number in white, indicating the very last position in the playback.

6.2.2 SPINELLO-UNIHALL

Another tested dataset was Spinello's [19, 34], which contains more than 3000 RGB-D frames acquired in a university hall from 3 vertically-mounted Kinect v2 sensors. This data-set contains mostly upright walking and standing persons, seen from different orientations and with different levels of occlusions. Also, since we have three sensors, the input processing burden and the point-cloud + detection synchronization is challenging by itself, as tackled in Section 5.4. In this data-set, we have the following situations:

- A limitation is that only 2D bounding-box annotations are provided, making it necessary to re-project points in 3D and find the actual coordinates of the persons in the camera frame of each Kinect sensor, which in turn renders a lot of noisy detections[1].
- Bad transformations of 2D bounding-boxes using depth occur often when the objective is too far away from the sensors. This happens particularly a lot on the Kinect that is facing the entrance of the hall: depth that is perceived on more than 5 meters away is quadratically less accurate, meaning that the depth is not detailed enough for geometry analysis.
- Also, we do not obtain so many points from 3D coordinates at around 6 meters or more, due to the weakness of the Kinect's measuring when exposed to strong sources of lights, affecting its infra-red sensor. We expect a lot of mistakes here in the identity, since the effect of having strong light behind opaque objects is that the colors are poorly perceived as they all appear darkened.
- Most trajectories in this data-set are easy to fit, but many detections are heavily occluded by other targets, and for longer intervals of time. we expect good tracking results for detections within the Kinect's optimal range of 0.5 meters to 5 meters. Detections further from 7 meters will be discarded as they are too noisy for point-cloud extraction from the bounding-box.

Filtered ground-truth data that will be used can be seen in Figure 6.2a, where white labels indicate the last position of the target owning the trajectory. Just like in the previous dataset, this filtering is done only for comparative purposes and with the intention of keeping the tracking process easy to interpret, both mistakes and successes.

6.3 Experimental results

Using the software presented in Chapter 5, we are able to play both data-sets stored in a ROS Bag file, to reduce the burden of the processor. As we have mentioned it, the sole task of making re-projection of dense point-clouds, applying down-sampling and merging and transforming all sensors into a common world frame is expensive. Playback from the ROS bag file publishes ROS topics for synchronized pairs of point-cloud and corresponding detections in the world coordinate system. These topics are consumed by the tracking node and put inside a queue or cache of messages, in order to permit the capture of all the detections and to extract the corresponding point-clouds as

[1]We do not make a deep analysis of any detection d_i that is partially visible, hence if there is another object d_j that is close, taking this depth may move detection d_i on top of d_j.

Tracking Benchmark										
Dataset	Model	MOTA↑	MOTP↑	MT↑	ML↓	FP↓	FN↓	IDS↓	FM↓	Hz↑
EPFL-LAB	NOMT	96%	0.2147	100%	0%	0	1	31	32	5
	NOMT+Ω	99%	0.2143	100%	0%	0	2	2	4	5
SPINELLO	NOMT	88.8%	0.1294	89.2%	3.5%	0	60	62	122	5
	NOMT+Ω	93.5%	0.1294	96.4%	3.5%	0	28	54	82	5

Table 6.1: CLEAR MOT Metrics obtained using our implementation of NOMT using RGB-D adapted model and our extended model for occlusions NOMT+Ω, using a shared code for fairness. The synchronized input of point-cloud with their corresponding detections is being published at around 5Hz. The tracker manages to process all the frames and output its solutions also at 5Hz. On average, 1000 ± 100 detections are received and processed during the full tracking experiment repetition, allowing to evaluate these CLEAR MOT average results. The upper arrow indicates that higher values are better. The lower arrow indicates that lower values are better.

far as possible.[2]

The experiments here consist of detections getting published at a frame-rate of 5Hz with their corresponding down-sampled point-clouds, paired by timestamp. We concede that higher frame-rates would be more interesting to test the near-online claim right now, but since we want to be able to catch as many ground-truth detections as possible on all the tests, we allow the tracker node to solve and catch the incoming data in the best possible way.

Experiments were done with a low-tier laptop with an old Intel Core i3-3110M and 8GB of RAM. Using a sliding window of $\tau = 1$ second and the default parameters as mentioned in Chapter 2 for NOMT. The tracker successfully solves the problem at the same frame-rate as the incoming data. We could go as fast as 30Hz, though a different ROS Bag file should be recorded to make the tracking go at that speed (for a correct ROS timestamp management).

Table 6.1 presents our evaluations of CLEAR MOT metrics at the end of all cycles needed to process the full sequences. We make the following observations:

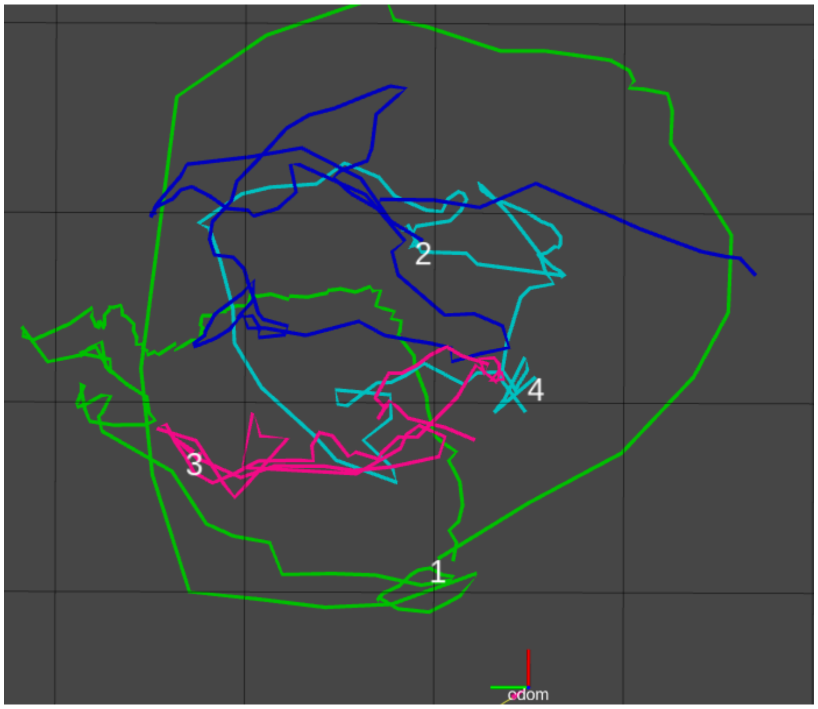
- MOTA indicates that almost every detection was given a correct label when compared to the total amount of detections that were processed. The accounted mistakes were only false positives and identity switches, which, depending on the application, could still lead to problems with decision making stages.
- MOTP indicates an average separation from the ground-truth 3D coordinates of just 21cm at EPFL-LAB and 12cm at SPINELLO-UNIHALL data-sets, which, as expected, is the average bias from the point-cloud centroid to the detections bounding-cylinder center.

[2]ROS does its best to maintain a frame-rate and frequency in which the data is passed to the tracking node. However, diverse priorities in the host processor could cause some messages not getting sent by the ROS Bag Playback, meaning that sometimes we work with more or less than the real amount of detections inside the data-set. This is good in the sense that real systems will behave like this, but it makes it difficult to repeat the same experiments exactly.

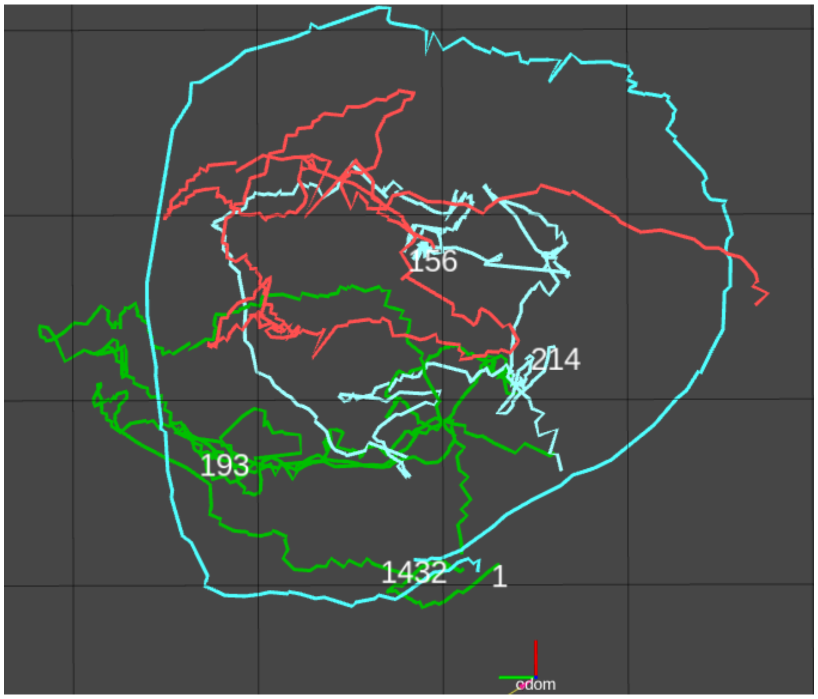
- As we expected, the false positives indicator is always 0, as our tracker does not propose additional hypothesis from those that correspond to a given target, and every hypotheses h_i stays near to some o_i with a very high precision, as reflected by MOTP.
- In both cases, MT and ML are really good and indicate that the targets were still processed for more than 80% of the total number of detections provided. Since we count so few misses, these indicators support what is observed.
- The comparison of the NOMT and NOMT+Ω models for the EPFL-LAB dataset shows a huge reduction on Identity Switches, which supports our claim that in an environment where targets get often occluded, we should allow non-smooth movement in the scene as long as the similarity indicates that it is the same target. Figure 6.1b shows the output trajectories when using the RGB-D NOMT+Ω model. We can visually appreciate a very fair match, so we did observe trajectories as they were happening, but we failed at maintaining target 1 identity right at the moment when he left the scene, changing its label to 1432 and color to cyan.
- In the SPINELLO-UNIHALL data-set, we made a considerable reduction on the false negatives count, meaning that less detections were unintentionally discarded, having many solid trajectories as shown in Figure 6.3. In our tracking system, we identify that most misses occur specifically when the identity of the target is not maintained right after the sliding window passes the trajectory: if a group of detections turn into an association with just one or two clean tracks (i.e. detections past $t - \tau$) they will still not have enough time of existence to overtake the Trackers decision that they are rather a different target (since new information arrived and the similarity was not good enough). The tracker has too few detections to measure the previous appearance and the trajectory history as can be seen in Figure 6.4. Other times, when the target has a good history and the latest detections begin to drop in quality as it goes away, identity switches also occur, as seen in Figure 6.5.

  Observing the output trajectories from our RGB-D NOMT+Ω model in Figure 6.2b, we note that some detections on the top right area ended up getting assigned just once an identity that immediately got discarded since the closer the target got to the sensor, the point-cloud data was clearer, and the appearance measures in the model indicated that the previously observed detections were not similar enough to the closer detections, so the tracker decided to start a new trajectory. This is a problem related to the drastic changes in the depth cloud when a far target approaches the sensor.

  Both problems happen much less for detections closer to the sensors inside their optimal depth ranges. Though mostly all the trajectories are tracked, the system does not reduce the total number of identity switches, as seen on Table 6.1, due to the quality of detections in the 3D plane, rendering the sliding window weak to trajectories without enough clear tracks or those cases the target still in sight moves away from the optimal ranges.

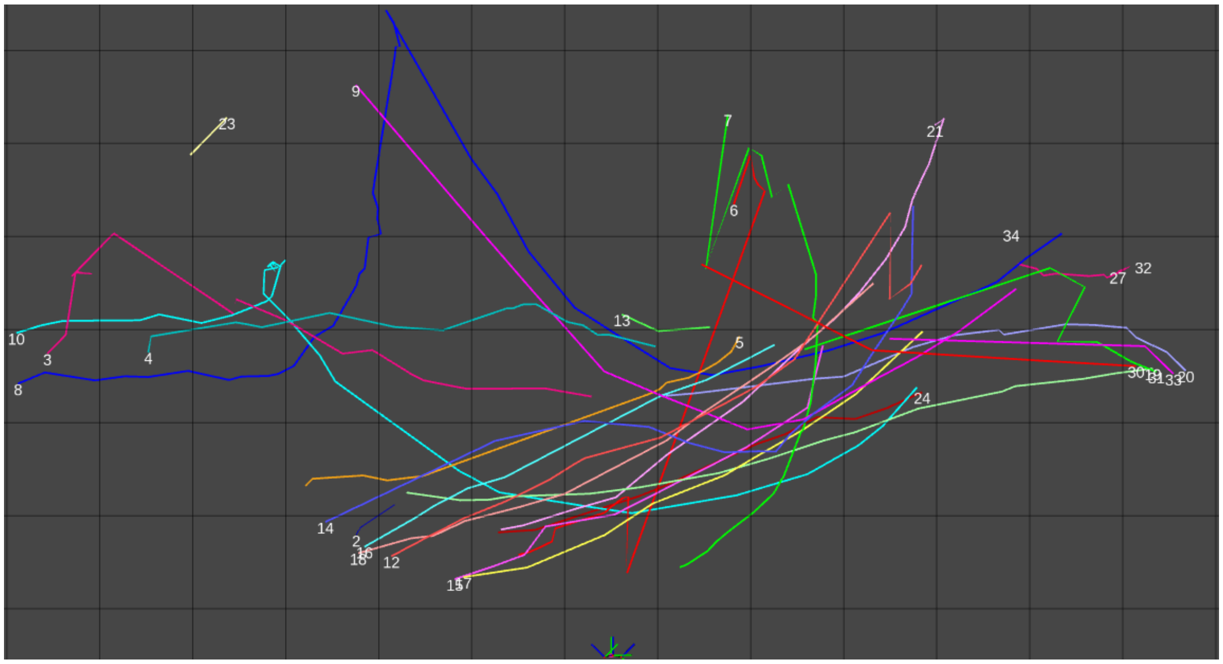


(a) Ground truth trajectories of targets 1, 2, 3 and 4 from direct 3D annotations.



(b) Resulting trajectories obtained by the RGB-D NOMT+Ω approach.

Figure 6.1: EPFL-LAB dataset: This data-set's challenge lies on the difficulty to predict direction changes for the objectives, as we can observe many circular movements, many short and long occlusions with very sudden changes in direction. In the classic NOMT, this makes it difficult for the Tracklet generator to link a decent Hypothesis, hence incurring in the creation of new targets, losing the original identities. We note that NOMT had in average 30 identity switches out of many executions.


(a) re-projected 2D ground-truth annotations into 3D trajectories.

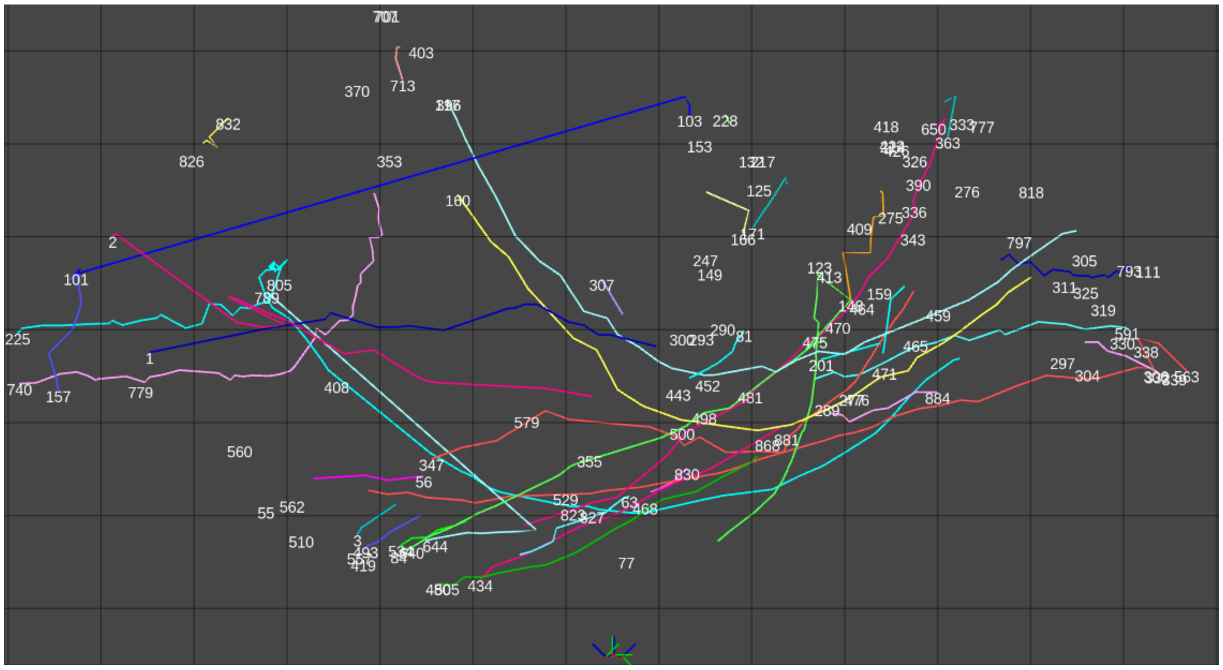
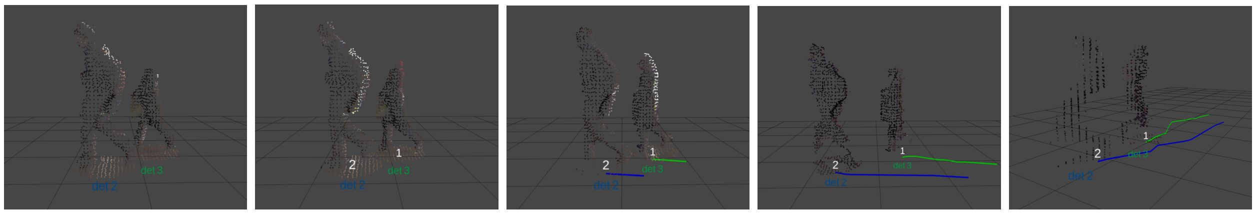

(b) RGB-D NOMT+Ω Tracker's output trajectories.

Figure 6.2: SPINELLO-UNIHALL dataset: Resulting trajectories that were tracked by RGB-D NOMT with ground-truth projected detections. Note that only trajectories with tracks in the trusted depth-range of the Kinect sensors are considered. Most trajectories end up looking smoothed out thanks to the use of point-cloud centroids instead of the bounding-box projected location. Straight lines indicate the lack of detections for short time lapses due to occlusions with other targets. There are lots of identity switches still occurring due to the low quality from the sensor further of the 5 meter mark, specially on third Kinect located at the right, where targets get shadowed by lots of background light and where max-depth is reached.



(a) Not tracked, as all detections are behind $t - \tau$

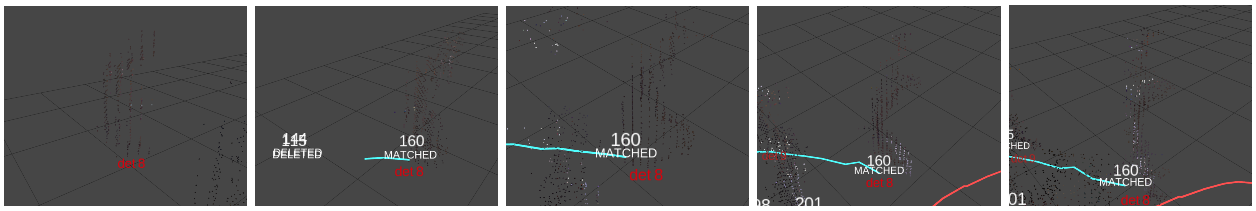
(b) Past $t-\tau$, an identity is given, and targets are tracked.

(c) Loose and clean tracks are shown as a colored line.

(d) History of positions are included in the association.

(e) Tracking is maintained in spite of loss in the cloud quality.

Figure 6.3: On trajectories 1 and 2 of SPINELLO UNIHALL data-set, the detections lie inside the optimal depth ranges of Kinect sensor. Good color and depth registration, along with a good cloud reconstruction, allow for a great consistency in the similarity measures between detections. We observe at (e) how, as they go further, the reconstruction is still good enough to perceive a person with a big backpack. However, a level-curve-like segmentation occurs parallel to the camera view-port due to the infra-red scattering around the 5 meter mark.



(a) The first time the target appears, the furthest from the sensor, a bad point cloud is formed.

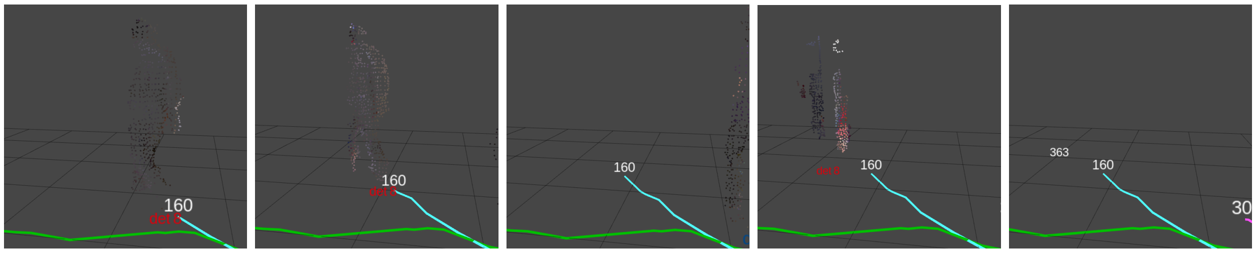
(b) Tracker fails to maintain identity twice due to bad cloud similarity inside the window.

(c) Identity is now well maintained as the target approaches the sensor.

(d) Denser clouds along the camera view-port axis. Better geometry extraction.

(e) Sometimes large jumps occur due to lack of detections in short periods of time.

Figure 6.4: In the SPINELLO UNIHALL data-set, we have ground-truth detection 8 coming from the hall entrance where the right Kinect is observing. Light sources and target 8 being too far to make good enough reconstructions, we show the mentioned effect: many identity switches may occur on these targets until they get close enough to the sensor, since we require RGB-D information to measure similarity and since the 3D detections positions can only be obtained through re-projection, as the original data-set annotations are in the 2D images.



(a) Target begins to leave the optimal depth ranges.

(b) Tracking is good until the last good detection arrives.

(c) Detector failed to give a detection at this frame.

(d) Target appears again, but gives low similarity.

(e) Identity switch given to disappeared target.

Figure 6.5: Again in SPINELLO UNIHALL, the same ground-truth detection 8 is leaving the scene from the stairs area, on the middle Kinect. The tracking is good until the detection appears too far away within the range and the last detection of a partially observable target ends up reconstructing a wall, as RGB and depth registration were too bad.

Chapter 7

Future Work and Conclusions

7.1 Future Work

As we have explained it in the previous chapters about the techniques and the motivations for their use, we have tried to stick to defining inference processes and target similarity measures that have a low cost on the processor. If we consider an upgrade of hardware in order to have room for more expensive techniques, here is a list of improvements that could be done to make the current work even better:

- Coupling the detector mechanism with the tracker. It is clear that the tracking requires a way to measure the similarity between detections and that it turns out to be a process of further reducing the point-cloud data to smaller packets of information, in this case, histogram distances per pair of detections. Building histograms requires passing on every point in the cloud and in its neighborhood. Yet, on the first place, to get this person point-cloud, we have required the detector to tell us its position in the 3D space, and to give us the dense point-cloud of the subscene inside. Opportunities of tuning the whole tracking-by-detection pipeline lie in the possibility of parsing and evaluating all the knowledge on detected targets in the fewest pixel-passes and without requiring additional ROS message passes, which translate to bulk data to be bottle-necked on port-to-port communication.

- Re-design a multi-thread version of RGB-D NOMT. In our current implementation, only some processes are paralalellized, particularly, some point-cloud pre-implemented methods such as the normal estimation. The original Choi's NOMT software is said to take advantage of multi-threads, as each target could be evaluated in its own thread, solving data-association by giving each A_m^{t-1} its optimal H_m^{t-1}. It also could benefit from having the detector coupled with tracking, since each detection could begin with its association once the descriptors are available for their usage by the tracker.

- Mobile design. Fixed camera systems are very useful. However, mobile cameras could allow for a localization system to be consumed by the ROS platform and this could theoretically be done transparently without further changes on the tracking software, allowing a robot to take advantage of the moving targets. Of course, if we consider that now they are prone to be lost from sight by the robot movement once it turns or gets past an individual, just like humans do, this is the minimal we need to just move around in a crowded place.

- Upgrading the tracking strategy. Along with neural-network-based-detectors, it is foreseen a rise of tracking techniques based on deep neural networks techniques. Their promising advances could lead to top-notch intelligent systems that make the most of cheaper GPU devices.

- Dealing with the model weaknesses. As we have seen during the experiments, when depth information is not available for re-projection, the detections end up discarded and this introduces a lot of noise for tracklet and hypotheses generation. This translates into highly fragmented associations due to identity switches. A re-identification step for new target decisions or some short-term training in the tracked objectives appearance could alleviate this problem. A reduction of the number of manually-tuned-parameters involved in the current model should also be done by upgrading particular steps in the tracking process for attaining more robustness. Also, the current model depends a lot on similarity measures, and we still could end up having situations when both the geometry and color histograms indicate high similarity between detections that could not belong to the same target. The introduction of heavier techniques like body-part and facial recognition could make it more robust in most cases, whilst compromising real-time performance. Even today, it is hard to point out to what extent a person is similar enough to another to make a fair decision on how "distant" are they in terms of appearance.

7.2 Conclusions

Most human activities are dependent in their ability to see and keep track of their workplace, team and other entities around them. As humans, we are limited in what we can perceive and account for decision making: observation, prediction and identification. Industries around the globe make use of specialized hardware, reducing intervention, personnel, costs and failures through automated computers and robots, hence the importance of intelligent camera systems, since most industrial and critical tasks nowadays are focused on detecting static and "simple" objects under controlled environments for simpler decisions to be made, fast and repeatedly.

The increasing interest in tracking-by-detection schemes for humans tracking gives a chance for better business strategies to be designed and exploited. With the introduction of cheap and powerful cameras and faster wireless coverage, we have been able to observe, react, communicate and interact faster, opening doors for new paradigms and a world of applications that can be created to make our lives better.

We have presented an updated version of an inference-based tracking approach that now takes advantage of 3D information for considering the possibility of occlusions between targets and that uses a target matching strategy based on a simple point-cloud and color descriptor. This strategy allows to attain a good performance and uses the best practices available for comparing descriptors, maintaining a near-online speed in spite of the processing overhead caused by the sole usage of dense multi-source RGB-D information. A modular separation of both tasks has been implemented thanks to the Robotics Operative System platform.

We empirically obtained indicators that supported our claim of improved performance from the vanilla NOMT tracker. We have also described the problem of associating and tracking a set of targets in a RGB-D video sequence, which we believe will become a more popular paradigm, as RGB-D sensors may replace plain cameras in future hand-held devices, smart phones and even, industrial security systems.

Along with the RGB-D NOMT implementation, the code provided allows for easier development of new RGB-D based tracking strategies, since all the problems associated to varying frequencies among the sensors and data synchronization, time and frame management are covered up by the NOMT library plus the ROS interfacing and visualization utilities.

Our results are based on sole ground-truth detections and considering a static camera system. However, mobile sensors paired with localization systems could also use our presented tracking system, since it assumes that all detections are represented on a fixed world coordinate system. Plans for a mobile design are on the works.

Bibliography

[1] Timur Bagautdinov, Francois Fleuret, and Pascal Fua. Probability occupancy maps for occluded depth images. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2829–2837, 2015.

[2] Jilliam María Díaz Barros, Frederic Garcia, and Désiré Sidibé. Real-time human pose estimation from body-scanned point clouds. In VISAPP (1), pages 553–560, 2015.

[3] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. Journal on Image and Video Processing, 2008:1, 2008.

[4] Adel Bibi, Tianzhu Zhang, and Bernard Ghanem. 3d part-based sparse tracker with automatic synchronization and registration. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1439–1448, 2016.

[5] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. arXiv preprint arXiv:1611.08050, 2016.

[6] Wongun Choi. Near-online multi-target tracking with aggregated local flow descriptor. In Proc. of Int. Conf. on Computer Vision (ICCV), December 2015.

[7] Xiao Chu, Wei Yang, Wanli Ouyang, Cheng Ma, Alan L Yuille, and Xiaogang Wang. Multi-context attention for human pose estimation. arXiv preprint arXiv:1702.07432, 1(2), 2017.

[8] Joao Paulo Silva do Monte Lima and Veronica Teichrieb. An efficient global point cloud descriptor for object recognition and pose estimation. In Graphics, Patterns and Images (SIBGRAPI), 2016 29th SIBGRAPI Conference on, pages 56–63. IEEE, 2016.

[9] Ke Gong, Xiaodan Liang, Dongyu Zhang, Xiaohui Shen, and Liang Lin. Look into person: Self-supervised structure-sensitive learning and a new benchmark for human parsing. In CVPR, volume 2, page 6, 2017.

[10] Domingo Iván Rodríguez González and Jean-Bernard Hayet. Fast human detection in rgb-d images with progressive svm-classification. In Pacific-Rim Symposium on Image and Video Technology, pages 337–348. Springer, 2013.

[11] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. Communications of the ACM, 16(6):372–378, 1973.

[12] Eldar Insafutdinov, Mykhaylo Andriluka, Leonid Pishchulin, Siyu Tang, Evgeny Levinkov, Bjoern Andres, and Bernt Schiele. Arttrack: Articulated multi-person tracking in the wild. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), volume 4327. IEEE, 2017.

[13] Umar Iqbal, Anton Milan, and Juergen Gall. Posetrack: Joint multi-person pose estimation and tracking. arXiv preprint arXiv:1611.07727, 2016.

[14] Omid Hosseini Jafari and Michael Ying Yang. Real-time rgb-d based template matching pedestrian detection. In Robotics and Automation (ICRA), 2016 IEEE International Conference on, pages 5520–5527. IEEE, 2016.

[15] Daphne Koller and Nir Friedman. Probabilistic graphical models: principles and techniques. MIT press, 2009.

[16] Mude Lin, Liang Lin, Xiaodan Liang, Keze Wang, and Hui Cheng. Recurrent 3d pose sequence machines. In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pages 5543–5552. IEEE, 2017.

[17] Timm Linder, Stefan Breuers, Bastian Leibe, and Kai O Arras. On multi-modal people tracking from mobile platforms in very crowded and dynamic environments. In Robotics and Automation (ICRA), 2016 IEEE International Conference on, pages 5512–5519. IEEE, 2016.

[18] Xiao Liu, Ling Mei, Dakun Yang, Jianhuang Lai, and Xiaohua Xie. Feature visualization based stacked convolutional neural network for human body detection in a depth image. In Chinese Conference on Pattern Recognition and Computer Vision (PRCV), pages 87–98. Springer, 2018.

[19] Matthias Luber, Luciano Spinello, and Kai O Arras. People tracking in rgb-d data with on-line boosted target models. In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pages 3844–3849. IEEE, 2011.

[20] Anton Milan, Konrad Schindler, and Stefan Roth. Challenges of ground truth evaluation of multi-target tracking. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 735–742, 2013.

[21] Joris M. Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. Journal of Machine Learning Research, 11:2169–2173, August 2010.

[22] Matteo Munaro, Filippo Basso, and Emanuele Menegatti. Tracking people within groups with rgb-d data. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 2101–2107. IEEE, 2012.

[23] Erickson R Nascimento, Gabriel L Oliveira, Mario FM Campos, Antônio W Vieira, and William Robson Schwartz. Brand: A robust appearance and depth descriptor for rgb-d images. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 1720–1726. IEEE, 2012.

[24] Tanguy Ophoff, Kristof Van Beeck, Toon Goedemé, and KU EAVISE-Campus De Nayer. Improving real-time pedestrian detectors with rgb+ depth fusion.

[25] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In ICRA Workshop on Open Source Software, 2009.

[26] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pages 3212–3217. Citeseer, 2009.

[27] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pages 3384–3391. IEEE, 2008.

[28] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, May 9-13 2011.

[29] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent point feature histograms for 3d point clouds. In Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany, pages 119–128, 2008.

[30] Samuel Schulter, Paul Vernaza, Wongun Choi, and Manmohan Chandraker. Deep network flow for multi-object tracking. In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pages 2730–2739. IEEE, 2017.

[31] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, pages 1297–1304. Ieee, 2011.

[32] Jeremy Siek, Andrew Lumsdaine, and Lie-Quan Lee. The boost graph library: user guide and reference manual. Addison-Wesley, 2002.

[33] Jeany Son, Mooyeol Baek, Minsu Cho, and Bohyung Han. Multi-object tracking with quadruplet convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5620–5629, 2017.

[34] Luciano Spinello and Kai O Arras. People detection in rgb-d data. In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pages 3838–3843. IEEE, 2011.

[35] Luciano Spinello, Rudolph Triebel, and Roland Siegwart. Multimodal people detection and tracking in crowded scenes. In AAAI, pages 1409–1414, 2008.

[36] Siyu Tang, Mykhaylo Andriluka, Bjoern Andres, and Bernt Schiele. Multiple people tracking by lifted multicut and person reidentification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3539–3548, 2017.

[37] Luchao Tian, Mingchen Li, Guyue Zhang, Jingwen Zhao, and Yan Qiu Chen. Robust human detection with super-pixel segmentation and random ferns classification using rgb-d camera. In Multimedia and Expo (ICME), 2017 IEEE International Conference on, pages 1542–1547. IEEE, 2017.

[38] Dong Zhang, Omar Javed, and Mubarak Shah. Video object co-segmentation by regulated maximum weight cliques. In European Conference on Computer Vision, pages 551–566. Springer, 2014.

[39] Haiyu Zhao, Maoqing Tian, Shuyang Sun, Jing Shao, Junjie Yan, Shuai Yi, Xiaogang Wang, and Xiaoou Tang. Spindle net: Person re-identification with human body region guided feature decomposition and fusion. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1077–1085, 2017.

[40] Kaiyang Zhou, Adeline Paiement, and Majid Mirmehdi. Detecting humans in rgb-d data with cnns. In Machine Vision Applications (MVA), 2017 Fifteenth IAPR International Conference on, pages 306–309. IEEE, 2017.