University of Padua

Department of Information Engineering

Master's Thesis in

Computer Engineering

People tracking and following with a smart wheelchair using an omnidirectional camera and a RGB-D camera

Supervisor

Prof. Enrico Pagello

Candidate

Federico Zanetello

Co-Supervisor

Prof. Chen Weidong (Shanghai Jiao Tong University)

Academic Year 2013/2014

Ai miei genitori, che, nonostante le tante difficolta', mi hanno supportato e sopportato ogni giorno, dall'inizio alla fine, di questa lunga e straziante battaglia.

"It's better to light a candle than curse the darkness"
Peter Benenson - 1961

Abstract

This thesis has been carried out within a project at ARL (Autonomous Robot Lab) and IAS-Lab (Intelligent Autonomous Systems Lab) of Shanghai Jiao Tong University and University of Padua respectively. The project implements a new service that enables a wheelchair user and another person to have a normal talk while strolling freely around the environment, without the need of any interaction towards the wheelchair, called JiaoLong.
From the wheelchair point of view, the goal consists mainly in two steps: detect the person whom the wheelchair user wants to talk with and follow her by standing by her side.
The project started from scratch: as further explained in the upcoming chapters, we chose to use one RGB-D camera, since there are many related works available based on this sensor, and one omnidirectional camera, made by ARl of Shanghai Jiao Tong University, since a single image frame contains all the information we needed about the wheelchair surrounding scene. The approach we followed is quite simple: first we detect the person that the user wants to talk with with the RGB-D camera, and then the smart wheelchair will put itself aside her by using the RGB-D camera as an obstacle detector and the omnidirectional camera as a person tracking gizmo. This approach exploit most of the advantages of the two sensors while concealing their disadvantages by using one camera instead of the other.
In this thesis we used many tools from many different sources that made the work more exciting since the results never put us down. The work reaches near state of the art performance and very high frame rates in our distributed ROS-based CPU implementation.

Contents

List of Figures

Chapter 1

Introduction

1.1 Summary

As the proportion of elderly population has increased consistently over the past decades, more and more smart wheelchairs, also known as wheelchairs equipped with traditional mobile robots technology, are developed to assist and help old and disabled people during their daily lives. By using smart wheelchairs, people could get rid of onerous maneuvers and dangers like collisions with other objects and user falls. The enhancements of independent mobility by a smart wheelchair also help people to rebuild their confidence on social skills. Since people's control ability could be affected by various factors, such as inherent control ability, fatigue, environmental visibility, and distractions, the purpose of this work is to implement a new service in a smart wheelchair, described in chapter 2.3, that will allow a natural conversation to take place between the smart wheelchair user and somebody else without the need of interacting with the wheelchair at all during the whole length of the conversation. From the smart wheelchair point of view, this service can be executed in these three functional steps:

1. Detect the person whom the smart wheelchair user wants to talk with;
2. Track her;
3. Stay by her side while the conversation is taking place;

1.2 Related works

There are many works about person following robot. The big difference in our project is that the mobile robot, our wheelchair, doesn't have to simply follow the person but to stay by her side, in order to let the wheelchair user and the target to have a natural conversation. Probably, the most important related works are:

- Person Following Robot ApriAttenda™ [25]
  This robot, from Toshiba corporation[1], uses stereo vision for target detection and target tracking while it uses ultrasonic sensors for obstacles avoidance while following the person. The main problems with this solution are:
  - the robot easily loses the person during the tracking;
  - the robot can't both stand aside the person and follow her at the same time due to its limited degree of freedom of the head;
- USC's Segway RMP [14]
  Very similar to our final product, this robot uses one omnicamera and one Laser rangefinder (LRF) for target detection, target tracking and obstacles avoidance while following the person. The final work is quite effective, trustworthy and the Segway can keep following the person even when it is far away. The only drawback of this solution is that is not made for the same purposes as ours: it can't both follow and stand by the person side at the same time;

Regarding the tracking side of the work, several approaches for people tracking have been developed over the last years. The most widely used approach is visual tracking [26] [15] [24] [13]. Most of the existing techniques differ mainly in the features or models that they use to extract people meta-information from the images. Since most of these algorithms use standard still cameras, they are little to no use in our project: we will have to build a new algorithm.

1.3 Proposed solution

Our solution is similar to the USC's Segway RMP described above. The project consists in the JiaoLong smart wheelchair, introduced in chapter 2.3, equipped with two sensors: one omnidirectional camera and one RGB-D camera.In the upcoming chapters we will discuss in detail the main reasons on why we decided to use those sensors and present you how everything works together.

1.4 Content

This document is structured as follows:

- Chapter 2: describes the main hardware and software used to build the system;

[1] `http://www.toshiba.co.jp`

- Chapter 3: describes the approach we followed to set up the system;
- Chapter 4: describes the people detection algorithm within the kinect;
- Chapter 5: describes in details the calibration algorithm between the omnidirectional camera and Kinect;
- Chapter 6: describes in details how the people tracking algorithm works;
- Chapter 7: describes how the machine chooses a goal and how the motion control works;
- Chapter 8: contains the work conclusions and proposes further implementations and improvements;

Chapter 2

Work Environment

2.1 Microsoft Kinect

Fig. 2.1: Microsoft Kinect

Microsoft Kinect is an accessory for the Microsoft Xbox 360 video game platform. It is considered a "controller-free gaming and entertainment experience". In fact, it can interpret 3D scene information using a RGB camera, a depth sensor and a multi-array microphone.
The depth sensor consists of an infrared laser projector, which creates a grid of points, combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions creating a depth map. The sensor has an angular field of view of 57° horizontally and 43° vertically, while the motorized pivot is capable of tilting the sensor up to 27° either up or down. The horizontal field of the Kinect sensor at the minimum viewing distance of 0.8 m is therefore 87 cm, and the vertical field is 63 cm, resulting in a resolution of just over 1.3 mm per pixel.

2.1.1 Video Stream

Reverse engineering has determined that the Kinect's various sensors output video at a frame rate of 9 Hz to 30 Hz depending on resolution consisting

in two different streams:
The first one is provided by the RGB camera which uses 8-bit VGA resolution (640x480 pixels) with a Bayer color filter, the other one, instead, is produced by the monochrome depth sensor with the same resolution and 2,048 levels of sensitivity (11 bits).

2.1.2 Software Technology

The software technology provided to the Xbox developers enables advanced gesture recognition, facial recognition and voice recognition as well as tracking up to six people simultaneously (declared, but it is able to track probably as many people as they can fit in the field-of-view of the camera), including two active players for motion analysis with a feature extraction of 20 joints per player.
While this is all true under a Microsoft Windows environment, in our project we worked under a Linux Ubuntu environment, which is not supported by official Microsoft sources. However, as we'll show in the next few chapters (chapter 2.5.2), a similar software technology is provided by a non-profit organization called OpenNI[1], in which one of the main members is PrimeSense, the company behind the technology used in the Kinect. Under this organization PrimeSense released an open source version of the drivers for the sensor: thanks to this and the community behind OpenNI, the differences between the two environments (Microsoft Windows and Linux) are getting slimmer and slimmer: it has become quite easy to use the Kinect even under an officially non-supported Operating System.

2.2 Omnidirectional Camera

(a) (b)

Fig. 2.2: Figure 2.2(a) shows us an example of a common camera, the Webcam Philips SPZ5000. Figure 2.2(b) shows us a glass holding the omnidirectional mirror inside.

[1] `http://community.openni.org/openni`

While a camera with standard lenses normally has a field of view that ranges from a few degrees to, at most, 180°, an omnidirectional camera is a camera with a 360° field of view in the horizontal plane. The main advantage of this kind of camera over a standard one is clearly the new wider and more accurate perception capacity, especially if used in a mobile robot.
The omnidirectional camera used in this project is composed by a Point Gray Chameleon®[2] camera and a customized curved mirror. The Chameleon camera is a USB 2.0 1.3 MegaPixels digital video camera that uses a Sony EXview HAD CCD®sensor, which allows to capture images up to 1296x964 of resolution at 18 FPS.

2.3 JiaoLong

Fig. 2.3: The Shanghai Jiao Tong University Smart Wheelchair JiaoLong

JiaoLong is a smart wheelchair built by the Autonomous Robot Lab of Shanghai Jiao Tong University. The main goal of this smart wheelchair is to help its target users, that are mainly elderly or people with a various range of disabilities such as visual defects and trembling hands, during their movements around the environment in their daily lives.
Figure 2.4 shows us that there are three interactive ways to input the human guidance: the first one is a standard microphone, used for voice commands, the second and third ones are an omnidirectional joystick and a touch screen, used both for physical and more straight forward commands. Obviously this means that the JiaoLong users can interact with and control the robot through voice (microphone) and hands (joystick, touchscreen). The touch-screen also provides all kind of real-time information to its users, such as their location in a global map and the current speed of the wheelchair.

[2] `http://ww2.ptgrey.com/USB2/chameleon`

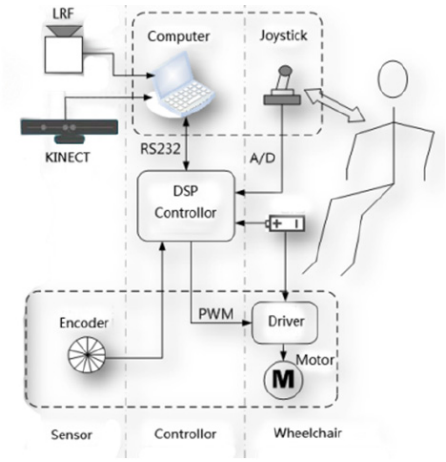


Fig. 2.4: The Shanghai Jiao Tong University Smart Wheelchair JiaoLong

2.3.1 Hardware and Software technology

The machine length, width and height are respectively 110cm, 90cm and 105cm. Its power is provided by a group of 12V batteries which allow the wheelchair to operate autonomously up to 8 hours continuously. The back wheels adopt DC motors which are controlled by a motor controlling board (DSP 2407) adopting two wheeled differential driving. The maximum translational and rotational speeds of the wheelchair are 500 mm/s (1.8 km/s) and 50 deg/s (0.8727 rad/s) respectfully. The smart wheelchair has an Industrial Computer (WinXP, Intel Core2 Duo @ 2.0GHz CPU) for management of interactive information, and a Laptop (Linux, Intel ATOM N270 @ 1.6GHz CPU) for real-time execution of the navigation algorithm and every other computation that regards our project. The sensors are: an encoder-based odometer, a Laser Range Finder (LRF, SICK LMS100), and an RGB-D camera (Microsoft Kinect).

2.3.2 Shared Control

Shared control is a common algorithm used for human and wheelchair cooperation. The one used in our project is based on two key parts: the reactive controller and the weight optimizer. The reactive controller provides obstacle avoidance help using Minimum Vector Field Histogram (MVFH) [1] [10] and Vector Force Field (VFF) [3] [2] methods. The weight optimizer optimizes three indicators, which will be discussed in the following section obtaining an equilibrium between the machine reactive control and the user control.

Weight Optimization

In the previous work [18], indicators of wheelchair's performance were proposed: safety, comfort and obedience. Safety measures the wheelchair probability of collision. Comfort measures the smoothness of the wheelchair speed. Obedience measures the degree of the machine obedience to the user's commands. These indices are introduced in the three following paragraphs:

Safety

$$safety = 1 - exp(-\alpha * dis) \tag{2.1}$$

Where α is a constant, and *dis* represents the distance in millimeters between the wheelchair and closest obstacle in its path. *dis* is determined by the current and desired speed of the wheelchair and by the obstacles distance around the wheelchair. Since the kinematic model is inaccurate in predicting long-term movement, we only predict the path for the next 4 seconds. The negative exponential function is used to normalize the index, and to make the comparison between the three indices (Safety, Comfort, and Obedience) easy.

Comfort

$$comfort = exp(-\beta|\omega - \omega_0|) \tag{2.2}$$

Where β is a constant. To avoid violent change in motion command, we define ω as a linear combination of ω_{user} and ω_{match}, where ω_{user} is the desired rotational speed from the user and ω_{match} is the desired rotational speed generated by the reactive controller. Since the wheelchair's translational speed is given directly by the user and it won't change unless a collision is about to happen, there is no component that reflects the translational speed change in the equation.

Obedience

$$obedience = exp(-\gamma|\xi - \xi^*|) \tag{2.3}$$

Where γ is a constant and ξ^* is the orientation calculated from the user's input v_{user} and ω_{user} while ξ is the orientation determined by ν and ω.

The aim of weight optimization is to maximize all three indicators. However, these three indices, under typical circumstances, are normally contradictory to each other. For example when a wheelchair is traveling through a crowd, the most influent index should be safety. Therefore, there is no absolute optimal solution for the three equations (equations 2.1, 2.2, and 2.3).
An evaluation function of this problem is needed to achieve an effective solution. It was found that increasing a certain index which is already above a

certain value will make the other two indices drop drastically. For example, enforcing an increment to safety to when it's already above 0.9 will make the wheelchair always choose the most spacious path, and the user will feel like the wheelchair is not moving under his control at all. Therefore, the principle we chose to solve this problem is to always increment the smallest of the three indexes. In accordance with this principle, we use the minimax method to simplify the previous equation into a single objective problem:

$$\begin{cases} max_k(min(safety, comfort, obedience)) \\ s.t \\ \nu(t) = \nu_{user}(t) \\ \omega(t) = k\omega_{user}(t) + (1-k)\omega_{mach}(t) \\ 1 \geq k \geq 0 \end{cases} \tag{2.4}$$

The precedence relation among indices will change naturally when facing different situations. For example, when a user is cruising in a spacious room with a wheelchair, the possibility of hitting an obstacle is small, i.e. safety has a high value. In this case $max(min(safety, comfort, obedience)) = max(min(comfort, obedience))$, then comfort and obedience become priorities. Whereas when a user is passing through the crowd, the possibility of a collision may significantly increase. Then safety could be taken into consideration as a priority since $max(min(safety, comfort, obedience)) = max(safety)$.
A user with good control abilities can drive a wheelchair smoothly and safely, in this situation $max(min(safety, comfort, obedience)) = max(obedience)$, so the wheelchair will be driven completely under the user's will. When the user's control ability drops, he would not be able to preserve smoothness and safety. In this case, the reactive controller will take over the control of the wheelchair and assist the user.

2.4 ROS

Robot Operating System[3] (ROS) is a software framework for robot software development, providing operating system-like functionality on a heterogeneous computer cluster. As of 2014, development continues primarily at Willow Garage[4], a robotics research institute/incubator, with more than twenty institutions collaborating in a federated development model.
ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It is based on a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and

[3] `http://www.ros.org`
[4] `http://www.willowgarage.com`

other messages. The library is geared toward a Unix-like system (Ubuntu Linux is listed as supported while other variants such as Fedora and Mac OS X are considered experimental).
ROS has two basic "sides": The operating system side, called ros, as described above and ros-pkg, a suite of user contributed packages (organized into sets called metapackages) that implement functionality such as simultaneous localization and mapping, planning, perception, simulation etc.
Despite the importance of robot reactivity, ROS is not a real-time OS, though it is possible to integrate ROS with real-time code.
ROS is released under the terms of the BSD license, and is open source software. It is free for commercial and research use.

2.4.1 Computation graph architecture

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

Nodes.
A node is substantially a process that performs computation: it is where all the main operations are done, and it is also the main entity in the graph. ROS wants a robot to be composed by multiple nodes, each one of them with a specific function. Nodes communicate among them using streaming topics, RPC services or the Parameter Server.

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

Topics.
ROS Messages are managed as in a Distributed Event Based Systems: this means that nodes have to follow the publisher/subscriber semanthic. A topic is a named bus over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption and are intended for unidirectional, streaming communication. There can be multiple publishers and subscribers to each topic.

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

Master.
Name service for ROS. It provides name registration and lookup to the rest of the computation graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services: this is also why the master must be launched before anything ROS-related.

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

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

An example of a graph created during a ROS session is visible in figure 2.5

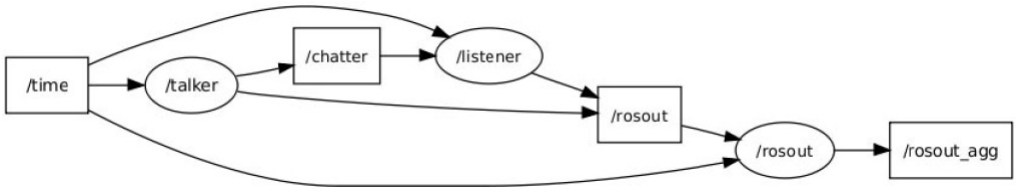


Fig. 2.5: Graphical representation of nodes (ellipses) and topics (rectangles).

2.5 Libraries

2.5.1 PCL

The Point Cloud Library[5] (PCL) is a large scale, open project for point cloud processing [20].
The PCL framework contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. These algorithms can be used, for example, to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract keypoints and compute descriptors to recognize objects in the world based on their geometric appearance, and create surfaces from point clouds and visualize them. It is well integrated into ROS which also provides some functionalities as ready to use nodes.

[5] `http://www.pointclouds.org`

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

2.5.2 OpenNI

OpenNI[6] (Open Natural Interaction) [27] is a multi-language, cross-platform framework that defines APIs for writing applications utilizing Natural Interaction.
NI (Natural Interaction) refers to the concept that Human-Machine-Interaction is achieved by human senses and, most of all, vision and hearing. OpenNI aims to define a standard API that is able of dealing with both vision and sensors, and a vision and audio perception middleware, allowing communication between the two components.
OpenNI provide two types of APIs:

1. implemented APIs: allow to deal with the sensor device;
2. not implemented APIs: allow to deal with the middleware components.

The clear distinction between sensors and middleware components is based on the "write once, deploy everywhere" principle. In fact OpenNI allows the porting of applications and moreover enables to write algorithm that works with known raw data independently from the sensor that has generated them. From the producer point of view, instead, OpenNI offers the possibility of building sensors for applications by just providing raw data and not APIs on how to deal with them. An application of OpenNI is for example the tracking of real-life 3D scenes.
OpenNI is an open source API that is publicly available.
The OpenNI Framework is an abstract layer (Figure 2.6) that provides the interface for both physical devices and middleware components. Multiple components can register to this framework based on the specific API and they are called modules.
A module is responsible for producing and processing the data of the sensor and the currently supported ones are:

1. 3D sensor;
2. RGB camera;
3. IR camera;
4. audio device.

Based on this, OpenNI provides also the following middleware components:

[6] `http://community.openni.org/openni`

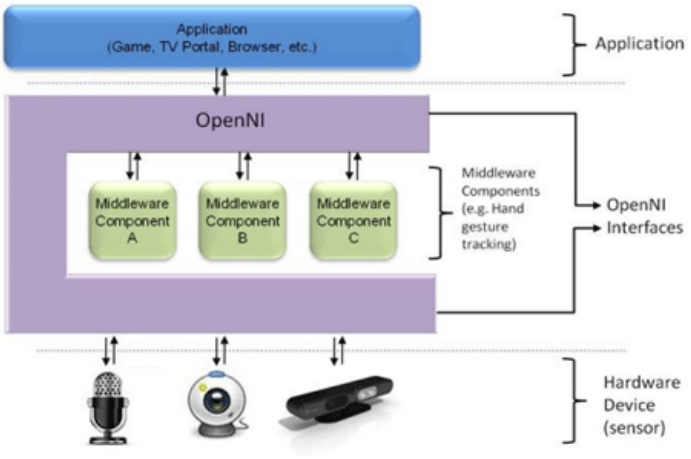


Fig. 2.6: The OpenNI abstract layered architecture.

1. full body analysis;
2. hand point analysis;
3. gesture detection;
4. scene analyzer (segmentation, clustering and coordinates framing).

OpenNI relies on Production Nodes. They represents the productive part of the system, that is they create the data required for the interaction. These data can be either low level ones, RGB for example, either composited ones. In fact production nodes can also control lower level production nodes and they can in turn be used by higher level ones. In order to define communication and hierarchy these nodes are organized in production chains.

2.5.3 OCamCalib

The OcamCalib (Omnidirectional Camera Calibration Toolbox for Matlab) allows the user (even inexpert users) to calibrate any central omnidirectional camera, that is, any panoramic camera having a single effective viewpoint The Toolbox implements the procedure initially described in the paper [22] and later extended in [23], [21], and [19].
The Toolbox permits the user to easily and quickly calibrate the omnidirectional camera through two steps. First, it requires the user collect a few pictures of a checkerboard shown at different positions and orientations. Then, the user is asked to extract the corner points.
After the calibration, the toolbox provides two functions (cam2world and world2cam) which express the relation between a given pixel point and its projection onto the unit sphere (this is a 3D vector emanating from the single effective view point). This relation clearly depends on the mirror shape

and on the intrinsic parameters of the camera. The novel aspects of the OCamCalib Toolbox with respect to other toolboxes are the following:

- The toolbox is the only one with Automatic Corner Extraction (no manual extraction is required).
- The toolbox does not require a priori knowledge about the mirror shape.
- It does not require calibrating the perspective camera separately: the system camera-mirror is treated as a unique compact system that encapsulates both the intrinsic parameters of the camera and the parameters of the mirror.
- The detection of the image center is performed automatically. It does not require the visibility of the circular external boundary of the mirror. Unlike other toolboxes, which require the visibility of the external boundary of the mirror to determine the image center, the OCamCalib Toolbox automatically identifies the center without any user interaction!

The calibration performed by the OCamCalib Toolbox is based on the following hypotheses:
The camera-mirror system possesses a single effective viewpoint, or also a "quasi" single viewpoint. In fact, the Toolbox is able to provide an optimal solution even when the "single view point property" is not perfectly verified (for instance when the camera optical center is not exactly in the focus of the hyperbola or also for spherical mirrors). The Toolbox showed to give very good calibration results even in the latter case (reprojection error $<$ 0.5 pixels!).

2.5.4 OpenCV

OpenCV (Free Open Source Computer Vision) is a cross-platform library of programming functions mainly aimed at real time computer vision [4] [12] . Example applications of the OpenCV library are Human-Computer Interaction (HCI), object identification, segmentation and recognition, face recognition, gesture recognition, motion tracking, ego motion, motion understanding, structure from motion (SFM), stereo and multi-camera calibration and depth computation, mobile robotics. As for PCL, OpenCV is completely integrated into ROS which also provides image type conversions between OpenCV and ROS formats. It has a BSD license and it is free for commercial or research use.

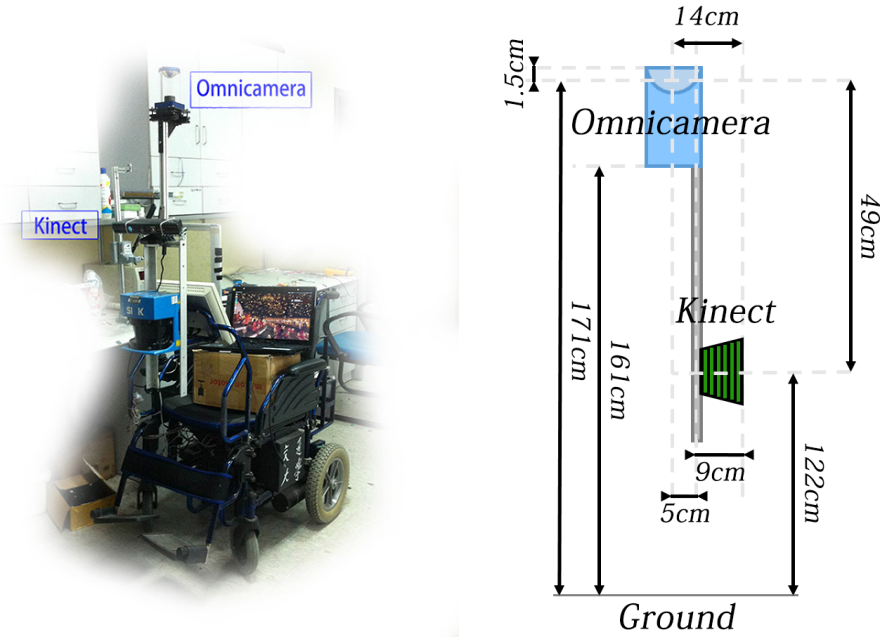
Chapter 3

System Overview

While tracking and following a person with a Microsoft Kinect in a robot is not very complicated, the problem is taken to a whole new level when the person, like in our case, is constantly out of the Kinect frame. This happens because, in our project, the tracked person must stay by the side of the wheelchair and because the narrow angle vision of the Microsoft sensor simply can't look at both obstacles in front of the wheelchair and at the person at the same time. Remember that the main function for the Kinect in our project is not to track the person but to cover part of the navigation job in the machine, providing information about the environment in front of it, such as obstacles and more.
The first solution that probably comes into everybody's mind when thinking on how to fix this problem is to implement and use more than one Kinect sensor at the same time. While this is completely feasible, with this solution two main problems must be considered immediately: the first one is the higher cost that comes with the requirement of buying more than one Kinect; the second is the more powerful system machine horsepower that this solution needs in order to process all the data coming from more than one Kinect sensors in real time. Both those new issues represent what we are trying to avoid by all costs in order to keep the machine affordable to the consumer market.
The solution we provided, instead, is a combination of a Kinect sensor and an omnidirectional camera, cheaper than a single Kinect sensor: thanks to the 360° field of view of the omnidirectional camera, the machine can always track the person even if she can't be seen in the Kinect frame for a long period of time. Furthermore, working with the omnicamera 2d image frame requires way less machine horsepower than working with another (or more) point cloud provided by another hypothetical Kinect sensor.

3.1 Sensors positions

Once we settled down on which sensors we were going to use, the next challenge was to find out where to setup both sensors in the machine, in order to work properly and efficiently. Since the assigned smart wheelchair has been already used with both these sensors in previous works with similar purposes as ours, we chose to trust these previous works and use the same position for both sensors.
In more detail, we have that:



(a) (b)

Fig. 3.1: Figure 3.1(a) displays the actual JiaoLong Smart Wheelchair. Figure 3.1(b) is a scheme with the sensors position in the JiaoLong Smart Wheelchair

- The Kinect sensor is located 122cm above the ground and 70cm from the back of the seat of the user.

- Since the main purpose of this sensor is to give useful information to the Navigation stack of the machine, the Microsoft sensor is not parallel to the ground but slightly tilted towards the floor, allowing a more precise information about the obstacles that are in front of the wheelchair. To be more specific, the pitch of the camera is 0.25 rad (14.3°) while the yaw and roll are obviously 0 rad.

- The Omnidirectional camera's base is located at 161cm above the ground, with the virtual perspective camera at 171cm, and 52.5cm from the back of the seat of the user (with the virtual perspective camera at 56cm). This location guarantees the visibility of the tracked person in the camera frame in as many conditions and positions as possible without making the sensor too annoying to the smart wheelchair user. Because of reasons explained in the next chapters, this time the camera is parallel to the ground.

This way both centers of the cameras are aligned and we had little job to do for setting these sensors up into the machine. The pitch of the Kinect sensor will introduce a few troubles with the camera's calibration but nothing that we couldn't manage, as we'll see in a few chapters (chapter 5.2.2).

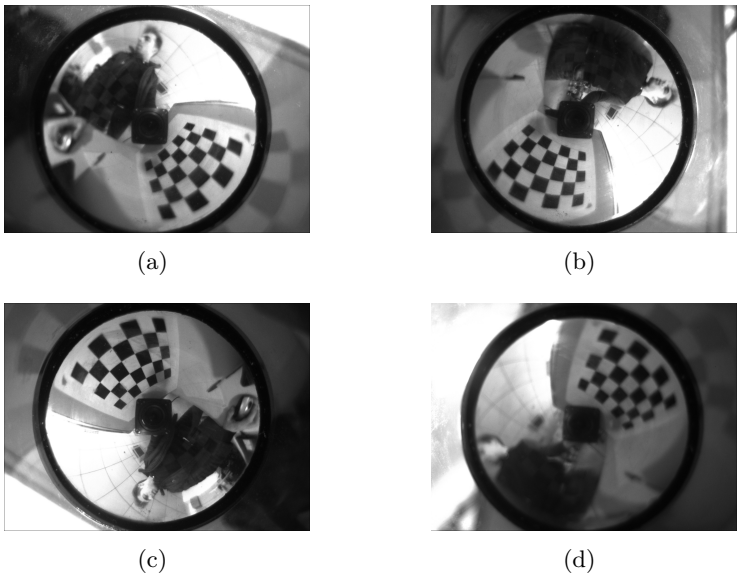
3.2 Sensors inner calibration

Thanks to the work done within the OpenNI framework, the Kinect calibration is nothing to concern about: everything has already been take care of in the OpenNI grabber framework of PCL.

Since the omnidirectional camera is custom made, we cannot obviously apply the same line of thought for this second camera. As introduced in chapter 2.5.3, the omnidirectional camera calibration needs to use OcamCalib toolbox developed by Davide Scaramuzza. By calibrating this kind of camera that, remember, is just a linear perspective camera with a parabolic mirror in front of it, we are allowed to generate linear perspective images that are free of distortion or to obtain useful information from the omnicamera image frame.
But before diving into that, we suggest you to refresh your memory on how an omnicamera works by reading our introduction of some foundamentals on the omnidirectional sensor in our appendix A.

3.3 Omnidirectional camera calibration

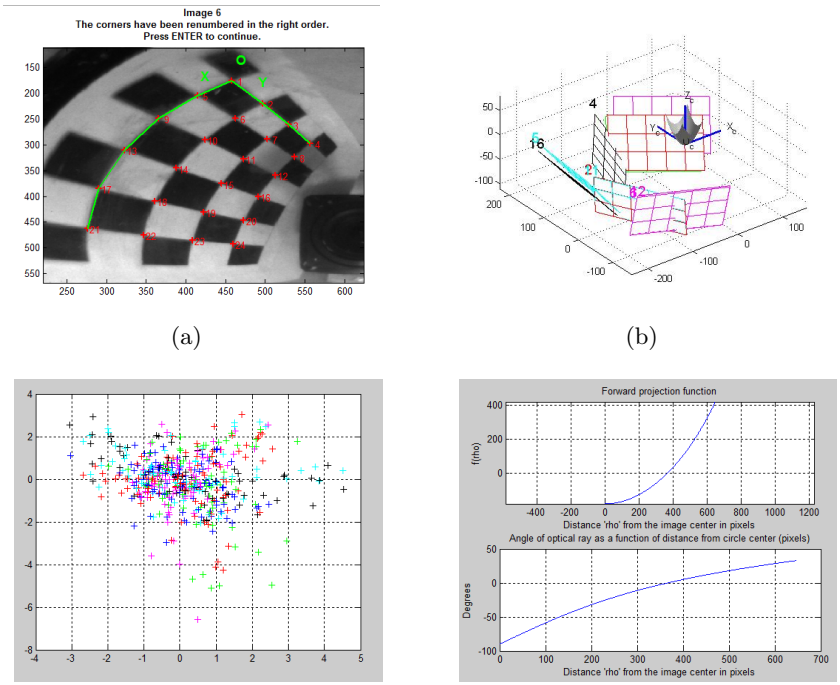
Once the camera is set up, the first thing to do is take pictures. Since we still are in the calibration phase, we have to use a calibration pattern that will be recognized afterwards by our calibration tool. The calibration pattern for the OcamCalib tool is a common black and white chessboard that has to be printed and placed on a flat rigid surface that guarantees the pattern to stay flat for all the duration of the calibration.
The images taken (figure 3.2) should have the whole chessboard as close as possible to the optics of the omnidirectional camera while always showing clearly all the chessboard four corners. Also, in order to obtain the best calibration results as possible, the calibration pictures, together, should have



(a) (b) (c) (d)

Fig. 3.2: Sample of snapshots used during the calibration of our omnidirectional camera

the checkerboard to cover all the visible area of the camera (all around the parabolic mirror). Once these pictures are ready, the next phase of the calibration is also the most important one: in this phase the OcamCalib tool will extract all the grid corners from all the pictures. This phase teaches to the OcamCalib tool what the omnidirectional camera sees and in which way, in fact, thanks to this phase, the tool will be able to estimate the camera's mirror shape, absolutely fundamental for the scope of the calibration.
Once this last phase is complete, OcamCalib tool will have a few more little things to do in order to improve the calibration results.
The first of which is the image center estimation: it would be ideal that the pixel at the center of the 2D image would also be the effective center of the omnidirectional image, unluckily, most of the time, this is not the case: this is the reason why the OcamCalib tool runs a routine that iteratively just applies a linear estimation method to each image used in the corner extraction phase to estimate where the effective center is.
Once the center position is computed, the OcamTool offers another phase, called Calibration Refinement Service, which refines all the calibration parameters again, including the position of the effective image center, using a non-linear method called Levenberg-Marquadt algorithm, which gives back improved calibration results. The optimization is performed by attempting to minimize the sum of squared reprojection errors.
The non-linear refinement is done in two steps. First, it refines the extrinsic camera parameters, that is, the rotation and translation matrices of



(a) (b)

(c) (d)

Fig. 3.3: Figure 3.3(a) shows us the extract grind corners phase, figure 3.3(b) shows where the checkboard was in each picture (each picture has a different color), figure 3.3(c) shows us the tool-estimated omnidirectional camera mirror shape, and figure 3.3(d) shows us the complete error analysis of the pictures taken for the calibration

each checkerboard with respect to the camera (i.e. RRfin). Then, it refines the camera intrinsic parameters (i.e. ocam_model). As the extrinsic and intrinsic parameters are not independent, the refinement may need a few iterations to converge to a solution that minimizes both intrinsic and extrinsic parameters. [22] [23] [21] [19]

3.4 Sensors communication

As we can read from the last 20 years of literature, detecting and tracking people by using only an 2D image frame is not an easy task, especially when the image frames come from an omnidirectional camera mounted on a mobile robot: this is why in this project we chose to detect the person whom our wheelchair user wants to talk with by using the Kinect camera. This obviously requires that person to stand in front of the wheelchair in order to be detected.
Once detected, the person tracking algorithm will be handled by the omnidirectional camera. As we shall see in chapter 5, this kind of collaboration between the two sensors requires one more calibration: this time we're talking about a calibration that is not within the cameras but between them. Furthermore, the people detection algorithm part is described in detail in chap 4, while the people tracking part in described in detail in chapter 6.

Chapter 4

People Detection

Although we said that the Microsoft sensor is mainly used to avoid obstacles in the environment, in our project the Kinect camera is also used for one more fundamental step: people detection. This step is the first one out of three (as we introduced in the section 1.1), in our project: once the person has been detected, the Kinect is free to go back permanently to its usual function of obstacle avoidance, while the omnidirectional camera will start to track the person with the information received by the Microsoft sensor.

4.1 Ground Based People Detection Algorithm

By default, the Kinect camera gives us an everlasting flow of point clouds data, so what should we do in order to detect people?
Thankfully there's a work, called *ground based people detection* [16], publicly available in the official site of the Point Cloud Library[1] that we can use freely within our project:
During its initialization, the algorithm asks the user to point out where the ground plane (as know as the floor) is from one RGB-D frame that is displayed on the touchscreen of the wheelchair as soon as the algorithm is launched: while not completely necessary, this initialization enhances the outcome of the detection and further avoids fake positives. Since the position of the Kinect sensor mounted on the wheelchair never changes, this initialization can be, and in our work is, completely skipped in our work by calibrating it offline. This also results in a better integration of the algorithm within the JiaoLong wheelchair.
Once the initialization phase is completed, all the RGB-D data coming from the Kinect is processed by a detection module that alters the point cloud data by removing the points of the ground and it performs a 3D clustering of the remaining points. Furthermore, in order to keep only those points that are more likely to belong to the class of people, the method applies a

[1] `http://www.pointclouds.org`

HOG-based people detection algorithm to the RGB image of the resulting clusters. All these actions are made by working directly on the point cloud received from the camera.
The 3D clustering is probably the most important part of this method and it uses some human shaped SVM models: in machine learning, Support Vector Machines (SVM) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.
Thanks to this model of recognition, every cluster of the remaining points of our point cloud is evaluated and given an HOG confidence (as know as the confidence that the analyzed cluster represents a real-world person): once the method finds a cluster with HOG confidence higher than a certain customized threshold, the algorithm stops, draws a bounding box around the detected person, and publishes details about that person over a dedicated ROS topic: details that will be promptly read by the omnidirectional camera node.
After this message publication, the algorithm quits, leaving the Kinect sensor to its usual job in the navigation part.

Chapter 5

Calibration between Omnidirectional camera and Kinect camera

As introduced in section 3.4, another challenge that we had to face during the project was making the communications between the Microsoft Kinect camera and the Omnidirectional camera possible. By communication we mean that if the Kinect camera wants to point at something in its point cloud, we can view exactly the same point in the omnicamera's 2D image.
By exchanging information that both sensors can see in their own frame, there could be many new important developments in the field of robotics, and computer engineering in general.
Once the inner calibration of both cameras is completed (Chapter 3), the only job left to do is the transformation of the information seen from the point cloud data into information seen on the 2D frame image and viceversa. The task is even harder when we consider the low-precision of both cameras and the different tilt position (as stated in section 3).
Furthermore, converting a random point from the Kinect point cloud into a pixel of the omnidirectional camera 2D image frame is nearly impossible unless some hypothesis on the point are made.

5.1 Cam2World vs. World2Cam

Note: we suggest you, again, to refresh your omnicamera knowledge by reading the appendix A.

A fundamental part of this section has been thoroughly taken care of by [22] [23] [21] [19], this toolbox not only provides the omnidirectional camera calibration tool, but also two class methods (CAM2WORLD and WORLD2CAM) which expresses the relation between a given pixel and its

projection onto the unit sphere (that is a 3D vector emanating from the single effective view point). This relation clearly depends on the mirror shape and on the intrinsic parameters of the omnidirectional camera: this, in fact, is the reason why we need to calibrate the camera first.

5.1.1 World2Cam

$$m = world2cam(M, ocam\_model) \tag{5.1}$$

World2Cam projects a 3D point on to the image and returns the pixel coordinates. M is a *3xN* matrix containing the coordinates of the 3D points: $M = [X; Y; Z]$. ocam_model contains the model of the calibrated camera (as known as the outcome of the inner calibration). $m = [rows; cols]$ is a *2xN* matrix containing the returned rows and columns of the points after being reproject onto the image.

5.1.2 Cam2World

$$m = cam2world(M, ocam\_model) \tag{5.2}$$

Back-projects a pixel point to the unit sphere M. $M = [X; Y; Z]$ is a *3xN* matrix with which contains the coordinates of the vectors emanating from the single-effective-viewpoint to the unit sphere, therefore, $X^2+Y^2+Z^2 = 1$.

$m = [rows; cols]$ is a *2xN* matrix containing the pixel coordinates of the image points. ocam_model contains the model of the calibrated camera.

Since these functions request and return only a direction in the real world, we must assume that whatever is visible from the Kinect sensor is also visible by the omnicamera, while this is not always true in general, we can rest assured that this is the case in our application. Thanks to this hypothesis all our conversions can be made easily just by using the functions introduced.

5.2 The conversion

5.2.1 Offset

As we saw in the previous section (section 5.1), world2cam can identify a pixel in the omnicamera 2D frame just by knowing the direction (pixel point m onto the unit sphere) of the point from the virtual perspective camera. Thanks to this method and the information published by the Kinect (see chapter 4.1) we meet all the requirements for identifying the same point seen from the Kinect point cloud into the omnicamera 2D image.
Let's say that we have one point from the point cloud that we want to see in

the omnicamera image frame: thanks to how the point cloud is built, we have the direction, a 3D vector, pointing from the center of the Kinect to exactly that point, according to the Microsoft sensor axes. This 3D vector contains also information on the distance of the desired point from the Kinect, but is not needed for what's next.

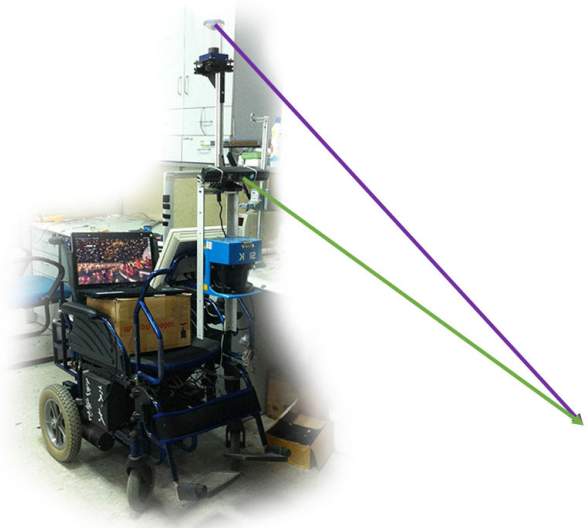


Fig. 5.1: A graphical example of the two sensors pointing at the same point

The first thing that shall be done is converting this direction into one compatible with the omnicamera axes, obtaining a 3D vector emanating from the single effective view point of the omnidirectional camera and pointing directly to the same point seen from the Kinect camera (as seen in figure 5.1). This is done by converting each axis value from the Kinect coordinate system into the correspondent ones in the omnicamera coordinate system. The following steps outline this conversion:

- The x-axis value of the omnicamera is the opposite value as of the x-axis of the Kinect: so if in the Kinect sensor z-axis has the value of +3, in the omnicamera the same value will be -3, also, we have to consider the offset between the omnicamera and the Kinect (chapter 3.1) in this direction, we do this by adding 14cm (as seen on figure 3.1(b)) to the final result;

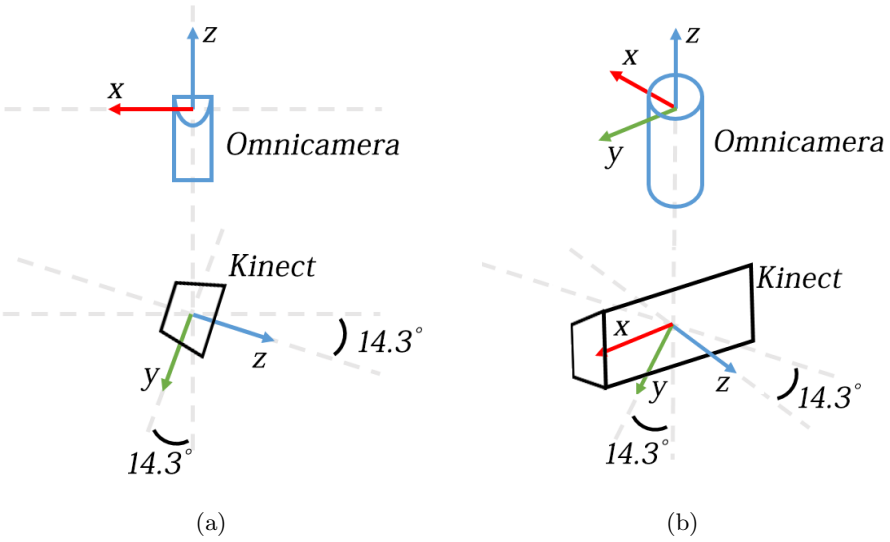
- The y-axis of the omnicamera is the same as the x-axis of the Kinect, we also don't have to add any offset in this direction;

- The z-axis, like the x-axis, must change the sign of the value of the Kinect y-axis and also consider the offset of 49cm between the two cameras (as seen on figure 3.1(b));

Mathematically speaking, the coordinates translation conversion leads to these three equations:

$$\begin{aligned} Omni\_Coord_x &= -(Kinect\_Coord_z + 0.14) \\ Omni\_Coord_y &= Kinect\_Coord_x \\ Omni\_Coord_z &= -(Kinect\_Coord_y + 0.49) \end{aligned} \tag{5.3}$$

5.2.2 Pitch



(a) (b)

Fig. 5.2: Figure 5.2(a) shows us an over simplified scheme of the side view of the sensors as they are mounted on the wheelchair, empathizing the rotation of the Kinect sensor. Figure 5.2(b) shows us exactly the same thing, but in a 3d world, displaying all the axes of the two different coordinate systems. Note that the omnicamera coordinate system is centered on its own virtual perspective.

Another very important aspect that must be considered during the coordinates conversion is the different rotation of the cameras (picture 5.2).
As we said in (chapter 3.1) the roll and the yaw are the same in both the cameras, but the pitch isn't. The Kinect sensor is slightly tilted (0.25 rad) toward the ground to better catch the obstacles in front of the wheelchair. We must take account of this difference while converting the coordinates from the Kinect to the Omnidirectional system.
Luckily, this matter is simply taken care of by using the trigonometry theory that stands behind the pitch difference among the cameras.
Let's imagine for a moment that the Kinect sensor was parallel to the ground

(pitch value 0 rad), now let's tilt it a little toward the ground: by doing this rotation, the coordinate system of the Kinect sensor will tilt as much as the camera itself and the z-axis and y-axis will measure different values if compared with the values measured from the previous sensor's position.
Even though the pitch angle is small, is not a good idea to ignore this rotation: the error made in the conversion would be way too unbearable, bringing to large errors in the conversion. Since the pitch rotation, like the roll and the jaw, involves only two axis, the following steps tell us what we must do to convert the direction of a point seen by the tilted Kinect to the direction of the very same point seen by a non-tilted Kinect in the same location:

- X-axis: nothing changes, no operations needed;
- Y-axis: by looking at the picture (Figure 5.2(a) and Figure 5.2(b)), you can see that the new registered value can be seen as the hypotenuse of a right-angled triangle in which we'd like to know the length of the vertical side. This is simply accomplished by multiplying the recorded value by the cosine of 0.25 radians.
- Z-axis: in here we must follow the same logic applied to the Y-axis above. This time, as we can see from the picture (same as before) the registered value from the Kinect is the length of the inclined side and we must find out the length of the hypotenuse. Thanks to the trigonometry theory we know that this can be accomplished just by dividing the registered value by the cosine of 0.25 radians.

Mathematically speaking, the Kinect coordinates pitch leads to these three equations:

$$\begin{aligned} Kinect\_Coord_x &= Kinect\_Coord_x \\ Kinect\_Coord_y &= Kinect\_Coord_y * \cos(0.25) \\ Kinect\_Coord_z &= \frac{Kinect\_Coord_z}{\cos(0.25)} \end{aligned} \tag{5.4}$$

Since this conversion changes the values of the direction a lot, we should do this pitch conversion first, and only afterwards we are free to proceed with the offset conversion introduced above.

5.3 The cameras communication

Once we understood how to convert one point from one camera to the other, it is time to use this new knowledge in practice into our project.
As we've seen already (section 1.1), the first step in our solution is the people

Fig. 5.3: First step of the camera communication

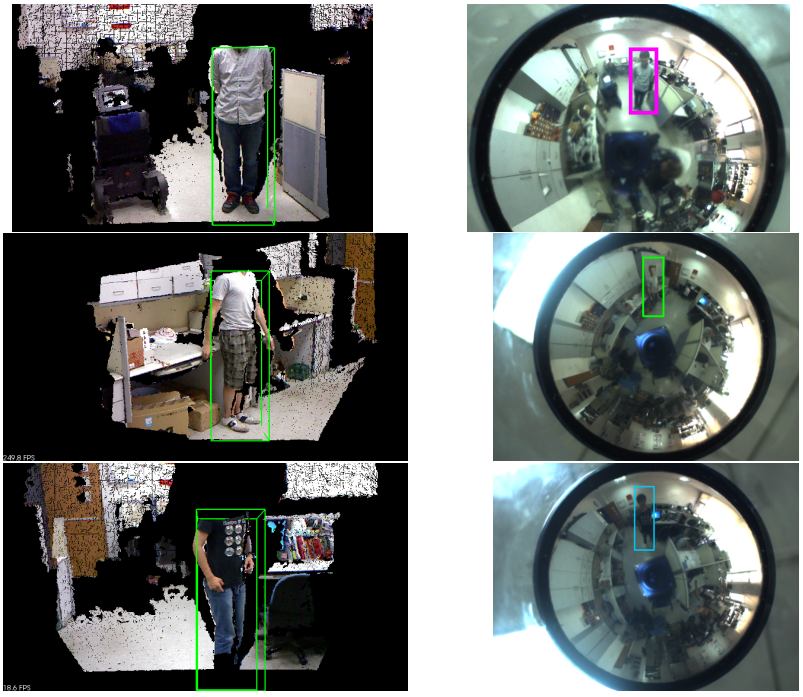


Fig. 5.4: Examples of the second step of the camera communication

detection with the Kinect camera: once the person is detected, the Kinect node defines a bounding box in the 3D world (the point cloud) that surrounds the person among all the three dimensions, as seen on figure 5.3 and described in section 4. The box, called bounding box, is composed by many 3D points that can be extracted for further developments: in our cases we extract the top and the bottom point of the bounding box. With just these two points we can obtain a lot of information from the omnicamera image frame and that's all we need from this Kinect node: once those two points are published, the Kinect people detection node will shut down, leaving the Kinect free to work with the navigation node.
Obviously this first step is not ended since we are still missing the people detection in the omnicamera frame: in order to detect the person, the first thing to do is reading the coordinates published by the Kinect node. Thanks to the conversion we just introduced (section 5.2), we can now obtain two points in the 2D image frame pretty quickly.
By considering those two points as two opposite corners of a rectangle we can quickly define a 2D bounding box surrounding the person in our camera frame. Since the top and the bottom of these 3D bounding boxes are usually vertically close, we enlarge the 2D bounding box in order to get a better detection of the person. Furthermore, because of the pitch of the Kinect camera that cuts most of the detected persons head off in the Kinect Image frame, we also expand the 2D bounding box vertically by 30cm: we have to do this because otherwise the 3D bounding box would not surround the tracked person head, leaving the real people detection a bit off. Anyway, this task is easily computed by adding 30cm to the top point published by the Kinect camera prior its conversion in the correspondent 2d omnicamera point.
Once we have the final 2D bounding box, it's time to proceed to the second step of the solution.

Chapter 6

People Tracking

Once the person has been detected by the Kinect camera and her position has been transferred from the Kinect node to the omnidirectional one, the next challenge, and our second project step (section 1.1), is to do the visual tracking of that person.
Most of the 2d image tracking algorithms are based on the description of the image region where the subject is detected, while many of them are based on the texture or edge histograms, the best results are obtained by those based on the colour histogram. [26] [15] [24] [13]
The histogram gives us a global statistic description of the target to track, losing by any means the spatial information of the target. Furthermore, the computation of an histogram from an image frame is pretty quick and easy to implement even in a real time application.
As intuitable, colour histograms are very resilient against change of size and movements of the target in the camera frame. The downside is that, by using only a single colour histogram, we can't distinguish two subjects with very similar colors.

6.1 Mean Shift

One possible approach is presented in [17], but the method that really changed the tracking algorithms with the use of colour histograms is the Mean shift [9]. Mean shift is a non-parametric feature-space analysis technique that give us a procedure for locating the maxima of a density function given discrete data sampled from that function (the histogram, in our case). Application domains of the mean shift include cluster analysis in computer vision and image processing. The simplest of such algorithms would create a confidence map in the new image based on the color histogram of the object in the previous image, and use the mean shift to find the peak of a confidence map near the object's old position. The confidence map is a probability density function of the new image, assigning each pixel of the

new image a probability, which is the probability of the pixel colour occurring in the object in the previous image.

The probability density of the subject, as known as the color histogram, is estimated by using a decreasing monotone kernel. The Kernel needs to sadisfy some general properties, as discussed in [11] e [6]. It's recommended to use the Kernel with Epanechnikov profile [7], defined in general as:

$$K_E(x) = \begin{cases} \frac{1}{2}c_d^{-1}(d+2)(1-\|x\|^2) & if \|x\| < 0 \\ 0 & otherwise \end{cases} \tag{6.1}$$

Where d is the number of dimensions of the space where we work in, c_d^{-1} is the d-dimensional unit sphere and x is the target point coordinates (pixel, in our case).

In our case the frame from a videocamera is a 2d image: the Epanechnikov's kernel formula becomes:

$$K_E(x) = \begin{cases} \frac{2}{\pi}(1-\|x\|^2) & if \|x\| < 0 \\ 0 & otherwise \end{cases} \tag{6.2}$$

The similarity between the target model and the target candidates in the next frame is measured using the metric derived from the Bhattacharyya coefficient [8]. In our case, the Bhattacharyya coefficient has the meaning of a correlation score. The similarity function defines a distance among target model and candidates. To accommodate comparisons among various targets, this distance should have a metric structure. We define the distance between two discrete distributions as:

$$p[p(y), q] = \sum_z \sqrt{p_z(y) q_z} \tag{6.3}$$

Where q_z indicates the target histogram and p_z indicates the histogram of candidate region of interest in the frame.

Given a set $\{x_i\}_{i=1\ldots n}$ of n points in the d-dimensional space R^d, the *multivariate kernel density estimate* with kernel $K(x)$ and window radius (bandwith) h, computed in the point x is given by

$$\hat{f}(x) = \frac{1}{n}\frac{1}{h^d}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{6.4}$$

It can be shown [8] that maximizing the Bhataccharyya's coefficient is equal

to maximizing the value of

$$\sum_{i=1}^{n} w_i K\left(\left|\frac{y - x_i}{h}\right|^2\right) \tag{6.5}$$

Where h indicates the kernel's bandwith length and the w_i parameters are computed as:

$$w_i = \sum_{u=1}^{m} \sqrt{\frac{\hat{q_u}}{\hat{p_u}(\hat{y_0})}} \delta[b(x_i) - u] \tag{6.6}$$

In which δ is the Kronecker's function and where $\hat{q}_u$ and $\hat{p}_u$ are the values of the bin u of the target's histogram and the candidate's histogram, respectively. $\hat{y}_0$ indicates the current position of the center of the candidate.

The vector, called Mean shift vector and seen in figure 6.1, is computed at every iteration and always points towards the direction of the maximum increase in the probability density of the target position in the current frame. It is computed as:

$$\hat{y} = \frac{\sum_{i=1}^{n} x_i w_i g\left(\left|\frac{\hat{y}_0 - x_i}{h}\right|^2\right)}{\sum_{i=1}^{n} w_i g\left(\left|\frac{\hat{y_0} - x_i}{h}\right|^2\right)} \tag{6.7}$$

where the function $g(\dots)$ indicates the derivative function. Since the Epanechnikov's kernel is a constant, the derivative function is quite simple and it is reduced to a simple weighted average:

$$\hat{y} = \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i} \tag{6.8}$$

In conclusion, it is possible to reach a local maximum by iterating the process multiple times. It's worth noting that the module of the mean shift vector is longer when the starting position of our current centroid is far from the local maximum, and it decreases as the centroid gets closer to it. This condition is necessary for the procedure to stop [7].

Therefore, the mean shift procedure, as know as the procedure that will locate a local maximum in our density by starting from a point x, called centroid, can be summarized in these three steps:

1. Computation of Mean Shift vector starting from the centroid;

2. Translation of the search window in the quantity defined by the Mean Shift vector;

3. If the mean shift vector's length is lower then a certain threshold, the procedure ends, otherwise we re-start again from the step 1;

6.2 Mean Shift Example

The intuition behind the meanshift is simple. Consider you have a set of points. (It can be a pixel distribution like histogram backprojection). You are given a small window (may be a circle) and you have to move that window to the area of maximum pixel density (or maximum number of points). An example is illustrated in the simple figure 6.1.

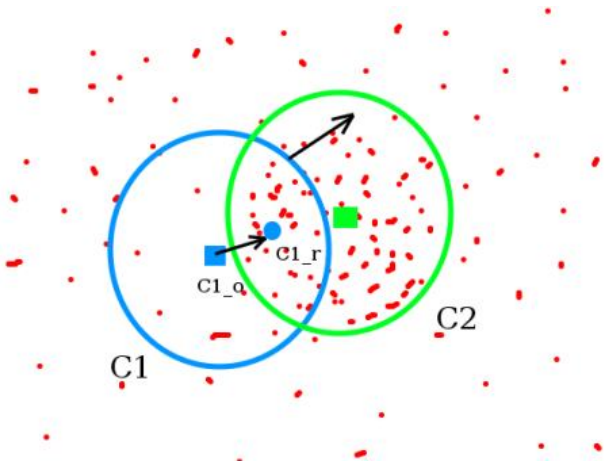


Fig. 6.1: Example of the Mean Shift algorithm in action

The initial window is shown in blue circle with the name "C1". Its original center is marked in a blue rectangle, named "C1_o". But if you look fo the new centroid of the points inside that window, you will get that the point "C1_r", marked in small blue circle, is the real centroid of window. Surely they don't match at the moment. So in this step we move our window such that circle of the new window matches with the centroid just spotted. Now, again find the new centroid. Most probably, it won't match. So move it again, and continue the iterations such that center of window and its centroid falls on the same location (or with a small desired error). Finally what you obtain at the end of the algorithm is a window with maximum pixel distribution: in figure 6.2(a) this window is marked with a green circle, named "C2".
When the object moves, obviously the movement is reflected in histogram backprojected image. As a result, the meanshift algorithm moves our window to the new location with maximum density.

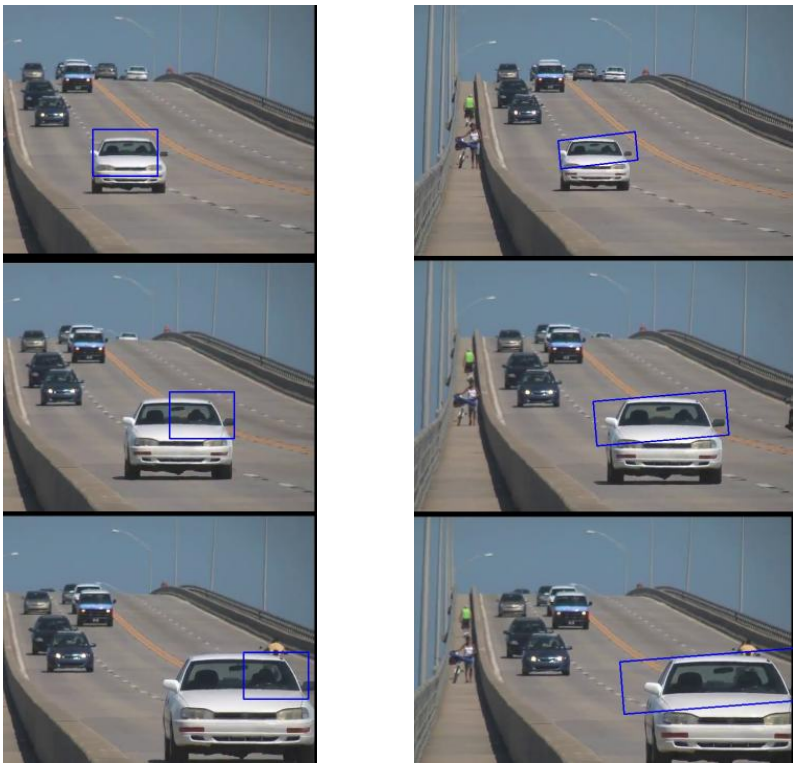
6.3 CamShift

The drawback of the mean shift procedure is that it is not dynamic at all, infact the Mean Shift algorithm works well on static probability distributions but not on dynamic ones like a movie or a video streaming. For this kind of meta data we need an algorithm that has a window that adapts with the size and the rotation of the target. This is why we have to look for a more complex solution, solution that is found in an algorithm called Camshift (Continuously Adaptive Meanshift) [5].
Camshift's is able to handle dynamic distributions by readjusting the search window size for the next frame based on the zeroth moment of the current frames distribution. It also computes the orientation of the best fitting ellipse to it. This allows the algorithm to anticipate object movement and to quickly track the object in the next scene. Even during quick movements of an object, Camshift will be able to correctly track said object.
Note that this method is not a completely new one but just an expansion of the Meanshift one (as the name suggested): the new algorithm still works by tracking the histogram of the subject as in its predecessor.
The Camshift is implemented as such:

1. Initial location of the 2D search window was computed.
2. The color probability distribution is calculated for a region slightly bigger than the mean shift search window.
3. Mean shift is performed on the area until suitable convergence. The zeroth moment and centroid coordinates are computed and stored.
4. The search window for the next frame is centered around the centroid and the size is scaled by a function of the zeroth movement.
5. Go to step 2.

6.4 CamShift example

There is very little to say more about the Camshift: It applies meanshift first. Once meanshift converges, it updates the size of the window as, $s = 2 \times \sqrt{\frac{M_{00}}{256}}$. It also calculates the orientation of best fitting ellipse to it. Again it applies the meanshift with new scaled search window and previous window location. The process is continued until required accuracy is met.
An example can be seen in figure 6.2(b)



(a) (b)

Fig. 6.2: Example of the differences between Mean Shift and Cam Shift algorithms on action: Figure 6.2(a) shows us the MeanShift behaviour, while figure 6.2(b) shows us the CamShift algorithm behaviour

Chapter 7

Motion control

In the last few chapters we have introduced the first two steps, out of three of our algorithm: we described how the machine detects a person (step 1, chapter 4), how the sensors tracks her (step 2, chapter 6), and how both sensors work together (step 1 again, chapter 3 and 5).
At this point, as our last step, what we need to do is converting all the information gained from the sensors into an actual action taken by our machine in order to accomplish the main goal of this project: allow a natural conversation to take place between a person and the smart wheelchair user without the need to control the wheelchair during the length of the conversation.

7.1 Linear & Angular Speed

Once the person is being tracked by the omnidirectional camera, the machine always knows, or at least has an estimation of, where the person is. Remember that, since the DC motors of the wheelchair allow a maximum translational speed of 500 mm/s (1.8 km/h, a little over 1 mile per hour), we must assume that the person to *follow* doesn't run or goes too fast. By doing so, the only other thing that we have to take into account is the direction of the wheelchair movement at any given point of time. The goal is to have the wheelchair user and his partner to be by each other side: the easiest and ideal thing to do is to send a new command, with the new wished linear speed (velocity) and angular speed (direction), at every new frame received from the omnicamera. Since our algorithms usually take more than a few milliseconds to compute, we implemented a buffer in which we keep only the latest omnicamera frame received. By doing so, we dump older unused frames received during the last computation and we compute only the most recent one at our disposal. Obviously this will make the work of the Camshift algorithm (section 6.3) harder, but it's something that we must give in for real-time's sake.
Note that, thanks to our people's tracking algorithm, there will not be any

problems even if the person changes direction suddenly or, eventually, slowly. In practice, the outcome of our method replaces the human-controlled joystick (or the other command interfaces, all introduced in section 2.3) during the whole conversation.

7.1.1 Linear speed

As introduced earlier, the maximum speed of the wheelchair is rather slow. When the wheelchair is at its top speed, the machine is as fast as a human being walking at a normal pace: what we must take care of is to make sure that the change of the linear speed is as smooth as possible, this is accomplished by adjusting the liner speed commands carefully at every change of state (we will introduce the concept of "state" in an dedicated upcoming section) in order to let the wheelchair user to have a pleasant ride.

7.1.2 Angular speed

The only thing left to do now is to determine how to choose the direction of the wheelchair movement properly: we know the exact position of the person in the omnicamera frame at any given time, obviously, we also know the exact position of the omnidirectional camera in the wheelchair.
Thanks to these pieces of information, we can estimate how far the person is from the smart wheelchair.

Target Distance

While the distance estimation of any given point from the omnicamera frame is a nearly impossible to compute without any further assistance, we can easily find out the distance of any given point if we are assured that the chosen point lies on the ground, called Ground Plane in picture 7.1. With this hypothesis and by knowing that the omnicamera's vertical axis is perpendicular to the ground, the task of estimate this distance is quite easy thanks to, again, the trigonometry theory:
The height h of the omnicamera is obviously known and constant, by knowing the direction p (please note that thanks to our omnicamera we know only the direction, not the distance of each one of the x,y,z axes: our p vector is always a unit vector) of the bottom point of the person in the camera frame we can compute the person distance from the wheelchair as follow:
Let's call p = [x,y,z] the 3D unit vector that emanates from the effective point of view towards the desired ground point (please refer to figure 7.2). To obtain our distance we have to separate the problem in three phases. Note that this is only one of the many ways that lead to the same result:

Phase 1

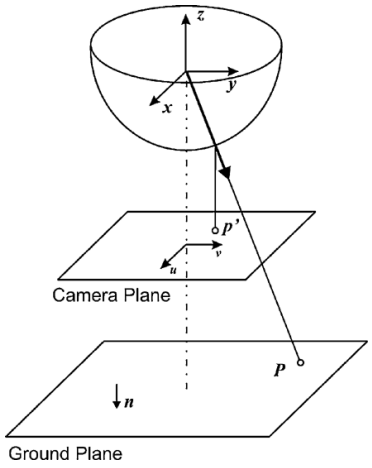


Fig. 7.1: Omnicamera planes

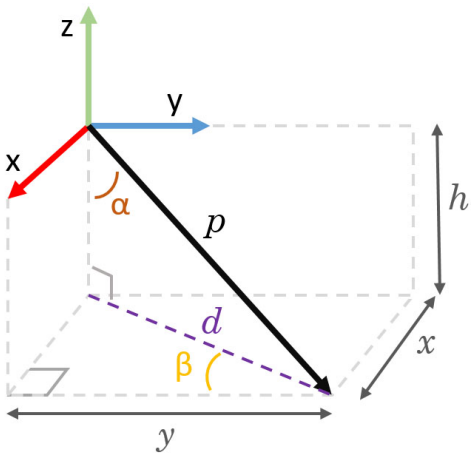


Fig. 7.2: Example of 3D vector with all its characteristic properties

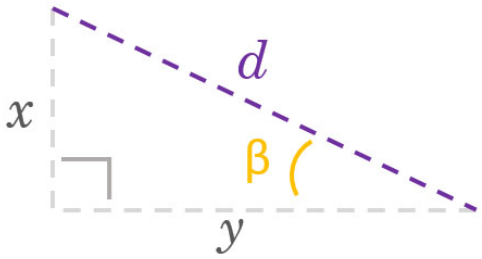


Fig. 7.3: Detail of 7.2 concerned in this phase. Please notice that instead of d we are computing the distance l, that is in the same place as d but much shorter.

In this phase we are going to compute the length of the triangle's hypotenuse formed by the x and the y edge. Let l be that length, then this can be computed by resolving the system of equations:

$$\begin{cases} y = l * \cos(b) \\ x = l * \sin(b) \end{cases} \tag{7.1}$$

Obtaining:

$$l = \frac{y}{\cos\left(atan\left(\frac{x}{y}\right)\right)} = y\sqrt{1 + \frac{x^2}{y^2}} \tag{7.2}$$

Remember that this length l is NOT the same length as our desired distance d, because our values of x and y refer only to the 3D unit vector (the figure may make you think otherwise).

Phase 2

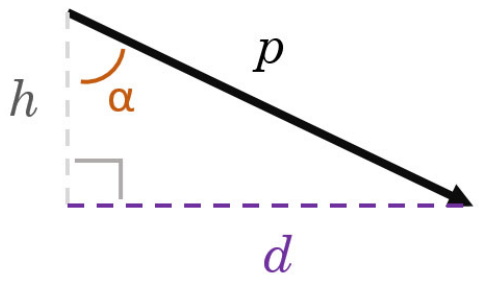


Fig. 7.4: Detail of 7.2 concerned in this phase. Please notice that, instead of h, we now use z, instead of d we now use l, and instead of p we use $\|p\|$

In this phase we are going to compute the vertical angle α of our 3d unit vector p:
Again, thanks to the trigonometry theory, the angle is obtained by resolving the system of equations

$$\begin{cases} z = \|p\| * \cos(\alpha) \\ l = \|p\| * \sin(\alpha) \end{cases} \tag{7.3}$$

Obtaining

$$\alpha = arctan\left(\frac{l}{z}\right) \tag{7.4}$$

Phase 3

At this point we have both the height h of the effective point of view

of the omnidirectional camera, measured in real life with a meter tool, and the vertical angle α, just computed in the last phase: nothing can now stop us to use the trigonometry theory again to finally compute the desired distance d.

$$\begin{cases} h = \|p\| * \cos(\alpha) \\ d = \|p\| * \sin(\alpha) \end{cases} \tag{7.5}$$

Obtaining

$$d = h * \tan(\alpha) = h * \frac{l}{z} = h * \frac{y}{z} * \sqrt{1 + \frac{x^2}{y^2}} \tag{7.6}$$

Where we used the result from phase 2 in the first step and the result from phase 1 in the second step.

Target Goal

Another fundamental aspect for defining what the next speed command will be is the position of the actual bottom point of the person in comparison with its wished position: by doing some tests we found out that the best position for the bottom point in order to let the users have a normal talk is slightly on the left (or on the right) of the back of the wheelchair, resulting in having the machine always around 40cm away from the walking person.

States

Now that we have introduced how we compute the distance and how we set the goal, it is time to introduce the wheelchair states, states that actually define the linear and angular speed of the robot.
All the wheelchair states are based on the different position that the actual bottom point of the person assumes in comparison with its target position. At any given moment the wheelchair will assume one of states showed in figure 7.5: the wheelchiar will change state every time the actual bottom point lies on another of those enlightened areas.
The first and easiest state, colored purple in the figure 7.5, is the one based on the user distance: if the person bottom point is less than 20 cm away from the target goal, then this state tells the wheelchair to simply go straight ahead, when the tracked person is in this position we can assume that the conversation is happening smoothly.

The other four different states are based on the position of the person, two of which are further split in two stops, and are defined in order to make the smart wheelchair to be by the walking user side:

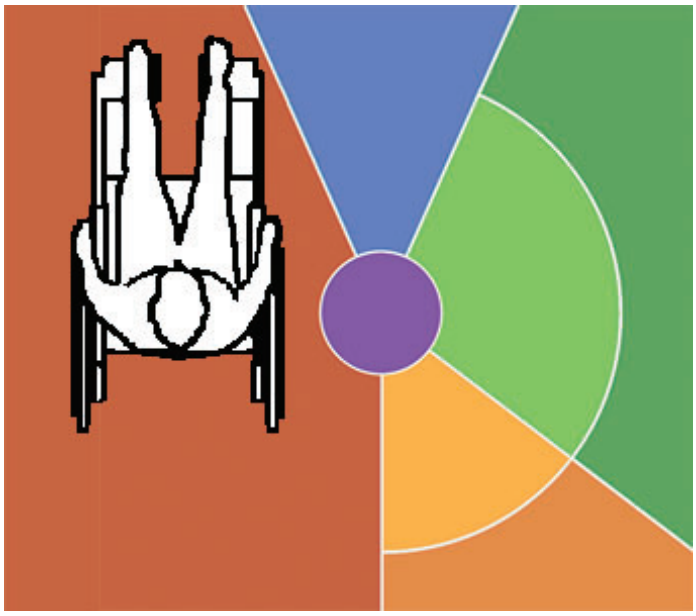


Fig. 7.5: Smart Wheelchair states, remember that these states are defined by the bottom point of the walking user.

- When the person is behind on the right side of the target goal, stop the wheelchair and let the person reach you. If the person is far behind on the right, stop and turn right. Those two states are colored respectfully light and dark orange in the figure 7.5;
- When the person is slightly in front of the wheelchair on the right or on the right side of the target goal, go straight ahead. If the person is far, stop and turn right. Those two states are colored respectfully light and dark green in the figure 7.5;
- When the person is in front of the target goal, just go straight ahead. The state is colored blue in the figure 7.5;
- If the person is on the left side of the goal, stop and turn left. This state is colored brown in the figure 7.5;

Obviously those are the states when the goal is set on the right of the wheelchair, the same way of thinking is applied when the goal is set on the left of the wheelchair, but with all the cases mirrored on the image frame.

From the coding perspective of view, these states are defined by the position and inclination of the imaginary straight line that connects the bottom point goal with the actual bottom position: aside the distance-only-dependent state, to disclose in which state the wheelchair is, we must compute this

inclination and compare the positions of the two points (actual position and goal position). Once we understood if the actual bottom point is on the left or on the right of the goal position, we can detect in which state the machine is by comparing the inclination of the straight line introduced above with some predefined inclinations that we experimented: since all our cases together surround the bottom point goal completely, the wheelchair will lie in one of those states at every single update.
Because we are working with humans, when a change of state happens, this doesn't mean that the wheelchair will adjust the direction or stop suddenly in order to reflect the new state: every change will see a smooth transition of the linear and angular speed, in the interest of preserving the comfort of both the wheelchair user and the walking person.

7.2 Obstacle Avoidance

Now that we are able to send commands to the wheelchair, we should take care of one final detail before wrapping up: what if there's an obstacle in our route? As stated previously, we use the omnicamera only to track the person, while the Kinect is doing the obstacle avoidance. This means that our omnicamera node doesn't know anything about the environment. This problem is resolved by our shared control (chapter 2.3.2): in fact, thanks to the safety index, our robot will never hit any obstacle and will, eventually, circle around the tracked user in order to move the goal from one side to the other of the wheelchair.

7.2.1 Change of Goal

As said in the previous paragraph, the omnicamera node doesn't have any information about the environment. To overcome this weakness, our obstacle avoidance node publishes information on when there are too many obstacles on the left or on the right in front of the wheelchair: these messages will tell the omnicamera node, that listens to this specific topic, when it is time to swap from the left-positioned goal to the right-positioned one or viceversa. Obviously, we pay attention to these messages only when the wheelchair is moving straight ahead, otherwise we simply ignore them.

7.3 System Evaluation

In this chapter we'll show the performance of the motion control, that is what matters from the client point of view.

7.3.1 Introduction to the tests

To do this evaluation we studied two major points: how well the smart wheelchair reach the target person and how well the wheelchair stays by the target side during the conversation.
In order to do this evaluation we had to change the code a little, making the system's performance worse than the real scenario. Nevertheless, the machine response was more than positive as shown in the next sections.
The first test records the target position once the wheelchair has reached it, while the second test record the target position while the wheelchair is following it, computing a target position chronology very useful for studying it later.

7.3.2 Wheelchair position after reaching the target

This first test records a data set of the position of the target once the wheelchair has reached it. We did this test several times, with the different person starting positions:

- 1,5 m in front of the robot, 25° on the left.
- 1,5 m in front of the robot.
- 1,5 m in front of the robot, 25° on the right.
- 3 m in front of the robot.

Results

These positions were chosen in order to the target to be visible from the Kinect camera. The recorded position of the target is taken 500 milliseconds after the target has entered the purple state of figure 7.5, described in section 7.1.2.
The figure 7.6 represents the Heatmap of our dataset: as we can see the person position on average is exactly where it is supposed to be, remember that once the wheelchair reaches the person, she should start moving in order to explore the environment and have the talk with the wheelchair user.

7.3.3 Wheelchair position after reaching the target

This second test records a data set of the target positions after the wheelchair has reached it. While the first test is just a snapshot of a precise moment, this second test is an mean average of the target position during the talk. Even in this case we did the test several times, with different starting positions of the target person:

- 1,5 m in front of the robot, 25° on the left.

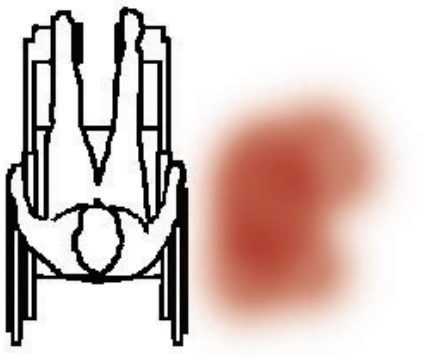


Fig. 7.6: Heatmap of the target position after the wheelchair has reached it.

- 1,5 m in front of the robot.
- 1,5 m in front of the robot, 25° on the right.
- 3 m in front of the robot.

results

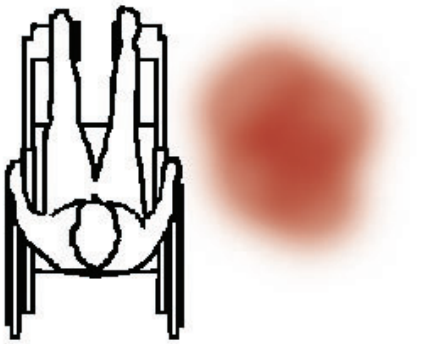


Fig. 7.7: Heatmap of the target position while the wheelchair is following it.

For an introduction on how to read the graph please reference to the previous subsection.
In this second test we can clearly see that our algorithm takes feedback actions: the wheelchair user is on average slightly behind the target user especially when he turns away from the wheelchair. Aside that, this algorithm is an excellent starting point and removes the hassle to control the wheelchair to its user.

An algorithm improvement could be an integration of a target user face direction detection in order to predict where the target is heading to, obtaining a foreseeing algorithm that will have a even better feedback action.

Chapter 8

Conclusions

In this thesis we presented a complete work, started from scratch, on how a smart wheelchair may allow a natural conversation to take place between anybody and a smart wheelchair user without the need to interact with the wheelchair during the whole length of the conversation. We have used many technologies: some new and some old, but their collaboration brought in the table completely new technologies that can empower future works in many fields.
We also have encountered many problems that made us to implement new technologies, especially the collaboration between the omnidirectional camera and the Microsoft Kinect sensor, that opened a wide range of future implementations of this kind in several other projects.

Obviously, this work could be further improved in many aspects, such as:

Implementation of an improved tracking algorithm (chapter 6)

- Camshift is just the standard tracking algorithm: by implementing something more specific to our needs (maybe a people tracking algorithm made exactly for using it with an omnidirectional camera) the tracking experience could be improved even further as this could also means less errors during the tracking phase;
- Missing person catch up: what if the person cannot be seen in the omnicamera frame anymore? This can happen, for example, when the tracked user walks around a corner and our wheelchair is far behind it. In this case the robot should figure out what happened and take action accordingly;
- If we don't want to change the Camshift algorithm, a nice improvement would be the update of the color histogram at every computed frame: this way the algorithm would work better when the environment light conditions change;

Smarter motion control

- By implementing an algorithm for the face direction detection of the tracked user, the smart wheelchair could use this new information to change its behavior and to improve the overall experience of the upcoming and/or ongoing talk;

Implementation of new sensors

- the second-generation of the Kinect sensor is better in every aspect in comparison with its predecessor and, especially, it has a wider angle of view both horizontally and vertically. This could bring to faster and more precise people detection, along other things such as better obstacle avoidance and further maximum distance for detecting obstacles. A better omnidirectional camera is much more needed, since the colors of the current one tend to be blueish in any given condition, bringing to an inefficient color histogram and, consequently, an inefficient tracking algorithm;

Better people bag for the Kinect node

- In the case that the Kinect sensor wouldn't be updated as suggested earlier, the Kinect node would benefit of a better people bag for the people detection: the people bag is something that our algorithm needs to understand what a person is. The bag contains a number of examples of people in different positions, so that the algorithm can try to identify new people in any point cloud given. Since the first generation of Kinect mounted on the wheelchair wouldn't see the heads of the people anyway, a people bag with the people's head cut off would result in a faster and more suitable algorithm for the project. Obviously this is not the case anymore if it will be decided to implement in the new generation of the Kinect sensors in the wheelchair.

Bibliography

[1] D. Bell. Modeling human behavior for adaptation in human machine systems. *Ph.D. dissertation, Univ. Michigan*, 1994.

[2] J. Borenstein and Y Koren. Histogramic in-motion mapping for mobile robot obstacle avoidance. *Robotics and Automation, IEEE Transactions on (Volume:7 , Issue: 3)*, pages 535–539, 1991.

[3] J. Borenstein and Y Koren. The vector field histogramâĂŤfast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on (Volume:7 , Issue: 3)*, pages 278–288, 1991.

[4] Adrian Bradski. *Learning OpenCV, [Computer Vision with OpenCV Library ; software that sees]*. O'Reilly Media, 1. ed. edition, 2008. Gary Bradski and Adrian Kaehler.

[5] G.R. Bradski. Real time face and object tracking as a component of a perceptual user interface. *Proceedings of Fourth IEEE Workshop on Applications of Computer Vision*, pages pp.214,219, 1998.

[6] D. Comaniciu. Non parametric robust methods for computer vision. *PhD Thesis, The State University of New Jersey*, pages 5–17, 2000.

[7] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, pages 603–619, 2002.

[8] Comaniciu D., Ramesh V., Meer P. Real-time tracking of non-rigid objects using mean shift. *Proceedings. IEEE Conference on Computer Vision and Pattern Recognition*, pages 142–149, 2000.

[9] P. Meer D. Comaniciu, V. Ramesh. Kernel-based object tracking. *Proceedings of IEEE International Conference on Pattern Analysis and Machine Intelligence*, pages 564–575, 2006.

[10] David A. Bell, Simon P. Levine, Yoram Koren, Lincoln A. Jaros, Johann Borenstein. Design criteria for obstacle avoidance in a shared-control system. *Proc. RESNA Int. Conf., Washington, DC*, page 581âĂŞ583, 1994.

[11] L. Hostetler K. Fukunaga. The estimation of the gradient of a density function, with application in pattern recognition. *IEEE Transactions on Information Theory*, pages 32–36, 1975.

[12] Robert Laganière. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing, May 2011.

[13] Parag Batavia Marin Kobilarov, Jeff Hyams and Gaurav S. Sukhatme. People tracking and following with mobile robot using an omnidirectional camera and a laser. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 557–562, 2006.

[14] Marin Kobilarov, Jeff Hyams, Parag Batavia and Gaurav S. Sukhatme. People tracking and following with mobile robot using an omnidirectional camera and a laser. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 557–562, 2006.

[15] Matsumura, A., Y. Iwai, and M. Yachida. Tracking people by using color information from omnidirectional images. *Proceedings of the 41st SICE Annual Conference*, pages 1772–1777, 2002.

[16] Matteo Munaro, Filippo Basso and Emanuele Menegatti. Tracking people within groups with rgb-d data. *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[17] P. Perez, C. Hue, J. Vermaak, M. Gangnet. Color based probabilistic tracking. *Springer-Verlag Berlin Heidelberg*, pages 665–670, 2002.

[18] Qinan Li, Weidong Chen, and Jingchuan Wang. Dynamic Shared Control for Human-Wheelchair Cooperation. *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[19] Rufli, M., Scaramuzza, D., and Siegwart, R. Automatic Detection of Checkerboards on Blurred and Distorted Images. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.

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

[21] Scaramuzza, D. Omnidirectional Vision: from Calibration to Robot Motion Estimation. *ETH Zurich, PhD Thesis no. 17635. PhD Thesis advisor: Prof. Roland Siegwart. Committee members: Prof. Patrick Rives (INRIA Sophia Antipolis), Prof. Luc Van Gool (ETH Zurich). Chair: Prof. Lino Guzzella (ETH Zurich)*, 2008.

[22] Scaramuzza, D., Martinelli, A. and Siegwart, R. A Flexible Technique for Accurate Omnidirectional Camera Calibration and Structure from Motion. *Proceedings of IEEE International Conference of Vision Systems (ICVS)*, 2006.

[23] Scaramuzza, D., Martinelli, A. and Siegwart, R. A Toolbox for Easy Calibrating Omnidirectional Cameras. *Proceedings to IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[24] M. Yachida T. Mituyosi, Y. Yagi. Real-time human feature acquisition and human tracking by omnidirectional image sensor. *IEEE International Conference on Fusion and Integration for Intelligent Systems, MFI2003*, page 258 âĂŞ 263, 2003.

[25] Takashi Yoshimi, Manabu Nishiyama, Takafumi Sonoura, Hideichi Nakamoto, Seiji Tokura, Hirokazu Sato, Fumio Ozaki, Nobuto Matsuhira, Hiroshi Mizoguchi. Development of a Person Following Robot with Vision Based Target Detection. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5286–5291, 2006.

[26] Teixeira T. and Savvides, A. Lightweight people counting and localizing in indoor spaces using camera sensor nodes. *ICDSC, IEEE*, pages 36–43, 2007.

[27] Norman Villaroman, Dale Rowe, and Bret Swan. Teaching natural user interaction using openni and the microsoft kinect sensor. In *Proceedings of the 2011 conference on Information technology education*, SIGITE '11, pages 227–232, New York, NY, USA, 2011. ACM.

Appendix A

Omnidirectional camera 101

A.1 Effective Viewpoint, Virtual Perspective Camera

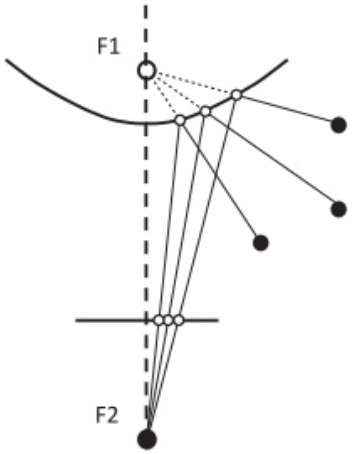


Fig. A.1: Graph showing the view points of the omnicamera.

The Omnidirectional camera's parabolic optics ensure that it has a single effective center of projection, known as Effective Viewpoint or as Virtual Perspective Camera, and also that this point is the single point through which all rays from a scene must pass on their way to the camera's lens.

A.2 Intrinsic and extrinsic parameters

It is quite important to estimate the objects position in the real 3D world from the 2D image obtained by the omnicamera: to allow this we need to know all the intrinsic and extrinsic parameters of our camera.

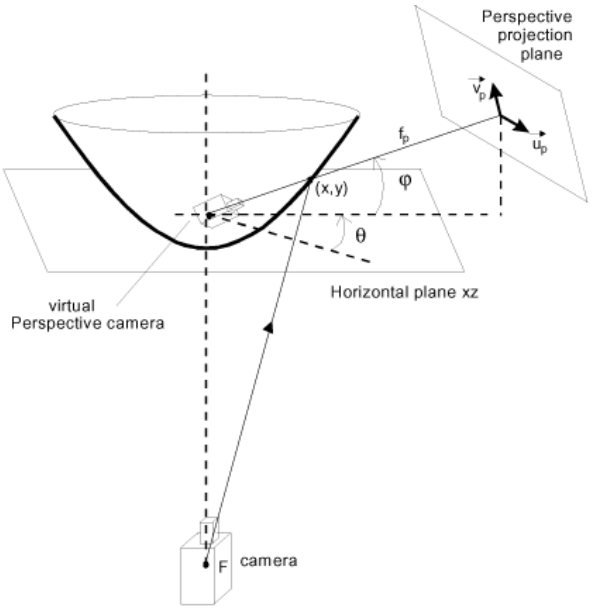


Fig. A.2: Virtual perspective projection camera at the focus of hyperbole

The intrinsic parameters, as the name reveals, are those parameters defined within the camera, among them we have:

- f, Focal length (zoom);
- o_x,o_y: coordinates of the optical center;
- s_x, s_y: sensor pixel size in mm;
- k_1,k_2: radial distortion coefficients;

These parameters allow us to map the geometric coordinates in the real world with the pixels in the digital frame. The extrinsic parameters, instead, are those parameters that tell you the position and orientation of the omnidirectional camera in the real world.