CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF ELECTRICAL ENGINEERING

MASTER'S THESIS

Bc. Šimon Fojtů

Nao Localization and Navigation Based on Sparse 3D Point Cloud Reconstruction

Department of Cybernetics
Thesis supervisor: **Mgr. Michal Havlena**
Prague, 2011

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze, ...

podpis

Abstrakt

Cílem Evropského projektu s názvem HUMAVIPS je vybavit humanoidy, konkrétně robot Nao, audiovizuálními schopnostmi za účelem zdokonalení možností interakce a komunikace mezi lidmi a roboty. Příspěvek této diplomové práce k tomuto odvážnému úkolu spočívá v navigaci a lokalizaci robotu pomocí 3D rekonstrukce okolního světa z obrázků pořízených robotem. Nástroj na zpracování obrázků SfMSeqv2, vyvinutý v CMP, je modifikován pro tuto úlohu a úspěšně použit pro výpočet struktury z pohybu (SfM) v reálném čase. Rekonstruované pozice kamer slouží pro lokalizaci robotu v prostoru a kombinace rekonstrukce scény s lokalizací umožňuje Simultánní Lokalizaci A Mapování (SLAM). Několik problémů spojených s robotem je popsáno spolu se způsobem jejich řešení.

Abstract

The goal of an European project called HUMAVIPS is to endow humanoids, robot Nao in particular, with audio visual capabilities, in order to improve the human–robot interaction and communication possibilities. This thesis contributes to this challenging task by focusing on robot localization and navigation, employing the visual input for 3D scene reconstruction from images captured by the robot. A state of the art image processing pipeline SfMSeqv2 developed at CMP is modified for this purpose and successfully used for real-time Structure from Motion (SfM) computing. The reconstructed camera poses serve as a localization mechanism and the combination of localization and 3D scene reconstruction results in Simultaneous Localization And Mapping (SLAM). Several hardware specific issues encountered are tackled and described.

I thank my thesis supervisor Mgr. Michal Havlena for his valuable advices and all help I have received from him. I also thank Ing. Tomáš Pajdla Ph. D. for his comments and ideas. Last but not least I thank my wife Štěpánka for her support and patience.

Contents

List of Figures

Chapter 1

Introduction

Mankind uses tools since very long time ago. Firstly, these were just simple levers, wedges, wheels, or inclined planes, but the complexity of such instruments grew gradually and as the development continued, the society started to change as well. Electricity became part of our lives and there are many that cannot imagine their lives without it. Instead of single purpose instruments, there are now complex devices that incorporate various functionality, e.g. mobile phones. Also robotics evolves rapidly, from robotic arms in factories to mobile robots semi–autonomously exploring the depths of space or seas. The development introduces machines more and more similar to human beings [12]. The visual appearance seems to be the simplest to mimic, even though the task itself is not simple at all. Contemporary results on geminoids (teleoperated android of an existent person) as produced by Aalborg University, Denmark or those coming from Osaka University, Japan [21] show, that the development is very advanced already. This takes the human–machine interaction to a completely new level, as in some cases it might be difficult to visually distinguish geminoids from humans. Geminoids are able, to some extent, to express *feelings* by their posture or expression of face, thus communicate non–verbally. Both verbal and non–verbal ways of communication present a challenging task for robot designers and developers.

As the development of humanoid robots introduces more and more human-like machines, the need for natural robot–human interaction and communication is increasingly important. One of the long–term goals in this field is a humanoid robot on a cocktail party, where there are many people, communicating with each other, moving freely around, or creating groups. Such an event presents the robot's visual and acoustical sensors with an abundance of sources and distractions that the robot needs to distinguish and react accordingly. This is the *humble* goal of an European project called HUMAVIPS (introduced in Section 1.1).

1.1 Project HUMAVIPS

HUMAVIPS, which is an abbreviation for HUManoids with Auditory and Visual abilities In Populated Spaces, is an European FP7 project, which started in February 2010 and which is aimed at endowing humanoid robots with audiovisual abilities, such as exploration, recognition, and interaction, so that they exhibit adequate behavior when dealing with a group of people [8]. The project is focusing on integration of audio and visual signals to enhance and improve the robot–human interaction. Following paragraphs provide information about the goal of this project and the relation between the project and this thesis.

Humanoid robots that are to cooperate and collaborate with people, should do so in the most natural way possible (from the human point of view). To accomplish this, they need to coordinate motion, information processing, and communication. These high-level tasks can be further decomposed into less abstract as follows. Motion includes localization, reactive obstacle avoidance, planning, and control of actuators (joints) to execute plans. Sensing involves processing of information streamed from various sensors and of creating and updating the internal representation of the environment. In order to communicate with people, humanoids need to recognize which persons are engaged in conversation, what do they say, and implement some kind of gesture, pose, and face recognition to understand non-verbal communication. According to the information received, they should update their internal state, formulate eventual responses, and plan further actions.

Imagine a cocktail party, a scenario in which there are several groups of people or individuals and the humanoids should be able to explore the environment, recognize individuals, decide which group of people or single persons to approach, and to engage in conversation, while adjusting behaviour to blend into the society. While many humans appear to solve such tasks with ease, it still remains a very challenging problem for robot developers and programmers [14].

In order for the robot to be able to engage in conversation, it must at first be able to approach a particular person, navigate through the environment, and be able to plan trajectories to reach its desired targets. It should also react on dynamic changes in the environment and avoid obstacles that are blocking its planned path. These are just a few of the many tasks from the field of mobile robotics that are vital for any robot that is supposed to move around.

1.1.1 Scenarios

In this thesis there are considered three situations or scenarios, in which a mobile robot can easily find itself while performing various tasks. They can be perceived as a subset of necessary conditions for humanoid robots (or mobile robots in general) to successfully navigate in their environment. The list of scenarios does not attempt to fully span all the preconditions of successful mobile robot navigation, merely shows which ones were considered important for the further work on the HUMAVIPS project. Successful operation in these scenarios enables the robot (and its developers) to focus on more difficult and complex tasks that the robot needs to solve, i.e. face, gender, or speech recognition.

The scenarios are: (i) the scanning of the surrounding environment in order to create an internal representation of the world, which is afterwards used in other scenarios, (ii) localization of the robot pose with respect to the previously created representation. This allows the robot to recover the knowledge about its pose when it has fallen or was moved away from its original pose by someone. (iii) navigation, that aims at using information from scenarios (i) and (ii) to navigate the robot to a target position. By integrating localization into the navigation process, the path can be followed more precisely, as the robot can account for odometry errors and adjust the walk accordingly.

These scenarios include common mobile robotics problems, that need to be solved in the first place, before any other tasks can be performed. In this thesis a visual system is employed that uses images captured by a humanoid robot Nao and processes them to gain information about the surroundings, obstacles, and robot's pose. Similar approach is taken by Oßvald et al. [22], who were using camera pointed downwards, looking just ahead of Nao's feet, and employed visual feature detection with reinforcement learning to achieve reliable navigation. Another way of robot localization was used by Hornung et al. [13], who used a laser rangefinder to localize the robot.

The application of the implemented algorithms and systems is targeted at humanoid robot Nao, which is thoroughly described in section 1.2, but is not restricted to this particular robot only, as the presented approach is generally applicable to all mobile robots that are capable of visual perception. On the other hand, it is necessary to mention that although the used principles are generally and widely usable, many tasks which needed to be solved in this thesis were closely related to and often limited by the used hardware. This robot was not selected randomly for this thesis, but is used in the HUMAVIPS project as a testing and a proof of concept platform, improved on the fly along with the progress and results from HUMAVIPS project.

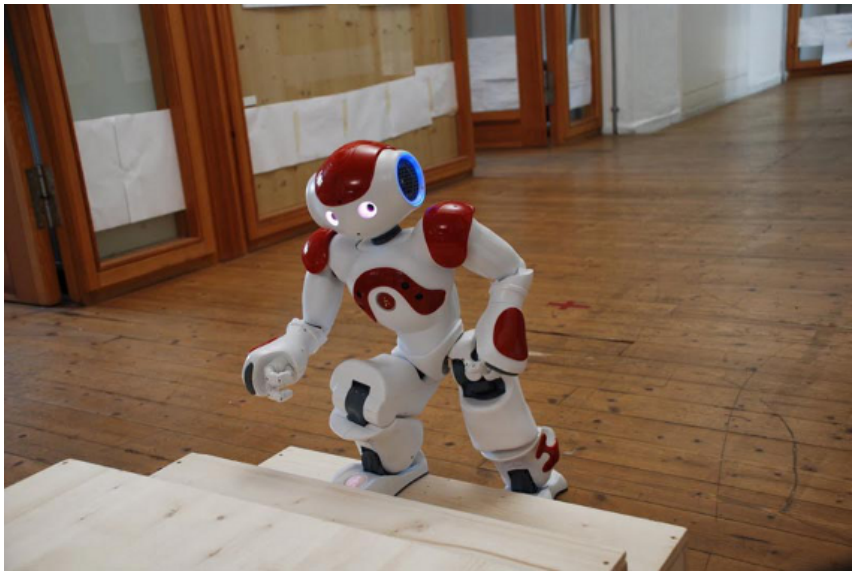


Figure 1.1: Robot Nao climbing stairs, courtesy of Humanoid Robotics Laboratory (University of Freiburg).

1.2 Robot Nao

Robot Nao is employed as a testing platform in project HUMAVIPS and this section tries to briefly describe the robot and its most important parts and functions from the point of view of the HUMAVIPS project. A more detailed description can be found in appendix A.

Humanoid robot Nao is manufactured by Aldebaran Robotics [1] from France. The development does not take place only in Aldebaran, but there are also some universities and other companies, that help with the development of specific components, e.g. laser ranger by University in Freiburg [13].

The robot is 58 cm tall and weights about 5 kg, thus compared to adults looks more like a child. In spite of its small dimensions, it is capable of performing quite complicated tasks. In order to perceive its environment, it is equipped with several sensors, such as two cameras, four microphones, two sonars, two bumpers, and an accelerometer to name a few [2].

To affect its environment and to be able to move around in it, the robot body has 27 joints (degrees of freedom), which allow it to walk in a human–like manner, except that its walk is *static*, whereas humans walk *dynamically*. Static walking consists of keeping the robot's center of gravity (CoG) vertical projection inside a convex polygon envelope (support polygon), bounded by parts of the robot that are touching the ground (usually feet). While static motion tries to overcome the influence of dynamics, in dynamic walking the

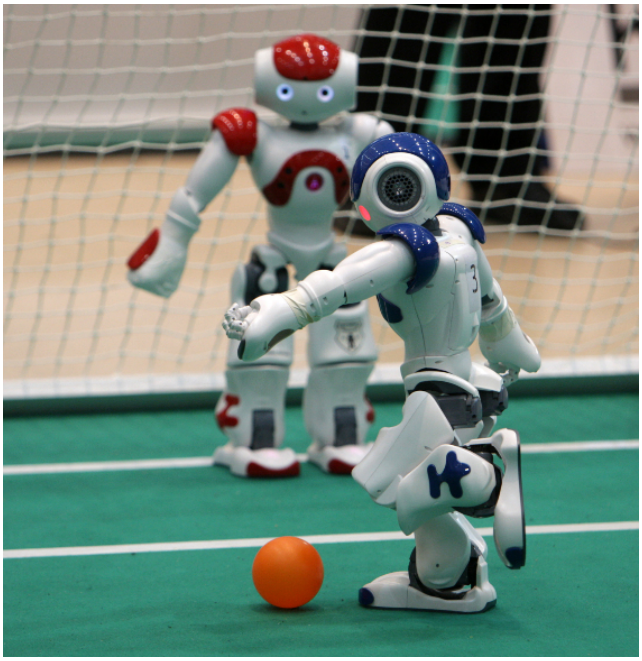


Figure 1.2: Robot Nao in Robocup competition, courtesy of Robocup.

body mass dynamics is used to improve the motion and results in more efficient, smoother, and faster walk. It could be described as a body falling in the direction of the desired motion and the legs trying to compensate this fall insofar that the desired velocity is maintained and the body does not fall [6].

Among the contemporary applications of robot Nao belong robotic soccer Robocup, where teams of humanoid robots compete in a simplified version of the well–known sport. Many universities use Nao (or Naos) for research or teaching purposes, aiming at various fields of humanoid robotics. It is also possible for individual researchers to use Nao for their development, as Aldebaran robotics is now offering the robot to a few best programmers from around the world at a reasonable price [1].

Regarding the future of this robot, Aldebaran Robotics envisions it as "an autonomous family companion", that would help with housework, play with children, or care for the elderly. Surely, limited by its size the robot is not of much help but the development of a 1.5 m tall robot, capable of such tasks, is already under way and the experience gained on development of Nao will definitely be useful.

It should be noted that the Nao robot is not yet a finished project and that the development of robot's hardware and software is continuing together with the HUMAVIPS project progress. This presents the project participants with

a constantly changing and upgrading environment, that needs to be accounted for in the project development process. Many of the HUMAVIPS participating groups are using their own hardware instead of the Nao robot to create and test their applications, which enables them to use more sophisticated methods and approaches for their research. In this thesis, on the contrary, the focus is mainly on the robot and the presented methods are trying to utilize the robot as much as possible despite all the drawbacks involved.

Finally, some pictures of the humanoid robot are presented in Figure 1.1, where the robot is climbing up stairs, and another in Figure 1.2, where there are two robots playing the robotic soccer.

Chapter 2

Resources

This chapter introduces resources that were available during the work on this thesis. It describes the most important hardware parts utilised, i.e. cameras and motion. The aspects of robot programming are discussed and lastly the state of the art sequential structure from motion image processing pipeline is presented.

2.1 Hardware

This section focuses on the robot Nao from the hardware point of view. Only those parts that are related to this thesis are discussed, for full description see Appendix A or Aldebaran online documentation [2]. In this thesis, mostly the vision system with a limited use of the motion (mainly walk) is employed.

2.1.1 Camera

For this thesis, the two of the most important pieces of hardware are the robot cameras. Their parameters and capabilities very much restrain the list of methods and options available for image processing. The following paragraphs give out the most important camera characteristics.

There are two almost identical CMOS VGA cameras placed on the Nao's head. From their physical location on head and vertical field of view (see Figure 2.1) it is apparent, that the images from the cameras do not overlap and thus cannot be used as a stereo rig for stereo vision. With the default head position, the top camera is looking forward and the bottom camera captures images of the floor right in front of the robot or its hands.

Field of view of the cameras is quite narrow—vertical field of view is 34.8° and horizontal 46.4°—thus the view of the environment from one pair of images

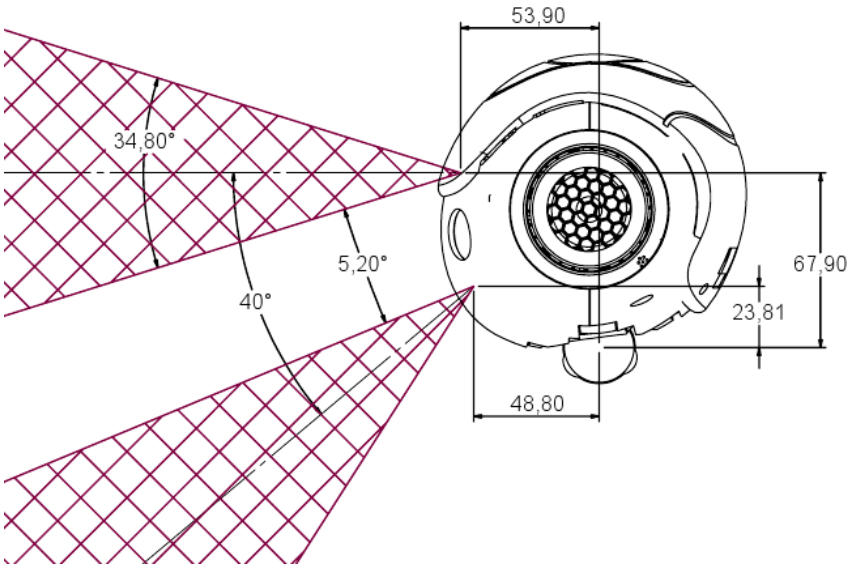


Figure 2.1: Position and orientation of cameras on Nao's head with respect to head coordinate frame. This configuration does not allow stereo vision, courtesy of Aldebaran Robotics [2].

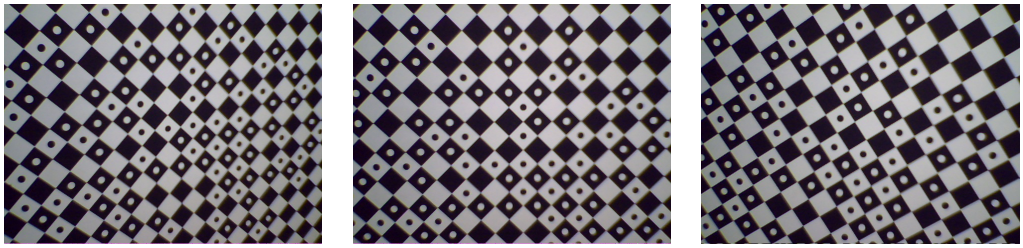
is very limited. Maximal resolution of the captured images is 640×480, which does not enable for detailed images. With lowest resolution, it is possible to capture up to 30 images per second, but with the highest resolution it is at most 3 images per second.

Since the camera chips are CMOS and there is no shutter, images that are taken while the robot head is moving (or while there are rapid scene changes) are composed of image rows from different time points. Such images can be used, e.g. for face recognition, where the difference of a few pixels is not a problem, but for 3D points and scene reconstruction, such images are not usable. Moreover, the images that are taken while the robot is moving are blurred, which renders them unusable for any reconstruction.

Both the camera lenses and the CMOS chips are tiny and thus great amount of noise influences the final image. Especially with the auto–gain of the camera turned on, which further increases the camera sensitivity.

Camera calibration

Internal camera parameters, that are used for transformation from image pixels coordinates to unit vector direction were calculated off–line [17] using images of



(a) (b) (c)

Figure 2.2: Calibration images captured by Nao's top camera. Pixel correspondences are possible due to specific dots layout.

calibration grid captured by each camera (see Figure 2.2). Internal calibration matrix of the top camera is as follows

$$\mathbf{K}_{top} = \begin{bmatrix} 749.1048 & 0 & 329.3791 \\ 0 & 750.2743 & 227.1093 \\ 0 & 0 & 1.0 \end{bmatrix}. \tag{2.1}$$

Calibration of the bottom camera

$$\mathbf{K}_{bottom} = \begin{bmatrix} 806.8948 & 0 & 282.0628 \\ 0 & 807.4827 & 223.9356 \\ 0 & 0 & 1.0 \end{bmatrix} \tag{2.2}$$

is not equal to the $\mathbf{K}_{top}$, thus there is some difference in the cameras' geometries. Acquired calibration matrices are in the following form

$$\mathbf{K} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix},$$

where α_x and α_y represent the focal length of the camera in terms of pixel dimensions in the x and y direction respectively and where $\mathbf{x}_0 = (x_0, y_0)$ is the principal point in pixel dimensions [10]. From small differences between α_x and α_y in both cameras it can be assumed that the cameras have square pixels.

Radial distortion

is a deviation from rectilinear projection, such that straight lines in the scene are not projected as straight lines in the image. In order to build accurate 3D representation from images, these images need to be undistorted, i.e. transformed, so that the lines are again straight.

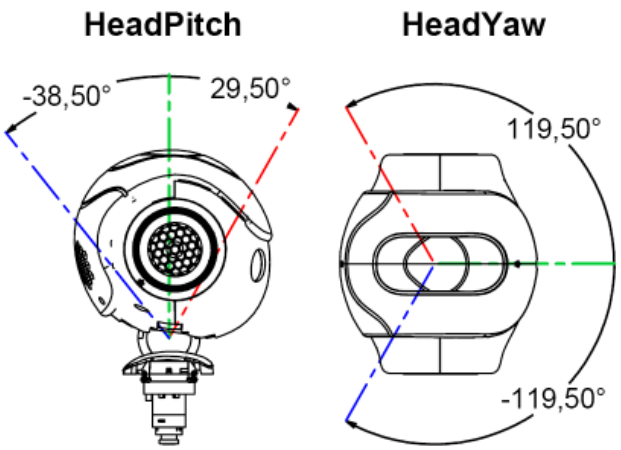


Figure 2.3: Head pitch and yaw limits, courtesy of Aldebaran Robotics [1].

The distortion of images captured by Nao's cameras is caused by small lenses, which are hard to model. But again using the calibration system [17], the two parameters of radial distortion according to second degree polynomial model were calculated.

2.1.2 Motion

In order to acquire images of the robot's environment which could be processed into 3D model, the robot must be able to change its position to obtain images from different viewpoints. The application interface (API) shipped with the robot provides many functions to control the movement of individual joints, i.e. low level control, and also functions to control the whole–body movement, e.g. walk.

Head movement

Movement of head is accomplished by two joints in the robot's *neck*. Limits of yaw and pitch are given in Figure 2.3. By merging the information from camera poses (in Figure 2.1) and pitch limits, it is obvious, that it is not possible to set the bottom camera to point horizontally. Due to the shape of the head, full yaw is possible only with limited pitch. This information is not available in the documentation [2] and had to be empirically measured.

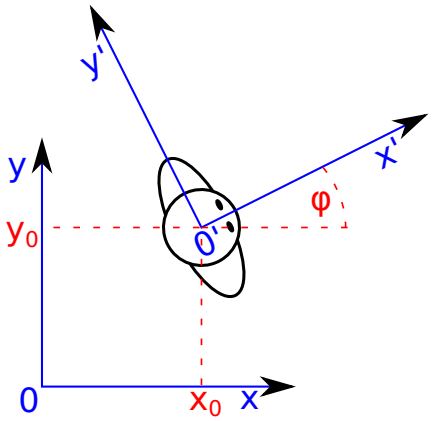


Figure 2.4: World coordinate and robot coordinate systems.

Walk

Robot's API contains several functions that control the robot's pace. It is possible to set a target position to go to, relatively to current robot position, set speed of walking, make individual steps or control arms movement when walking.

The API on one hand greatly simplifies the usage of the robot, but on the other hand, the implemented gait is simple and almost without any feedback concerning floor (un)evenness or dynamical changes[1]. The robot heavily depends on the floor surface structure and adhesion and can be easily affected by small obstacles, e.g. wires.

Moreover the motors that are placed in robot's legs seem to be underestimated, since they tend to overheat when the robot is walking. Robot developers introduced a simple mechanism to prevent joints overheating by reducing current to affected joints. This on one hand protects the effortful joints, but on the other hand can cause damage to whole robot if such current reduction occurs while the robot is walking or standing, as the limited joints will not be able to support the body mass sufficiently. To prevent damaging of the robot, a simple monitoring program was developed that informs the operator about all joints' temperatures, see Section 3.2.2. The operator can then act accordingly, e.g. reduce stiffness in the joints to let them cool down.

[1]There is only one feedback mechanism that stops the walking process if the robot looses contact with floor, i.e. has fallen or was lifted by someone, as it is not needed to walk anymore in such cases.

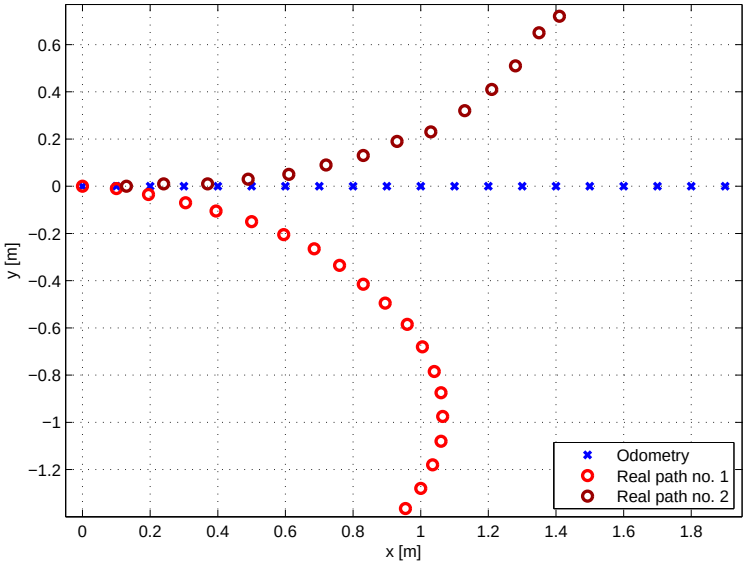


Figure 2.5: Difference of robot odometry and true motion. The real paths were measured by marking robots positions and measuring them with meter afterwards.

Odometry

The robot keeps track of its odometry based on the number of steps it makes and on their azimuth and length. This pose estimation is taken with respect to a world coordinate system, that has its origin in the place, where the robot was turned on (see Figure 2.4). Since there is no function in the API to reset or set the robot's odometry, the odometry needs to be transformed so that the robot's starting position is in the home position as will be explained later in this thesis. During experiments with the robot, it was noted, that the deviation of odometry from the true motion is affected by leg joints temperatures and that it can not be relied on the precision of walking. This forces the usage of other means of localization that would correct or measure this error and allow for safe and precise navigation. Example of robot's estimated and true path is given in Figure 2.5. It is known, that odometry estimated from mobile robot motion is not precise and cumulates error as the robot moves, thus the Nao's odometry was not expected to be correct, but from the experiments can be seen that it cannot be used even on small distances and another means of pose estimation need to be employed.

2.2 Naoqi

Naoqi is a framework running on the robot that is responsible for its control. It manages several proxies that mediate the interaction with robot's modules for specific peripheries, e.g. memory, motion, or camera modules. The modular architecture of this framework allows for interprocess communication, handles parallelism, synchronization, and manages events. It allows homogeneous communication between different modules and simple to use memory sharing with subscription mechanism. By connecting to a broker, which is an executable running on the robot and which is listening to commands from network, it is possible to control the robot from a remote computer.

2.2.1 Robot programming

Naoqi exposes functionality to modules that can control the robot. There are two types of modules, remote and local. Remote modules are running on a separate computer and communicate with Naoqi on the robot via network, i.e. Ethernet or Wi-Fi. Local modules, on the other hand, are cross compiled for the robot's architecture and are running directly on the robot. This allows faster feedback and removes the need for another computer. Moreover, some API functions are available only for remote or only for local modules.

All functionality is wrapped behind so called *proxies*. There are specialized proxies for every implemented functionality, e.g. motion, memory access, cameras, etc. Modules need to utilise these proxies in order to control the robot or benefit from its functionality. Proxies can be obtained after connecting to a broker.

The Naoqi framework supports programs written in C++, Python, and Urbi. It is possible to call the embedded Python interpreter from C++ code using *Python bridge*. Modules can also expose their public functions that can be subsequently called by other modules, independently on the language in which they are written.

A simple example of a remote module using memory proxy is given in Algorithm 1. This short program firstly connects to Naoqi running on robot on port 9559 and IP address specified in *robot-IP*. Then it inserts a value 3.14 associated with name "myValueName" into the shared memory using the memory proxy.

The version of Naoqi used for the programming is 1.8.16, but all developed software should run on previous versions, too.

Algorithm 1: Remote python module using memory proxy.

```
from naoqi import ALProxy
memProxy = ALProxy('ALMemory',robot-IP,9559)
memProxy.insertData('myValueName', 3.14)
```

2.3 Image processing pipeline

This section presents the reader with image processing pipeline SfMSeqv2, that is used in this thesis for 3D scene reconstruction from a sequence of images captured by the robot. The described pipeline uses Matlab® with several C written functions that are connected to Matlab® via MEX. All computation is therefore performed on an external computer instead of directly on Nao. The robot is thus used only for image acquisition.

2.3.1 Structure from Motion

Structure from Motion (SfM) is a well known task in the area of geometric computer vision [10] that still poses a challenging problem for researchers. It is a task of computing a 3D model of a scene or an object from a series of 2D images. The images either capture a rigid movement of objects in the scene or are taken by a moving camera in a static scene. Without the knowledge of camera movement or distances between the objects, the computed 3D model can be determined up to a scale factor that must be obtained by other means, if needed. Some of the tools available for the SfM computation are the Bundler [23] that accepts an unordered image collection, e.g. images from Internet, and SfMSeqv2 pipeline [24], which processes sequences of images to create the 3D model.

A typical approach to SfM is to firstly detect visual features in the input images. Secondly tentative matches between the images are found. Here the assumption of SfMSeqv2 pipeline, that the images are captured in a sequence, reduces the number of images that need to be tested, since the matches are searched for only between the successive images. Other steps are implementation dependent, but usually consist of applying further restrictions, calculating epipolar geometries, reconstructing the 3D model, and optimizing the camera poses and 3D points position.

2.3.2 SfMSeqv2

This pipeline was developed at CMP [5] for camera pose and trajectory estimation and for 3D reconstruction. It is based on the sequential SfM pipeline

with loop closing [24] and can be used to obtain camera poses and to create 3D models from sequential image sets e.g. video sequences. It works both with perspective and omnidirectional images as long as the internal camera calibrations are known. The order of the images in the sequence is determined by the lexicographical ordering of their names.

The pipeline runs as follows. Firstly, MSER+LAF [18], SIFT [16] and SURF [3] features are detected and described in each of the input images.

Secondly, tentative matches are created from pairs of consecutive images from the sequence. Matches are constructed from best matching descriptors using FLANN [19]. The matching is performed for each type of feature independently. Tentative matches are verified by calculating pairwise epipolar geometries using voting scheme based on RANSAC [9].

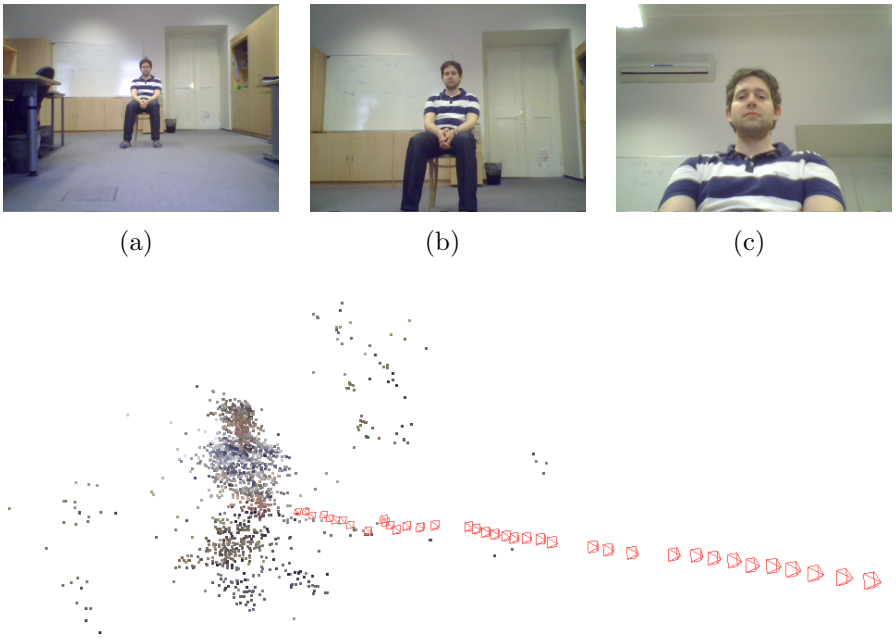
Thirdly, pairwise matches are concatenated into tracks. Tracks, that are shorter than three images are filtered out, because they might be created from incorrect matches.

Finally, camera poses with respect to the first camera are recovered by chaining epipolar geometries of successive images. Then a sparse bundle adjustment routine SBA [15] is employed to improve both camera poses and the positions of the reconstructed 3D points.

The pipeline generally accepts any sequence of images, that are taken by a moving camera. The movement (translation and rotation) between two successive images must not be too rapid, as that would present the pipeline with images that do not have any matching feature points. This can occur on sharp turns, when a large part of the scene moves away from the camera field of view. On the other hand, with too slow movement the resulting 3D reconstruction plausibility is weak due to small baseline and thus overly similar images are dropped from the sequence, resulting in a better reconstruction.

Using several parameters, the process of the pipeline can be adjusted to fit particular needs. Among the many parameters belong selection of feature detectors to use, minimal motion (in radians) not to discard image as explained above, minimal track length etc.

Example of a scene reconstructed from a sequence of images using the SfM-Seqv2 pipeline is given in Figure 2.6. The sequence is 39 images long and was taken by robot's top camera. From the reconstructed model it can be seen that most of the features are on the person itself and only few features were detected elsewhere. This is caused by rather plain and homogeneous background and could be troubling, if the person was not still. More details about the features detection is given in Section 3.1.3.



(a) (b) (c)

(d)

Figure 2.6: Scene reconstruction from images captured by the robot. (a)-(c) Images from the sequence, (d) reconstructed scene (dots) and camera trajectory (red pyramids), courtesy of [11].

2.3.3 Odometry and visual models merging

The result, produced by SfMSeqv2 is composed of several parts. Firstly, there is a VRML model, that contains 3D representations of feature points from captured images. Secondly, there is also a list of cameras' positions, that correspond to individual images, used for the reconstruction. These positions can be used for robot path reconstruction. Camera position are given in the form of camera matrices $\mathtt{P}_i = \mathtt{K}_i \begin{bmatrix} \mathtt{R}_i & \mathbf{t}_i \end{bmatrix}$, where $\mathtt{K}_i$ is a camera calibration matrix, as described in Section 2.1.1, $\mathtt{R}_i$ is a 3-by-3 rotation matrix representing the orientation of the camera coordinate frame and $\mathbf{t}_i$ is a translation vector, such that $\mathbf{c}_i$

$$\mathbf{c}_i = -\mathtt{R}_i^T \mathbf{t}_i \tag{2.3}$$

is the vector denoting the camera position in a world coordinate frame.

The model created from images is defined up to a scale factor, which needs to be determined by other means. Here the knowledge of robot movement obtained from odometry is utilized. Since odometry is not accurate, a median value d_v is taken from the distances of individual camera's centers, that is

divided by average robot movement per step d_o obtained from odometry (about 0.1 m), thus the scale factor $s = d_v/d_o$. Furthermore, the cameras need to be rotated and translated so that the first camera is in the *home position*, which is further denoted as the world coordinate system origin. Therefore, the i^{th} camera position in the world coordinate frame is expressed as

$$\mathbf{p}_i = (\mathbf{t}_0 + \mathtt{R}_0 \cdot \mathbf{c}_i)\,/s \tag{2.4}$$

$$= (\mathbf{t}_0 - \mathtt{R}_0 \cdot \mathtt{R}_i^T \cdot \mathbf{t}_i)/s, \tag{2.5}$$

where $\begin{bmatrix}\mathtt{R}_0 & \mathbf{t}_0\end{bmatrix}$ is the first camera matrix and $\begin{bmatrix}\mathtt{R}_i & \mathbf{t}_i\end{bmatrix}$ is the i^{th} camera matrix (without the calibration matrices) and $\mathbf{c}_i$ is the i^{th} camera position vector as described in equation (2.3). By transforming all the obtained cameras into the world coordinate frame, the path calculated from odometry is matched to the path from visual tracking and it is possible to command the robot to go to the desired pose, since the transformation from visual map to real world is known.

Chapter 3

Solution

This chapter describes the experiments that were conducted in the scope of particular scenarios. Hardware specific solutions, i.e. workarounds, are presented and discussed in the first section. The following sections give a brief overview of the applications developed for image capturing, robot monitoring, and scripts for processing the captured images. Finally, the experiments are described and their results discussed.

3.1 Workarounds

Following sections describe solutions to problems, brought about by hardware limitations that were enumerated in the previous chapter. Both problems are related to robot's cameras, which quality is not very high. This section tries to solve the main issues, so that the following image processing tools do not have to be concerned with it and can be designed as non hardware specific and thus commonly applicable.

3.1.1 Stop & go

Since the robot's cameras are CMOS with no shutter, images of the robot's surroundings are taken only when the robot is not moving. This results in a *stop and go* fashion of motion and significantly increases the time needed for capturing images. The waste of time is minimized by utilising these pauses for transferring images from robot to computer for further processing. This enables to process the images online and use the results to control the robot, thus closing the feedback loop.

3.1.2 Panoramas

Due to the narrow field of view of the cameras, it was at first considered a must to broaden the field of view by connecting more images taken from one place. The Capture module (described in Section 3.2.1) was originally developed exactly for this purpose.

The images for panoramas are taken by turning Nao's head 33.5° left and right, which creates a 10° overlap of the horizontally adjacent images. The movement of the cameras is approximated as a pure rotation, thus the images are related via a homography. For distant objects (farther than circa 2 m), this approximation is valid. The homography is computed from matches between visual features detected on the overlapping area using RANSAC algorithm.

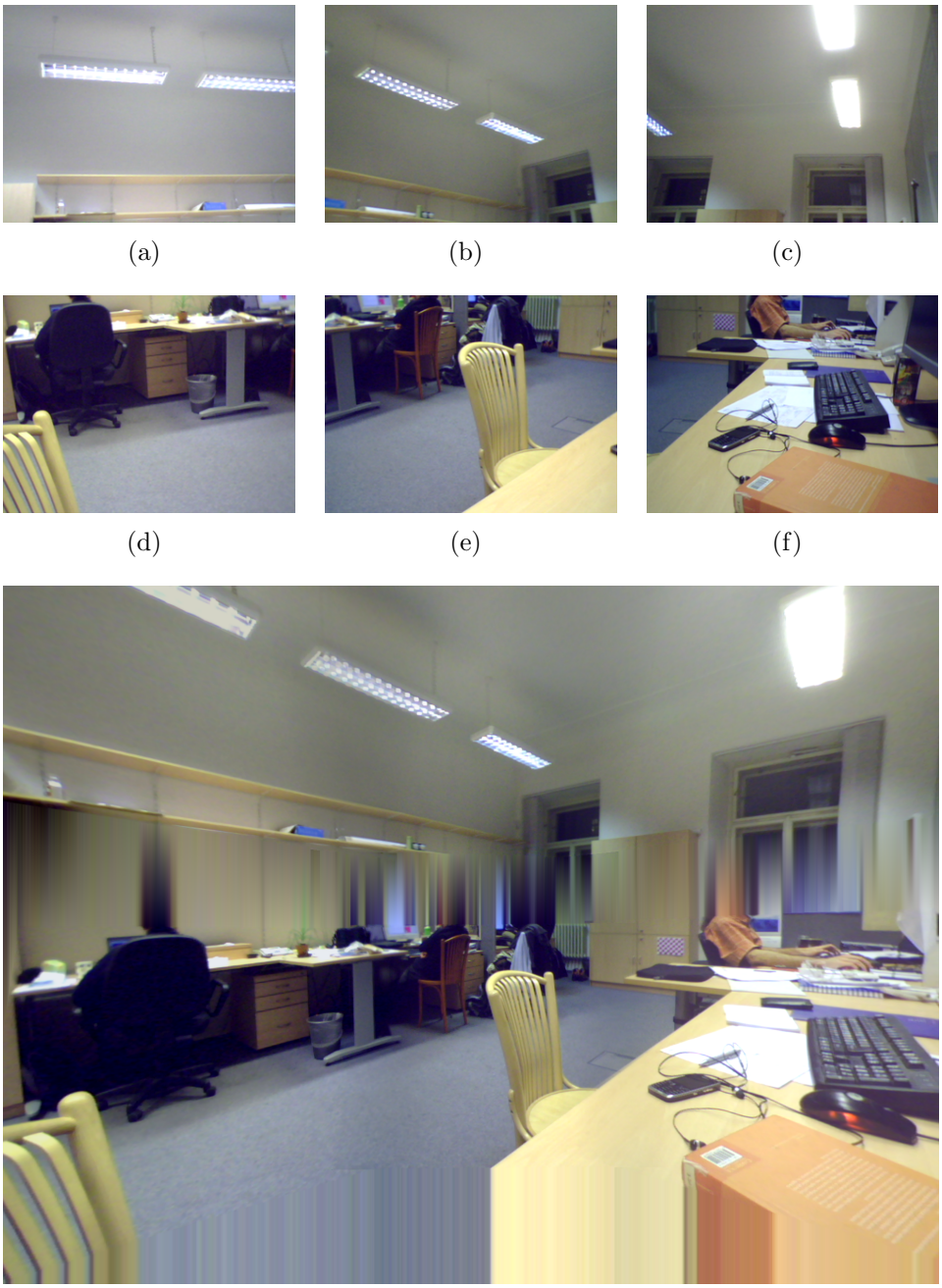
Unfortunately, the head movement is not precise and repeated empirical measurements of the pitch and yaw show that the average absolute error in yaw is $0.41° \pm 0.45°$ and in pitch $0.71° \pm 0.78°$. This means, that the homography would need to be computed separately for every image set. This on one hand increases the time consumption, but also requires enough correspondences between the images, which cannot be guaranteed for all images. This lead to the decision not to use this approach in this thesis. Furthermore, the overhead of creating panoramas may be solved in the future by equipping the robot with better cameras, that would have wider field of view.

Example of a panorama is given in Figure 3.1. The areas that are not contained in any of the images are impainted [11].

3.1.3 Lighting conditions and scene enhancement

Considering the robot's height and camera field of view, there are not many traceable features in the images of the office–like environment at CMP. Typical view is given in Figure 3.2(a). The lack of detected features is caused by insufficient lighting conditions, given the cameras low sensitivity, and by rather homogeneous areas, where there are not many features at the given resolution of the cameras.

Several steps are taken to avoid the noise in the captured images: (i) the auto–gain feature is turned off, which on one hand reduces the amount of noise in the images, but when the scene is dark, the captured images are again unusable for further processing, (ii) an artificial lighting is used to increase the brightness of the scene. There are available two different approaches or their combination. A strong static artificial lighting can be employed to illuminate the scene for better results, or a small source of light (headlight) can be attached to robot's head to point in the direction of the cameras, that would illuminate only the small portion of the scene that is currently in the robot's



(a) (b) (c)

(d) (e) (f)

(g)

Figure 3.1: Source images (a)-(f) captured by the robot and (g) the created panorama with impainted regions.

scene type	MSER+ inten. + LAF	MSER- inten. + LAF	SIFT	SURF	$\sum$
Plain, dark	181	12	15	32	**240**
Plain, bright	192	0	26	84	**302**
Enhanced, dark	754	682	186	308	**1930**
Enhanced, bright	1152	720	232	467	**2571**

Table 3.1: Number of detected keypoints in each scene separately for each detection method.

field of view. Alternatively, both the given approaches can be used together, to obtain the best luminary conditions for image acquisition.

For development and testing of the images processing tools this presents valid options, but in *real–life* situations, it is often not possible to introduce such luminary conditions.

To increase the heterogeneity of captured scenes, the environment is enhanced with artificial features, by adding magazines, books, and boxes. This can simulate a group of people, scattered around the scene. It is important to place the objects uniformly for the created 3D representation to be correct.

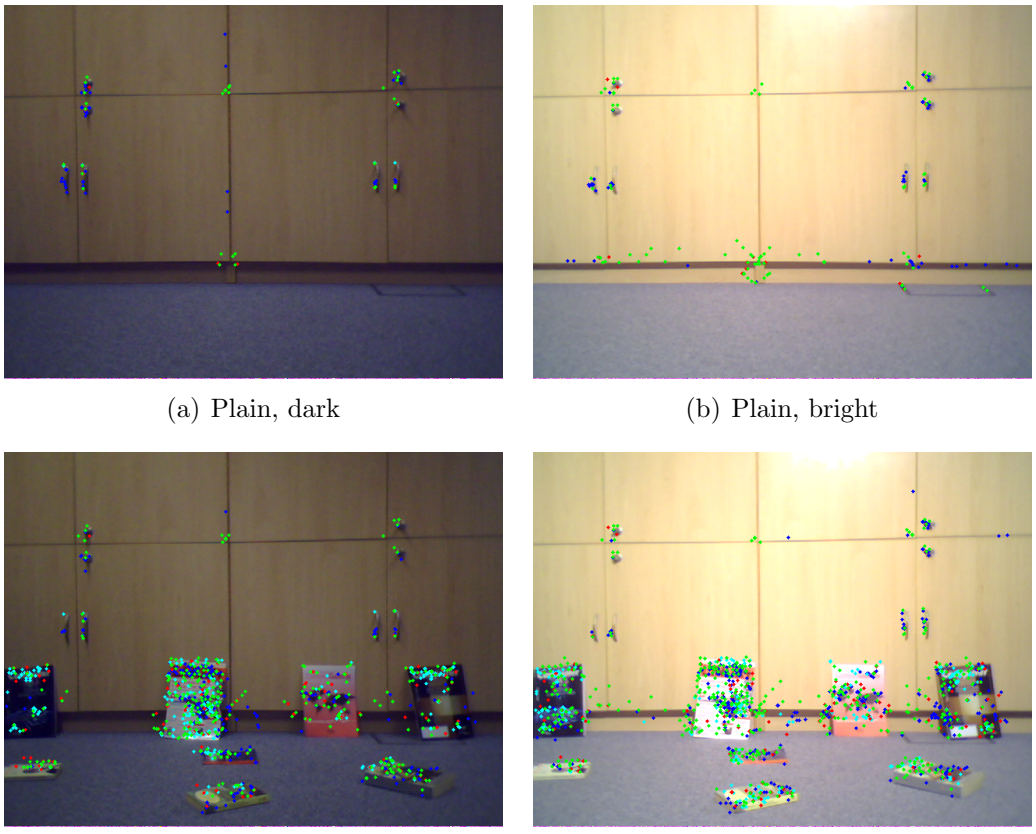
Quantitative comparison

Figure 3.2 shows detected keypoints in the scene for different conditions. Each feature is drawn in different colour, for better distinction.

In the plain scene, there are features detected only in corners or around edges of the lockers' doors. Only few features are found on the floor, where the hatch is placed. Due to small camera resolution, no keypoints are detected on the surfaces, e.g. on the floor or doors. The scene enhanced with artificial objects provides much more traceable features, that are mostly detected on the added objects. Quantitative comparison of the individual scenes is given in Table 3.1. From the images and table, it can be concluded, that the scene enhancement with artificial features has bigger influence on the amount of detected features than lighting condition.

Conclusion

During the experiments, the lighting conditions were achieved by using two stationary halogen lamps, which were placed so that they illuminated the scene uniformly. With such illumination, it was not necessary to attach another light source on Nao's head. Although the illuminated images are better in quality, it is sufficient to use ordinary ceiling illuminants. On the other hand, the scene



(a) Plain, dark

(b) Plain, bright

(c) Enhanced, dark

(d) Enhanced, bright

Figure 3.2: Office–like environment from Nao's point of view. The coloured dots denote detected keypoints. MSER+ inten. + LAF are blue, MSER- inten. + LAF are cyan, SIFTs are red and SURFs are blue.

enhancement with artificial objects is of utmost importance, as can be seen from quantitative comparison given in Table 3.1. The number of traceable features increased approximately 8 times and 8.5 times for dark and bright scene respectively.

3.2 Support applications

This section briefly introduces several applications, developed within the individual project, that preceded this thesis. They were further improved during the work on this thesis to provide more functionality.

3.2.1 Capture module

The Capture module was developed to ease the capturing of images taken by Nao's cameras and for quick camera parameters setting. It is designed as a local module i.e. cross–compiled and running independently on Nao. It can be used directly both from Python a C++ applications.

Using this module, it is possible to select, which camera to use for capturing. The module is also capable of turning off or on the auto–gain, setting the gain manually, setting white–balance and enabling or disabling automatic exposition.

Instead of sending the captured images directly to PC, the module saves the images on Nao, where a FTP server is running, for later download. This enables parallel control of the robot and image processing, as the images can be transferred from the FTP server independently, by using any file transferring applications, e.g. `wget`.

Detailed description and the application interface to this module as well as an example of usage is given in Section 4.1.

3.2.2 Monitor

This console application's main purpose is to monitor the temperatures of robot's individual joints. It connects to Nao's memory proxy and periodically reads the values, stored there. Its usefulness was proven especially during the hot summer days, when the robot's overheating was a big issue. Even when the ambient temperature is moderate, some joints tend to overheat, especially those in the robot's legs, while the robot is walking or crouching, since they are supporting the whole body mass. Thus it is wise to monitor the joint status and interrupt experimenting with the robot, as soon as they become too hot.

```
   LAnklePitch = 39.00°C
   RAnklePitch = 30.00°C +
    LAnkleRoll = 48.00°C
    RAnkleRoll = 37.00°C
    LKneePitch = 36.00°C +
    RKneePitch = 55.00°C ++
     LHipPitch = 37.00°C +
     RHipPitch = 36.00°C +
      LHipRoll = 35.00°C +
                .
                .
                .
         RHand = 38.00°C -
Battery charge = 92.00%

Refresh period 10 s, threshold1 = 55 °C, threshold2 = 65 °C
```

Figure 3.3: Sample output of the Monitor application (shortened).

As the Naoqi evolved, the application needed to be adjusted, since some monitored values were no longer stored in the Nao's memory, e.g. temperature of the processor or motherboard. It was further improved to show also the battery status. A sample output is given in Figure 3.3, the `+` and `-` signs show the difference in temperature from the last update.

3.2.3 Robot control and image processing scripts

Several scripts were developed for controlling the robot and for processing the images captured by the robot. In every experiment, there are two applications running on the PC, that are communicating with each other and with the robot. A simple diagram of dataflow and the communication is shown in Figure 3.4. It shows two separate units, robot Nao and the control computer. On robot Nao there is running the *Capture* module, described in Section 3.2.1 and the *Motion* module, which is part of Naoqi framework. Both modules are used by Python script on PC that communicates with them using a public API. The Python script controls the robot, i.e. its motion and cameras, transfers images from robot to the Matlab® script, and reacts on localization results produced by the script. All the scripts are described in later sections of this thesis, together with the particular scenarios.

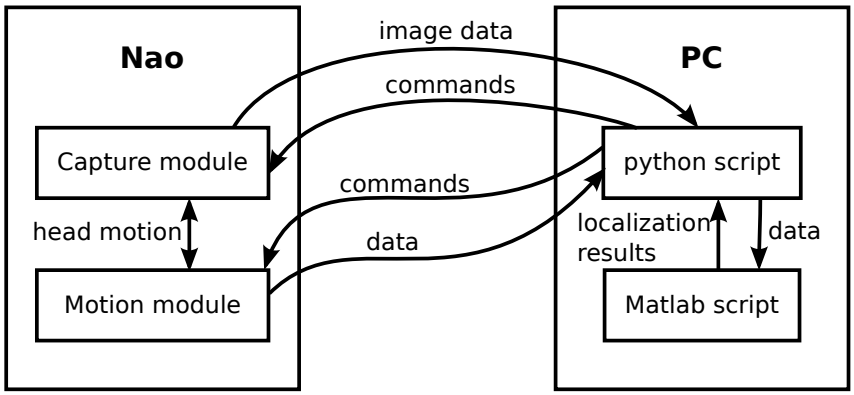


Figure 3.4: Overall scheme of interprocess communication.

3.3 Scanning

The scanning process is a building block for all other scenarios, as it produces a 3D map of the robot's surroundings. It employs the SfMSeqv2 pipeline, which is described in Section 2.3.2. The process of acquiring images is described in this section.

3.3.1 Setup

Scanning process was tested in an enhanced office–like environment at CMP. Since images from the bottom camera on our robot are better than from the top camera, it was decided to use images mainly from the bottom one. To acquire images other than of floor, the robot head pitch was set to its limit at $-38.5°$ so that it aims almost horizontally (see Figure 2.1 and 2.3 for details). This approach leaves the top camera free for e.g. people detection, as it aims upwards. Similar approach of *stop and go* movement and visual navigation was also employed by Oßvald et al. [22], but they were using the bottom camera to detect features on the floor, which means, that they would not be able to detect human faces or any higher placed objects.

The environment in all scenarios is mostly static, except for people that are occasionally walking in the background and do not occlude the scene perceived by the robot. Thus the created 3D model of the environment consists only of fixed 3D points and the algorithms do not have to account for any changes.

3.3.2 Batch vs. iterative processing

There are two ways of reconstructing the scene from acquired images. Firstly, the robot can be ordered to walk in the *stop and go* fashion and capture the

images, which are processed *after* the walk is finished, i.e. offline. Secondly, the 3D representation can be created online as the robot walks and thus the robot's path can be adjusted according to the built representation.

The first approach is a little faster, as the reconstruction is done only once, but heavily relies on the robot's walk, which is not perfect. The second approach decreases the dependency on precise walking, but requires more computational time, as the representation is built in each step and also a localization process takes place to correct robot's direction of movement. Moreover the original SfMSeqv2 pipeline is not designed for iterative image processing, but rather to process the whole set of images at once. The modified SfMSeqv2 pipeline is presented in detail in Section 4.2.

Merging of the localization and representation building into one process results in a Simultaneous Localization And Mapping (SLAM) technique [4]. In this case it is a variant known as Visual SLAM, since a visual system is employed for localization and representation building.

3.3.3 Experiment

The scanning experiment runs as follows. Firstly, the scene is created according to the setup described above. Then the robot is placed on its starting position. This position is in future text referred to as to the *home position* and in the created world model it is always positioned on coordinates $[0, 0, 0]^T$, where the first two coordinates denote 2D position in the world and the third one is azimuth. Thus the robot is in its starting position standing in the origin and looking in the direction of the x axis. Figure 2.4 shows the world coordinate system $(O, [x, y])$ and coordinate system, connected with the robot, in which the robot is commanded. Although the SfMSeqv2 pipeline produces camera positions in 3D (6 degrees of freedom), the elevation is ignored, since the floor is assumed to be even, thus only 2D position of the robot is used, i.e. robot's pose is determined by vector

$$\mathbf{r} = \begin{bmatrix} x \\ y \\ \phi \end{bmatrix}, \tag{3.1}$$

where x and y denote robot's position in the direction of x and y axis respectively and ϕ is robot's yaw with respect to the world coordinate system.

Further steps depend on the method of building the reconstruction and are described separately.

Batch The robot starts capturing images and walking forward, in a *stop and go* fashion as described in Section 1.2. In each step the robot takes one image

by the bottom camera and is ordered to walk 0.1 m forward. When the walk is finished and all images captured, the building process of the 3D representation is started.

Iterative In the iterative approach, the robot is walking in a *stop and go* fashion again and in each steps captures one image, but its walk is not fixed as in previous approach, but depends on its actual pose with respect to the built model. Thus the robot walks in fixed steps until a 3D model is created, then the walk is adjusted according to the result from localization, which corrects the odometry error and allows the robot to follow the desired trajectory.

The computation of the 3D model is carried out on an external computer, as the algorithms heavily depend on Matlab® and could not be cross–compiled to run on the robot. Moreover, the cost of transferring images from robot is outweigh by the speed of programs running on PC. The transfer from the robot to the controlling computer is happening in real–time, while the robot is walking. The application, that drives the robot (controls its movement and handles images) then hands the images to a Matlab® script, that creates the 3D model of the perceived scene and calculates poses of individual cameras, where the images were taken. These poses are transfered back to the control script.

3.3.4 Results

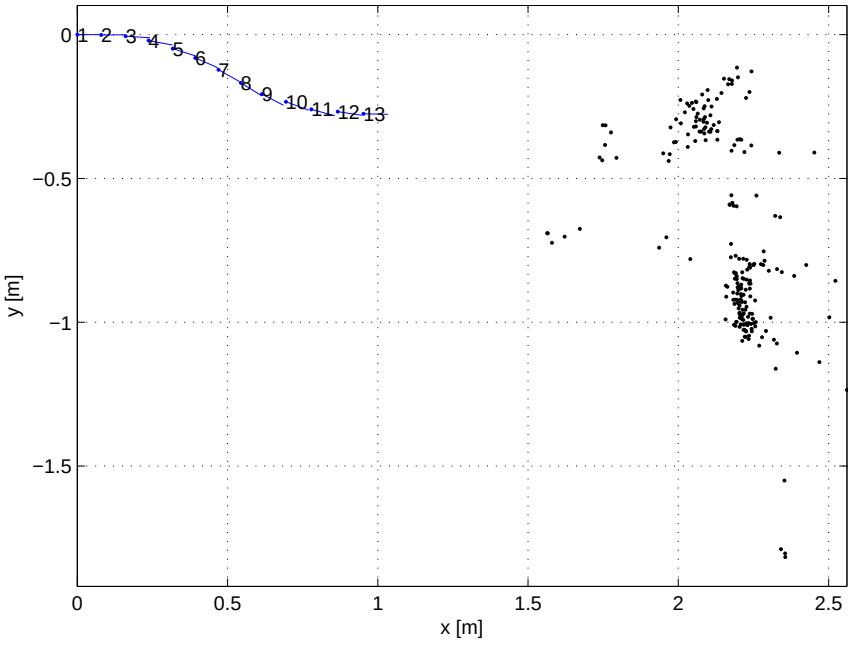
Experiment conducted in scope of this scenario verified, that the proposed approach of image acquisition is working and that the SfMSeqv2 pipeline is usable for building the 3D representation and with some adjusting even in the iterative mode.

Example of a robot walk and reconstructed 3D points is given in Figure 3.5. The model was created using the batch approach, thus the path, which should be straight is more like a double bent curve, which is caused by error in odometry estimation.

(a) no. 1

(b) no. 4

(c) no. 12



(d)

Figure 3.5: Reconstructed points (coloured dots) and camera poses (blue dots with lines denoting direction) from a sequence of images of an enhanced scene captured by the robot using the batch approach. (a)-(c) Sample images from the 13 image long sequence, (d) reconstructed 3D points and camera poses.

3.4 Localization

Localization is the ability of a mobile robot or any mobile device in general to determine its position with respect to some coordinate frame or internal representation. Localization can be global, i.e. using Global Positioning System (GPS), or local that is restricted to some area, without the knowledge of position of this specific area with respect to the ambient world.

In this thesis the latter variant is implemented, since the robot is not interested in where it is from the global point of view, but rather needs to determine its position with respect to surrounding objects, e.g. people, obstacles, etc.

3.4.1 Kidnapped robot problem

Kidnapped robot problem [7, 20] is often used in mobile robotics as a test for robot localization algorithms, to recover either from catastrophic odometry errors or from exterior actions affecting the robot. There are at least two main ways to look at this problem. Firstly, it can be brought about by an odometry error, caused by faulty sensor, actuator, or their uncertainties and secondly the robot is actually moved away from its original position by an external actuation, e.g. by humans, robots, or natural forces. Similar to kidnapped robot is a wake-up robot problem [20] that focuses on localizing a mobile robot after its power-on.

The main goal of the aforementioned problem solvers is to localize the robot with regard to previously built world representation and world coordinate frame. This assumes that the robot is able to find out that it has been kidnapped and it may enforce new mapping and exploration process that creates new world representation, which is then merged with the original representation, given there is a valid match between the two.

The 3D world representation, which is needed for successful localization is produced by a scanning process, described in Section 3.3.

3.4.2 Experiment

The robot is placed arbitrarily in the environment. This can be perceived as the robot being kidnapped, as there must be a previously built world representation. Moreover, since the robot odometry precision is low and each iteration in the movement brings great amount of uncertainty into robot's pose, even a simple walk may be understood as a continuous kidnapping.

The robot sequentially captures images that are transfered to a PC for further processing. Features are detected and described in the images and are matched with the 3D representation. If no match is found, the robot turns 12°

to the left and acquires another image. This is repeated until a valid match is found or until the robot performs a full spin, which means, that it is not able to localize itself. Successful localization yields robot position and orientation in a form of a camera matrix $\mathtt{P} = \begin{bmatrix}\mathtt{R} & \mathbf{t}\end{bmatrix}$, with the same meaning as described in Section 3.3). This process of kidnapping can be repeated to test the localization algorithms or used during navigation to correct the odometry error.

3.4.3 Implementation

This section concerns the software implementation of a control program (a script written in Python), a Matlab® script and their interaction and synchronization. Pseudocode of the control script is given in Algorithm 2 and the Matlab® part of the process is shown in Algorithm 3. Python was employed as it is supported by Naoqi framework and the API is well documented. Moreover the development process is faster than that of C code and control program speed is not an issue in this case.

Communication

Synchronization of the two processes is accomplished by simple *lock* file, that is created by the control script, just before the copying of new images starts and is removed by the same process, as soon as the file transfer is complete. This prevents Matlab® from using incomplete image files for processing.

For the communication in the opposite way a Python module called Pickle is employed. It is a module capable of serializing and de–serializing Python object structure. It converts object into a byte stream, that can be saved in a regular file. Once the structure of the file was experimentally determined, it was relatively easy to implement a code in Matlab® to output results to a file understandable by Python. The output from Matlab® consists of an array composed of tuples holding image ID (timestamp) and the pose of camera, given as vector $\begin{bmatrix}x, y, \phi\end{bmatrix}^T$.

3.4.4 Results

The presented algorithms are able to localize the robot with respect to a built representation, if at least a part of the reconstructed scene is visible from the robot's point of view. An example of a successful localization is given in Figure 3.6, where it is used several times during a homing navigation.

Algorithm 2: Localization - control script

```
initialize robot;
localized := false;
while not localized do
    capture image;
    transfer image from robot to Matlab® new folder;
    wait for response;
    switch response do
        case LOCALIZED
            localized := true;
        end
        case NOT_LOCALIZED
            if not whole turn then turn right;
            else exit;
        end
    end
end
finalize robot;
```

Algorithm 3: Localization - Matlab® script

```
initialize SfMSeqv2;
foreach file in img folder do
    undistort(file);
    detectFeatures(file);
end
computeTentativeMatches();
calculate3DModel();
while true do
    wait for new file in new folder;
    undistort(file);
    detectFeatures(file);
    matched := matchFeaturesTo3DModel();
    if matched = true then send LOCALIZED to control script;
    else send NOT_LOCALIZED to control script;
end
```

3.5 Navigation

Navigation is the ability of a mobile robot to change its configuration to a desired one. From the point of view of planning theory, it is a process of finding a sequence of actions that—when successfully realized—would result in the desired position.

In order to perform this task, the robot needs to know the transition between the current and target configurations and be able to plan a sequence of actions, that would result in the desired configuration.

3.5.1 Homing navigation

Homing navigation is a special case of general navigation, which always has the same goal position. It is the ability of a robot to return to its home configuration or home position, considering a mobile robot.

Homing can be useful especially for mobile robots that are powered by batteries and for which the home position is a recharging station. By using homing navigation, the robot can return to the home position, when the battery status is low and resume operation once its energy is replenished. When automatized, this creates an interesting self-preservation behaviour, that brings machines closer to living things.

For a mobile robot, homing is based on a previously built map of the environment and on localization, that produces a relative position of current position with respect to the home position.

Let $\begin{bmatrix}\mathtt{R}_i & \mathbf{t}_i\end{bmatrix}$ be matrix of the i^{th} camera and $\begin{bmatrix}\mathtt{R}_0 & \mathbf{t}_0\end{bmatrix}$ matrix of the first camera (placed in the home position). Now since the robot is commanded relatively to its position, we do not need the robot's position relative to world coordinate frame, as expressed in equation (2.5), but rather the home position relative to current robot position. This is computed similarly as follows

$$\mathbf{p}_{home} = (\mathbf{t}_i - \mathtt{R}_i \cdot \mathtt{R}_0^T \cdot \mathbf{t}_0)/s. \tag{3.2}$$

This produces the desired relative position of home $\mathbf{p}_{home}$. Again, the scale factor and camera matrices are needed for the calculation.

Experiment

Prerequisite of this experiment is a 3D representation of the robot's surroundings, which is created by a scanning process, and robot position, different from the *home* position. To be able to calculate the path, the robot needs to successfully localize itself with respect to the 3D representation.

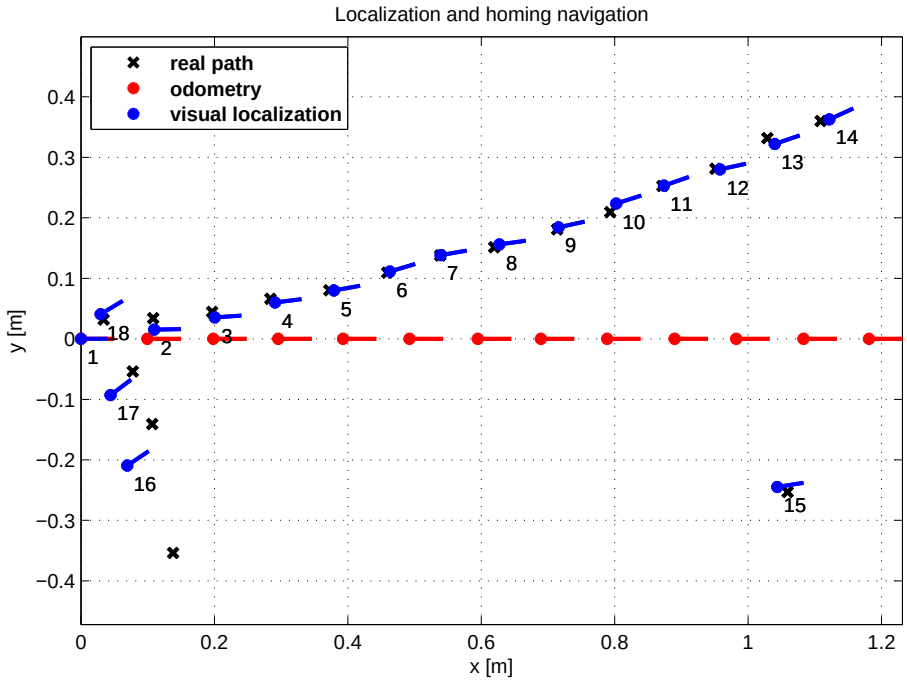


Figure 3.6: Robot localization and homing. Points 1–14 are poses of the robot while scanning the scene. Point 15 denotes the pose, where the robot localized itself after being kidnapped and points 16–18 are poses on the way to home position. Real path was measured by marking robot's positions and measuring them using a meter afterwards.

The robot could be then directly ordered to go to position $\mathbf{p}_{home}$, which in perfect world would lead to robot successfully reaching its home position, given there would be no obstacles on the way. However, one more workaround is needed, because of the unreliable walk. The way home is iterative, which means, that the robot traverses only a part of the whole path (maximally 0.5 m) and launches the localization process to correct for odometry error. This iteration stops, when the euclidean distance of the robot to the home position is less than 0.1 m. This approach on one hand again increases the overall time, but on the other greatly improves the robot navigation, especially when the home position is further away.

Results

Results from one experiment are presented in Figure 3.6. The dots represent the positions of cameras (or robot in the case of odometry) and the lines denote directions, where the robot or cameras are headed. Red data are obtained from

odometry, which is also transformed, so that the first pose is in home position, and blue are camera poses as calculated using SfMSeqv2 and localization. The robot's trajectory starts at $\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$ and continues in the direction of the x axis. Black crosses are positions of the robot, measured using a meter. The unnumbered blue points are camera poses during the scanning process. In this case, the scanning was performed using the *batch* variant. Blue point number 1 denotes the position of robot localized after it was kidnapped, points 2 and 3 are robot's localized positions on its way home. From this image it can be seen, that it is necessary to perform the (homing) navigation iteratively, as the position number two is about 0.5 m away from the desired home position. It can be seen that the error of localization is greater, as the robot is further away from the reconstructed scene. This is caused by small baseline and mostly frontal movement. Better cameras would improve the localization quality.

3.5.2 Iterative navigation

The iterative navigation employs the results obtained from modified SfMSeqv2 pipeline to correct robot's pose and trajectory in order to follow the desired path precisely. The path is given as a sequence of poses $\begin{bmatrix}x & y & \phi\end{bmatrix}^T$. The robot should navigate along these way–points and take picture in each of them. This is possible thanks to the use of built 3D model of the world, as the robot can localize itself after each step and correct its pose.

Experiment

In the experiment, the robot follows the desired path using its internal odometry estimate. As soon as a result from modified SfMSeqv2 pipeline is available, i.e. after at least 5 captured images, the robot uses this result to correct its position estimate. Since the internal odometry is overly unreliable, only the pose from visual localization is used. This could be improved in the future by intelligently merging the information from odometry and visual localization to receive better and more robust results.

Results

The performed experiment verified the possibility of using visual localization and 3D scene reconstruction using the modified SfMSeqv2 pipeline for iterative navigation. In Figure 3.7 there is shown the path of the robot and the reconstructed points. Points 1–6 are reconstructed camera poses, when the visual localization of the robot was not possible due to small number of images. As of pose number 6, the pipeline was able to localize the robot and thus correct

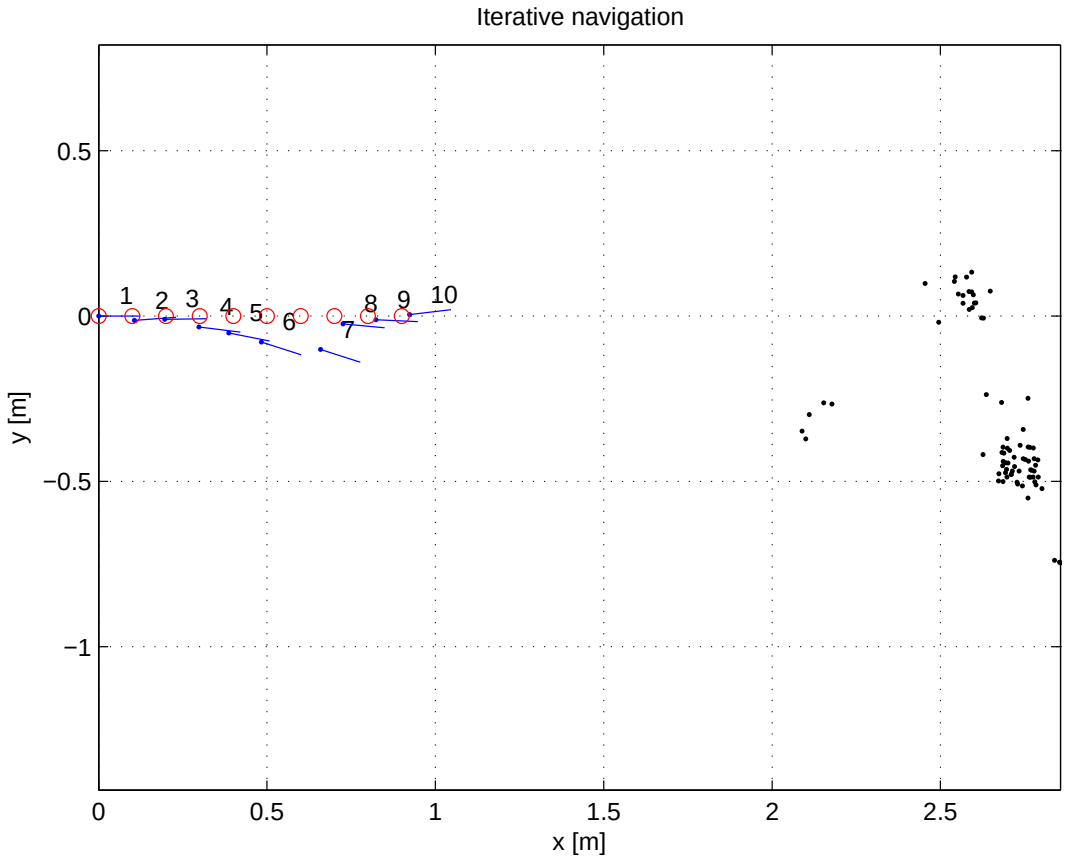


Figure 3.7: Robot iterative navigation (SLAM). Blue points and lines are camera poses computed using modified SfMSeqv2 pipeline, red circles denote the desired way–points, and black dots are points from the reconstructed scene.

its position estimate. This is shown in poses 7–10, as the robot slowly returns to the desired trajectory.

The process of iterative building of the scene reconstruction and robot localization is given in Figure 3.8. It can be seen, that the reconstructed 3D points and localized camera poses are recomputed in each step, as the modified SfMSeqv2 pipeline obtains more information about the perceived scene. The initial error is caused by small number of information from the images, since five images is only a minimum for the pipeline to work, not an optimal number.

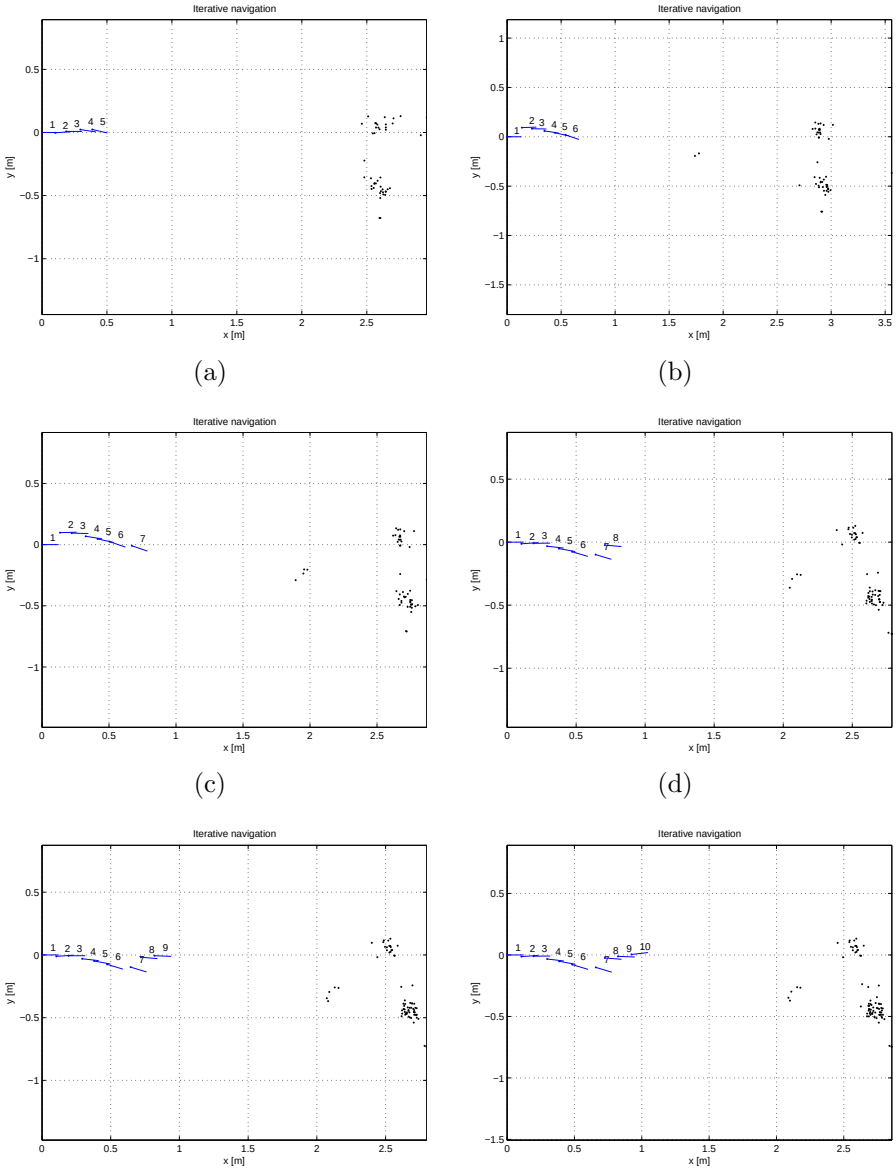


Figure 3.8: The process of iterative building the scene representation. The camera poses and reconstructed points positions are adjusted as more information about the scene is available from the captured images.

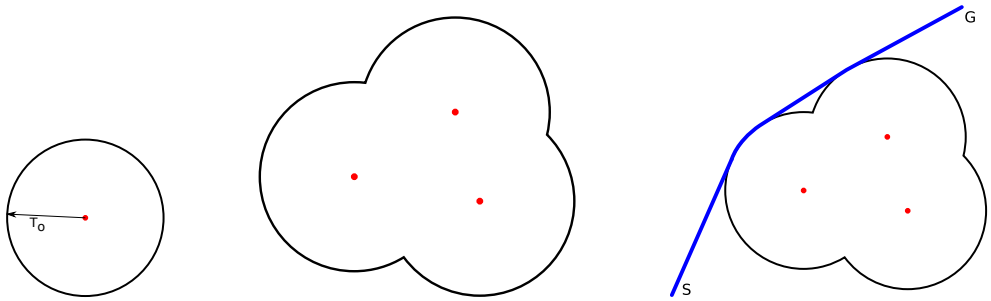
3.6 Obstacle avoidance

Obstacle avoidance is the ability of mobile robots to detect obstacles in their surroundings and deal with them. There are many ways of detection ranging from simple bumper sensors that detect collision, proximity sensors to laser or ultrasounds rangers and visual detectors. Sometimes it is sufficient, if the robot detects obstacles and reports this information, e.g. the robot detects closed door and asks for their opening. The more autonomous the robot is, the more it should be able to not only detect the obstacles but also remove or circumvent them if needed.

This thesis focuses on the visual detection of the obstacles. The obstacle detection is based on the 3D scene reconstruction. Thus only obstacles that are reconstructed using the previously described technique can be detected. This restricts the obstacles to (i) static objects, since only static objects can be successfully reconstructed using the employed algorithms, and (ii) objects that contain visual features traceable by the used detection methods. The distinction of obstacles and features detected on floor, e.g. tiles or knars, is based on the altitude of the corresponding reconstructed 3D point. A threshold T_a is chosen that divides the reconstructed points between obstacles and not-obstacles, such that if the altitude of a point is greater than T_a, it is classified as obstacle, and if it has the same or lower altitude, it is considered a reconstructed feature on the floor. Currently only an obstacle detection mechanism is implemented which stops the robot, if it gets too close to obstacles, i.e. the Euclidean distance in the x y plane is less than a threshold T_o. The threshold T_o is chosen so that it is at least the distance of robot camera to a most distant part of the robot's body. A safety margin should be added to this distance, in order to increase the clearance between the robot and obstacles.

For transferring the reconstructed 3D point cloud from Matlab® to control Python script the Pickle module is employed again. The reconstructed points produced by the pipeline are transformed into the robot's world coordinates and scaled, similarly as the cameras. Thus the points are determined by a vector $\begin{bmatrix} x_p & y_p & z_p \end{bmatrix}$, where x_p and y_p are the coordinates in the direction of x and y axis respectively and z_p is aiming downwards, such that points that have $z_p = 0$ are in the level of the first camera position. Thus the control script needs to remember the altitude of the camera h_0, when the first image was captured, and needs to subtract the two values $z = h_0 - z_p$ in order to obtain the point coordinates in the world coordinate system, i.e. where zero altitude is on the ground.

The implemented detection algorithm can be readily extended into obstacle aware trajectory planning, by enlarging each 3D point by a distance of T_o (see Figure 3.9). Obstacle enlargement is common in planning in mobile robotics



(a) Enlargement of one point

(b) Enlargement of a group of points

(c) Planned path

Figure 3.9: Reconstructed points enlargement for obstacle avoidance and planned path.

since it is possible to plan the trajectory for a point instead of a 2D shape, which simplifies this task greatly. The Figure 3.9(c) shows the planned path with 3 points as obstacles. Now the robot can follow this path directly and the reconstructed points will be circumvented.

The proposed principle of obstacles avoidance and path planning is able to produce a path that will evade all reconstructed 3D points classified as obstacles. The chosen obstacle radius T_o should be large enough so that no obstacles are hit, but should allow the robot to pass through a narrow corridor.

Chapter 4

Documentation

This chapter concerns the user point of view of the presented algorithms and applications. It is meant for people, that want or need to use either the Capture module or the modified SfMSeqv2 pipeline. It should serve as a user manual and give enough information on the usage. Full source code of all applications is given on the attached CD, see Appendix B.

4.1 Capture module

This section covers the module application interface and examples of usage. For instructions about compilation and cross-compilation, please see Aldebaran documentation [2].

4.1.1 Interface

The module offers a few public functions, that allow controlling all the module functionality. The three most important public functions provided by this module are briefly described in the following list

- **setParam(paramName, paramValue)** sets value of one of the several parameters determined by *paramName*. Only one parameter can be set during one call to this function. Available parameter names are
  - calibration—enables capturing of a middle row of images that connect the top and bottom row
  - chr—horizontal resolution, i.e. number of images in a row, defaults to 3 (left, front, and right)
  - pitch—pitch of robot's head

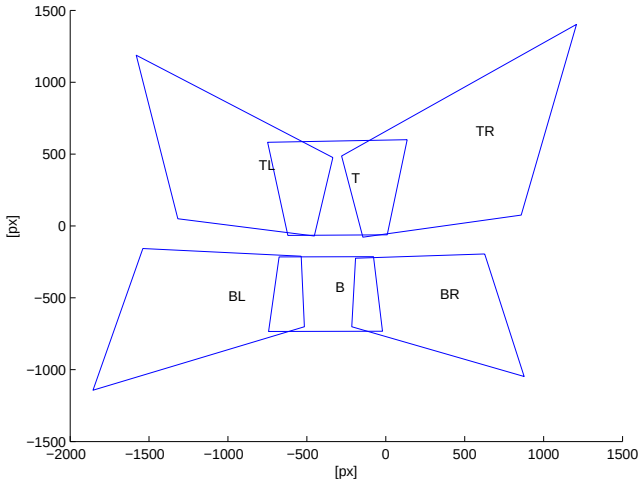


Figure 4.1: Double panorama, as captured by the robot. Abbreviations name the positions of images: B-bottom, T-top, L-left, and R-right and their combinations.

  - autoExposition—enable/disable automatic exposition
  - autoWhiteBalance—enable/disable automatic white balance
  - autoGain—enable/disable automatic gain
  - gain—set cameras gain (0-255)

- **run(series)** captures images for creating a panorama. An example of computed layout of the images in the panorama is given in Figure 4.1. *series* parameter is used for the panoramas numbering.

- **takeOneImg(cameraID)** captures a single image by camera determined by *cameraID*, where 0 is the top and 1 the bottom camera.

The module captures images in the BGR color space, which is the default color space of OpenCV, and in the highest resolution available, i.e. 640×480 px. These are default settings and currently not available for modification using the *setParam* function.

4.1.2 Usage

Since the module template is generated by a python script, shipped along with the Aldebaran software development kit, its usage is similar to other Aldebaran's modules, thus reading the generic help [2] is useful for using this module.

Algorithm 4: Capture module usage example I: camera settings and image capturing using Python.

```
from naoqi import ALProxy
IP = "192.168.0.1"
PORT = 9559
BOTTOM_CAMERA_ID = 1
# Connect to proxy
try:
        capture = ALProxy("Capture", IP, PORT)
except Exception,e:
        print "Error while creating Capture proxy:
        print str(e)
        exit(1)
# Capture module parameters
capture.setParam("pitch", -38.4431) # head pitch [degrees]
capture.setParam("autoExposition", 0)
capture.setParam("autoWhiteBalance", 0)
capture.setParam("autoGain", 0)
capture.setParam("gain", 0)
# Capture image and print its path
imgname = capture.takeOneImg(BOTTOM_CAMERA_ID)
print imgname
```

Algorithm 5: Capture module usage example II: Capturing images for panorama from a C++ code

```
captureProxy = pBroker->getProxy("Capture");
captureProxy->callVoid("setParam", "pitch", -36.5);
ALValue filenames = captureProxy->call<ALValue>("run", 1);
std::string filename;
for (uint i = 0; i < filenames.getSize(); i++) {
filename = (std::string) filenames[i];
printf("%s\n", filename.c_str());
}
```

The Algorithm 4 gives an example of the module usage in a Python script. Firstly, a connection to the Capture proxy is established, through which all the following interaction with the module passes off. Several camera parameters are set, e.g. automatic exposition, white balance, and gain are turned off and finally one image is taken by the bottom camera. The image filename in this example is then printed to the standard output for later retrieval.

The module can be also called from C++ applications, as shows the Algorithm 5, where images for one panorama are captured, with robot's head pitch set to $-36.5°$, and their names are again printed to standard output.

4.2 Modified SfMSeqv2

This section describes the modified version of SfMSeqv2, as presented in Section 2.3. It wraps multiple functionality in a single Matlab® script and provides an easy to use interface. The original pipeline accepts a sequence of undistorted images capturing a rigid scene motion or a sequence of images of static scene captured by a moving camera, processes them, and outputs results in the form of a plot of camera poses and of a VRML model of cameras and 3D points. This is useful for processing images off-line, to reconstruct camera path or scanned scene, but for real-time processing, this is not applicable. The modified SfMSeqv2 pipeline aims to produce results on the fly, which can be used in the process of images capturing and thus close the feedback loop. Moreover the modified pipeline does not need undistorted images as the process of undistorting images is already incorporated into it.

Pseudocode of the modified SfMSeqv2 pipeline is given in Algorithm 6. It periodically checks for new images in the input directory and after undistorting them, it processes them similarly as the original pipeline. If there are not enough images for SfM computing, the pipeline interrupts the loop and waits for more images. As soon as the number of images is sufficient, the model is computed and camera poses are determined.

4.2.1 Usage

The result is the same as in the original pipeline, except that the modified version also produces a list of camera poses in a form easily usable by Python scripts. Each camera pose is determined by the name of image captured from that pose. Thus it is possible, to find out robot's pose in every step, where an image was taken. If a particular image (camera pose) cannot be localized with respect to the built model, the pipeline produces vector with NaN values (Not a Number). The robot control script should be able to deal with such a

Algorithm 6: SfMSeqv2 adjusted for iterative image processing.

```
initialize SfMSeqv2;
load calibration matrices;
while true do
    undistort new images;
    detect descriptors;
    find tentative correspondences;
    compute epipolar geometries;
    if number of images < 5 then
        print('Not enough images for SfM');
        continue;
    end
    compute tracks from epipolar geometries;
    compute SfM from tracks;
    add skipped cameras;
    loop closing;
    bundle adjustment (SBA);
    send results to Python script;
    export VRML model and draw plots;
end
```

result and command the robot without precise knowledge about its position, preferably to a pose, from where the localization would be successful again.

The pipeline's entry point is in the `main.m` file, that should be launched using Matlab®. Prior to running the pipeline, it is required to set paths and other variables in the `init_sfmseqv2.m`. The original pipeline source files are needed, as some scripts are called from the `main.m` file.

4.2.2 Summary

The modified pipeline embodies functionality for all the presented scenarios (i) it creates the model of the surroundings, (ii) images that are matched with the created 3D representation yield a camera pose, thus localize the robot. Navigation itself is carried out by the control script, the modified pipeline merely presents the results needed.

Chapter 5

Conclusion

This chapter summarizes all the results, that were achieved in this thesis. It presents the workarounds, that needed to be employed in order to capture usable images, lists the applications developed for image capturing and processing and briefly discusses the results of the experiments.

The goal of this thesis is to develop algorithms that would allow humanoid robot Nao to reconstruct the surrounding world using sparse 3D point cloud reconstruction, localize itself with respect to this 3D representation and be able to navigate to a desired position.

It is shown that due to the low quality of robot's cameras, several workarounds are needed to achieve necessary conditions for using the visual system. First of the workarounds is the *stop and go* fashion of motion and images capturing, which avoids capturing blurred and scattered images, but on the other hand increases the time needed for traversing from one place to another. This waste of time is reduced by using the pauses to transfer the images, which allows for parallelism, since the images can be processed as the robot walks.

Second workaround is needed because of the low cameras resolution and sensitivity. The environment, in which the robot is navigating, is enhanced with artificial objects, that increase the diversity and heterogeneity of the perceived scene and introduce a great number of visually traceable features. Moreover, sufficient lighting conditions are needed for reliable results, but ordinary ceiling illuminants are usually adequate.

Thirdly, there is proposed a way of extending cameras field of view that connects several horizontally overlapping images into panoramas.

There are a few support applications, that were developed in order to streamline the work with robot Nao and to ease its usage.

The Monitor application observes the temperatures of individual robot's

joints and presents them in a simple and effective way to the user. It is advisable to watch the joints temperatures, as the robot may fall or otherwise damage itself, if they get too hot.

The Capture module handles images capturing and allows for easy setting of cameras parameters, such as gain, auto-exposition, and auto-white balance. It also controls the head movement.

The state of the art image processing pipeline SfMSeqv2 developed at CMP is modified to allow iterative processing of the captured images. The SfMSeqv2 pipeline computes Structure from Motion from a sequence of images. The modified version is also equipped with a communication mechanism, to be able to exchange information with other applications, such as the script that controls the robot and transfers the images from robot to PC.

Localization with respect to a built world representation and navigation of the robot are vital abilities of almost every mobile robot. In this thesis it is shown that with the modified version of SfMSeqv2 pipeline it is possible to process the images captured by the robot in real-time. This enables to use the results, i.e. 3D world representation and camera poses, to adjust robot's path and goals while the robot is walking. This results in a visual SLAM (vSLAM) technique. Obstacle detection from the reconstructed 3D point cloud is implemented and a proposal of obstacle avoidance mechanism and path planning is described.

The presented ideas and developed algorithms can be used in many applications that require a visual localization, iterative 3D scene reconstruction or vSLAM functionality. The presented approach is not limited to humanoid robots and can be employed for navigation of any mobile robot or other device equipped with camera.

Bibliography

[1] Aldebaran Robotics. http://www.aldebaran-robotics.com/, 2011.

[2] Aldebaran robotics online documentation. http://robocup.aldebaran-robotics.com/, 2011.

[3] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.

[4] J. Borenstein, H. R. Everett, and L. Feng. *Where Am I? Systems and Methods for Mobile Robot Positioning*. Edited and compiled by J. Borenstein, 1996.

[5] Center for Machine Perception at Czech Technical University in Prague. http://cmp.felk.cvut.cz, 2011.

[6] H. Choset, W. Burgard, S. Hutchinson, G. Kantor, L. E. Kavraki, K. Lynch, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, June 2005.

[7] S. Engelson and D. McDermott. Error correction in mobile robot map learning. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2555 –2560 vol.3, May 1992.

[8] European commission - CORDIS, project Humavips. http://cordis.europa.eu/fetch?CALLER=FP7_PROJ_EN&ACTION=D&DOC=1&CAT=PROJ&QUERY=012e01367bda:f86c:28cf7860&RCN=93630, 2011.

[9] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, June 1981.

[10] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[11] M. Havlena, Š. Fojtů, D. Průša, and T. Pajdla. Towards Robot Localization and Obstacle Avoidance from Nao Camera. Technical report, Center for Machine Perception, K13133 FEE, Czech Technical University in Prague, 2010.

[12] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of Honda humanoid robot. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1321 –1326 vol.2, May 1998.

[13] A. Hornung, K. M. Wurm, and M. Bennewitz. Humanoid Robot Localization in Complex Indoor Environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[14] HUMAVIPS web page. `http://humavips.inrialpes.fr`, 2011.

[15] M. I. A. Lourakis and A. A. Argyros. The design and implementation of a generic sparse bundle adjustment software package based on the Levenberg-Marquardt algorithm. Technical report, Inst. of Computer Science-FORTH, Heraklion, Crete, Greece, 2004.

[16] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91, Nov. 2004.

[17] P. Mareček. A camera calibration system. Master's thesis, Center for Machine Perception, K13133 FEE, Czech Technical University in Prague, 2001.

[18] J. Matas, Š. Obdržálek, and O. Chum. Local Affine Frames for Wide-Baseline Stereo. In *Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 4*, ICPR '02, Washington, DC, USA, 2002. IEEE Computer Society.

[19] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *In VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[20] R. R. Negenborn. Kalman Filters and Robot Localization, 2003.

[21] S. Nishio, H. Ishiguro, and N. Hagita. Geminoid: Teleoperated android of an existing person. *Humanoid robots-new developments. I-Tech*, 2007.

[22] S. Oßwald, A. Hornung, and M. Bennewitz. Learning Reliable and Efficient Navigation with a Humanoid. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2010.

[23] N. Snavely, S. M. Seitz, and R. Szeliski. Photo Tourism: Exploring image collections in 3D. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006), 2006.

[24] A. Torii, M. Havlena, and T. Pajdla. From Google Street View to 3D city models. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 2188 –2195, 2009.

Appendix A

Humanoid robot Nao

The robot weights about 5 kg and is approximately 58 cm tall. It can be powered by a power plug or by batteries, which are 6 Li-Ion cells in series, $U_n = 21.6$ V. Their capacity is 2 Ah. Network access is provided by a Wi-Fi (IEE 802.11g) or Ethernet connection.

Degrees of freedom

The version of the robot used in this thesis has 27 degrees of freedom, namely 2 in head, 5 in each arm, 1 is the pelvis, 5 in each leg, and 2 in each hand.

Audio

The robot is equipped with 2 loudspeakers, with diameter of 36 mm, which are placed in its ears. There are also 4 microphones placed around the head with frequency range from 300 Hz to 8 kHz.

Actuators

The joints are driven by Coreless MAXON DC motors. They are equipped with encoders: 36 hall effect sensors with 12 bit precision, i.e. 4096 values per turn corresponding to precision of about 0.1 °.

Sensors

The robot carries a bunch of various sensors. There are 32 Hall effect sensors in the joints, 2 one axis gyrometers, 1 three axis accelerometer, and 2 bumpers located at the tip of each foot, which are simple ON–OFF switches. There are

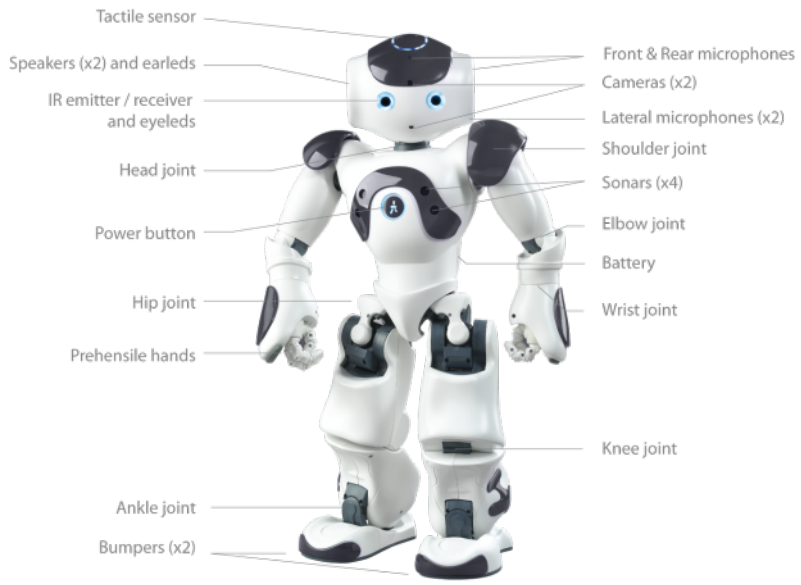


Figure A.1: Robot Nao (courtesy of Aldebaran Robotics [2]).

also 2 ultrasound sonars on the robot's chest and an array of capacitive sensors on Nao's head.

Robots cameras are VGA CMOS, with resolution of 640×480 px. Maximal framerate is 30 fps. Focus range is from 30 cm to infinity. The field of view is 58° on the diagonal.

LED

Many LEDs can be used to signalise information to users or operators. There are 2 times 8 RGB LEDs in its eyes, 2 times 10 blue LEDs with 16 levels of intensity in the ears, 1 RGB on the chest and 2 RGB on the feet.

Motherboard

The employed version is driven by a x86 AMD GEODE 500 MHz CPU, with 256 MB SDRAM and 2 GB of flash memory.

Embedded software

Operating system is an embedded GNU/Linux (32 bit x86 ELF) based on a custom OpenEmbedded distribution. It is possible to use the system from C, C++, Python, and Urbi programming languages.

Appendix B

CD content

The following tree structure shows the content of the attached CD. There are listed all folders and files in the root level and in the lower levels only important folders are given.

- `/`
  - `Capture` ..... source codes of the Capture module
  - `Matlab`
    - `naoloc` ..... modified SfMSeqv2 pipeline
      - `init_sfmseqv2.m` ..... variables and path settings
      - `main.m` ..... Matlab® script that launches the modified pipeline
    - `SfMSeqv2` ..... original pipeline
  - `scripts` ..... several Python scripts for robot control
  - `thesis` ..... thesis source codes
  - `thesis.pdf` ..... this thesis