Multi-view camera recalibration

MSc. Thesis VIBOT

Pablo Speciale

Le2i - Laboratoire Electronique, Informatique et Image,
University of Burgundy

A Thesis Submitted for the Degree of
MSc Erasmus Mundus in Vision and Robotics (VIBOT)

· 2013 ·

Abstract

Robot calibration is an important problem to be solved in any robot. Once the robot is calibrated, it can effectively interact with the environment. This work is focused in calibration of multiple cameras mounted in a robot. Combining measurements from different cameras, an optimization process is defined. Then, a solution is obtained by minimizing the reprojection error with a non-linear solver, that is, a bundle adjustment approach. As additional constraints, it is assumed that one camera is already calibrated with respect to the robot, and an initial calibration from the robot model is provided.

La única lucha que se pierde es la que se abandona...
(The only fight we lose is the one we abandon)

Ernesto Che Guevara

Contents

1 Introduction **1**

1.1 Thesis organization 1

1.2 Goals 2

1.3 Related Work 2

2 Background **3**

2.1 Theoretical background 3

2.1.1 Pinhole camera model 3

2.1.2 Bundle adjustment 3

2.1.3 Triangulation 4

2.2 Technical background 5

2.2.1 ROS 5

2.2.2 KDL 12

2.2.3 Ceres Solver 13

2.2.4 OpenCV 14

3 Multi-view recalibration **16**

3.1 Overview 16

3.2 Acquisition process 17

3.2.1 Data collection 17

3.3 Visualization . . . 18
3.3.1 Visual Markers . . . 18
3.4 Estimation process . . . 20
3.4.1 Initialization . . . 20
3.4.2 Optimizer . . . 22
3.4.3 Update URDF . . . 22
3.4.4 Results . . . 23

4 Implementation details **26**
4.1 Software engineering decisions . . . 26
4.1.1 Why to re-implement the calibration estimation package? . . . 26
4.1.2 Why KDL? . . . 27
4.1.3 Why to re-implement `robot_state_publisher`? . . . 27
4.1.4 Quaternions . . . 27
4.2 Code . . . 28
4.2.1 Cost function . . . 28
4.2.2 N-View Triangulation . . . 29
4.3 Problems and solutions . . . 29
4.3.1 Visible cameras . . . 29
4.3.2 Visual Markers . . . 30
4.3.3 Robot update . . . 30
4.3.4 Multi-view triangulation . . . 31
4.4 Stats . . . 32

5 Additional work **33**
5.1 Preliminary . . . 33
5.1.1 Essential matrix . . . 33
5.1.2 Camera resection . . . 34
5.2 Proposed method . . . 34

5.2.1 Demonstration 35
5.2.2 Resection generalization 35
5.2.3 Scenario 36
5.2.4 Application 37
5.2.5 Advantages / disadvantages 37
5.2.6 Open questions 38

6 Future work **39**

7 Conclusions **40**

A Lemma demonstrations **42**

Bibliography **46**

List of Figures

1 Three points X_1, X_2, X_3 are observed by three cameras P_1, P_2, P_3. 4

2 RViz interface. 6

3 PR2 overview 7

4 PR2 URDF model in RViz 7

5 Example of /tf in RViz (in a particular time). On the left, all coordinate systems; on the right, /tf path from `narrow_stereo_optical_frame` until the robot root (`base_footprint`). 8

6 rqt: ROS bag interface. 8

7 rqt: Topic introspection. 9

8 De-mosaics and undistorts the raw camera image stream. 9

9 The camera coordinate system assumed. 10

10 Derivation diagram 11

11 Rectification (and de-Bayer) process performed by `image_proc_node`. 12

12 Example of KDL tree. 12

13 Rodrigues' formula 14

14 High level flow. 16

15 Coordinate system of all the cameras in the PR2 head. 17

16 The PR2 holding a checkerboard pattern on the left. 18

17 Example of *one* sample from 4 different views, using the large checkerboard. . . . 19

18 solvePnP results for the all the cameras (same checkerboard). 19
19 Estimation process. 20
20 Relative transformations. 21
21 Introducing *camera father* notation. 22
22 Old calibration (red) vs new calibration (blue). Dashed arrow is the *calibration error*. 23
23 Checkerboard in hand. 23
24 Checkerboard free in the world. 24
25 Visible bias on first camera (with and without `prosilica` camera) 24
26 Reprojection error after optimization. 25
27 Median, mean, std after optimization. 25

28 Checkerboard in hand visible only for 3 cameras. 30
29 Initial versions 31
30 Triangulation points (white dots) must be closer to solvePnP solutions (color dots) 31
31 Github stats 32

32 A reconstructed point X is in front of both cameras only in (a). 34
33 Stereo pair with and without perturbation. 36

34 PR2 pleased to be free. 40

Chapter 1

Introduction

In robotics, sensor calibration is extremely important since a robot needs its sensors to measure and interact with the environment. Examples where calibration is necessary are:

- **Manipulation**: while grasping, or interacting with the environment using its arms, object detection is a common task. An uncalibrated robot will misgrasp the object leading to a failure.
- **Sensor fusion**: suppose that a sensor produces 3D data, and another sensor, color images. It might be desired to fuse both measurements in order to create a color 3D model. A correct fusion cannot be achieved without a calibrated robot.

In this thesis calibration of multiple cameras will be studied. The term *recalibration* refers to the existence of an initial calibration, provided by the robot model description. Another important assumption is that one camera is already calibrated with respect to the robot. This camera will be the *reference camera*.

1.1 Thesis organization

The background is presented in chapter 2, and it is divided in two main sections: first, a *theoretical background* will be explained in section 2.1; second, a *technical background* in section 2.2, where ROS (Robot Operative System) [1], RViz (ROS visualizer) [2], PR2 (Personal Robot 2, from Willow Garage) [3], KDL (Kinematics and Dynamics Library) [4], Ceres Solver (non-linear least squares minimizer) [5], and other infinity amount of tools will be briefly explained.

Chapter 3 explains the process and methods developed in this thesis. **Real data** obtained from PR2 is used to calibrate the *multiple cameras*; experimental results are also presented.

Implementation details will be discussed in chapter 4. Motion initialization is analyzed in chapter 5, where a novel method is presented and directions for further research are provided. Followed by *future work* in chapter 6, and *conclusions* in chapter 7.

1.2 Goals

The goals on this work are:

- the creation of a ROS calibration package for multiple cameras, such as RGB cameras, Microsoft Kinect, Prosilicas (high-definition camera), and other cameras mounted in a robot; in our test case, mounted in the PR2 head;
- the estimation of the best relative position between the cameras, using 2D measurements extracted from a known beforehand pattern (checkerboard). The relation among cameras is supposed to remain rigid over time. Even if the robot or its joints move, they are all located in the PR2 head which is a rigid entity.

1.3 Related Work

Many people have developed specific techniques for pairwise calibration between sensors. In [6], line constraints are used to calibrate a 2D laser range-finder to a camera. Similar work was done in [7], but focused on using plane constraints from a 3D laser range-finder system. A more complex case is when an actuated camera system needs to be calibrated (hand-in-eye system) and has been addressed in [8]. Articles [9] and [10] were particularly useful to understand fundamental concepts. In addition, the PR2 calibration package is described in [11].

Chapter 2

Background

The background will be divided in two sections: a **theoretical background** discussed in section 2.1, and a **technical background** in section 2.2.

2.1 Theoretical background

2.1.1 Pinhole camera model

A 2D point is denoted by $\mathbf{x} = [x, y]^T$. A 3D point is denoted by $\mathbf{X} = [X, Y, Z]^T$. It is used $\mathbf{x} = [x, y, 1]^T$ to denote the point in homogeneous coordinates (augmented vector by adding 1 as the last element) and $\mathbf{X} = [X, Y, Z, 1]^T$. A camera is modeled by the usual pinhole: the relation between a 3D point $\mathbf{X}$ and its image projection $\mathbf{x}$ is given by

$$s\,\mathbf{x} = K\,[R \;\; t]\,\mathbf{X} \quad \text{with } K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

where s is an arbitrary scale factor; (R, t) called the extrinsic parameters, is the rotation and translation which relates the world coordinate system to the camera coordinate system; K is called the camera intrinsic matrix, and (c_x, c_y) are the coordinates of the principal point, f_x and f_y are the focal lengths expressed in pixel units.

2.1.2 Bundle adjustment

Estimation of relative poses of several cameras, or many poses of a single moving camera, is a common problem. It is often solved by jointly estimating the set of camera poses along with 3D

features that are detected by some set of cameras. This approach is called bundle adjustment (BA) [12].

Bundle adjustment algorithms take the following three data as inputs (see Figure 1):

- a set of n 3D point positions $X_1, X_2, \ldots, X_n$,
- m camera parameters $P_1, P_2, \ldots, P_m$,
- and the positions of the projections x_{ij} (measurements) of the points Xi in the camera Pj where they are visible.

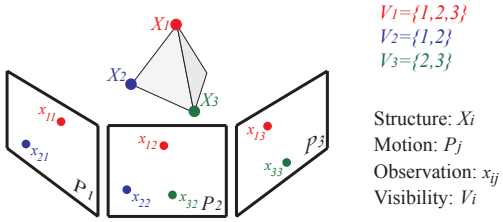


Figure 1: Three points X_1, X_2, X_3 are observed by three cameras P_1, P_2, P_3.

Bundle adjustment optimizes both the scene X_i and camera parameters P_j by minimizing the sum of squared reprojection errors:

$$\sum_{i=1}^{n} \sum_{j \in V_i} ||x_{ij} - P_i X_j||^2 \tag{2}$$

Note: Ceres Solver has been utilized to solve this optimization problem, see section 2.2.3 for a short introduction; and the cost function used in the estimation process is detailed in section 4.2.1.

2.1.3 Triangulation

A simple linear triangulation method is described [13]. Measurement $\mathbf{x} = P\,\mathbf{X}, \mathbf{x}' = P'\,\mathbf{X}$ can be combined to form a linear equation: $A\,\mathbf{X} = 0$. The homogeneous scale factor is eliminated by a cross product $\mathbf{x} \times P\,\mathbf{X} = 0$ to give equations that form the following equation system:

$$A = \begin{pmatrix} x\mathbf{p}^{3T} - \mathbf{p}^{1T} \\ y\mathbf{p}^{3T} - \mathbf{p}^{2T} \\ x'\mathbf{p}^{3T} - \mathbf{p}^{1T} \\ y'\mathbf{p}^{3T} - \mathbf{p}^{2T} \end{pmatrix} \tag{3}$$

where $\mathbf{p}^{iT}$ are the rows of P. Then the system $A\,\mathbf{X} = 0$ is solved with SVD[1] as it is explained in [13].

N-View Triangulation

It is easy to extend this method for multiple views by adding more rows to the matrix A. Implementation in section 4.2.2.

2.2 Technical background

In this section, a briefly explanation of the used tools is given: ROS (Robot Operating System) [1], RViz (ROS visualizer) [2], KDL (Kinematics and Dynamics Library) [4], Ceres Solver (non-linear least squares minimizer) [5] and others.

References to the official documentation is provided for further reading. This chapter does not intend to be a full documentation of each component, since their development is very dynamic.

This chapter is focused in individual explanations, and in further chapters, the relation between these tools will be provided.

2.2.1 ROS

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.

The primary goal of ROS is to support code reuse in robotics research and development. ROS is a **distributed framework** of processes (aka *Nodes*) that enables executables to be individually designed. These processes can be grouped into **Packages**.

The ROS runtime *graph* is a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure. ROS also implements several different styles of communication.

In summary, ROS has two basic *sides*:

- The operating system side ROS as described above,
- and a suite of user contributed packages (organized into set of nodes) that implement functionalities such as: simultaneous localization and mapping, planning, perception, simulation.

[1] SVD: Singular Value Decomposition

The main goal of this thesis is to be a **ROS calibration package**, in replacement of the existent one [14]. For further learning about ROS, an excellent documentation can be found in: www.ros.org.

Nodes and Topics

Packages is a set of nodes, and topics are named buses over which nodes exchange messages.

RViz

It is a 3D visualization tool for ROS. It is used to visualize **/tf** (section 2.2.1) and **URDF** robot models (section 2.2.1). Sending messages with basic shapes (cube, sphere, cylinder, arrow) to RViz is also possible. These Visual markers are used to display 3D points, see Figure 2. For more information: http://www.ros.org/wiki/rviz.

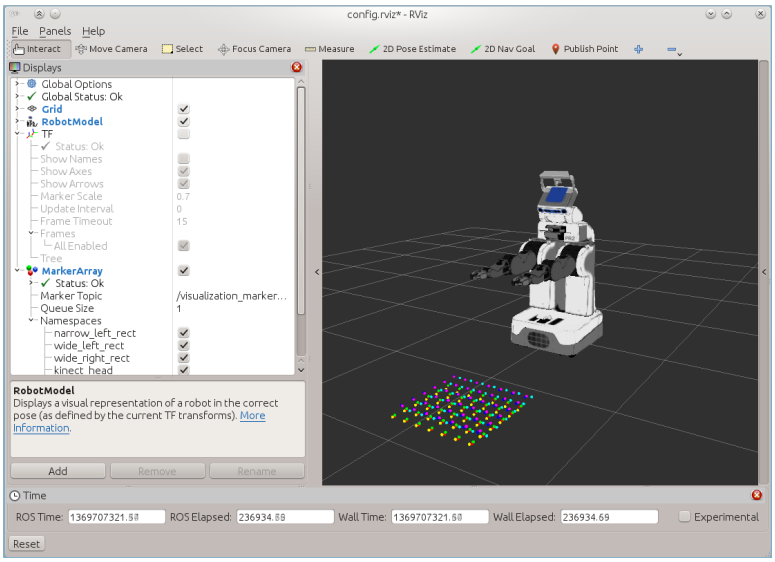


Figure 2: RViz interface.

Roscpp

Another primary ROS goal is *"language independence"*: the ROS framework is easy to implement in any modern programming language, and it is already implemented in:

- Python (http://www.ros.org/wiki/rospy) and,
- C++ (http://www.ros.org/wiki/roscpp).

PR2

The PR2 (Personal Robot, version 2) is a Willow Garage which has two 7-DOF arms with a payload of 1.8 kilograms (1,800 g). Sensors include a 5 megapixel camera, a tilting laser range finder, and an inertial measurement unit. The "texture projector" projects a pattern on the environment to create 3D information by the cameras. Also Microsoft Kinect can be head-mounted. In this work, its head (see left on Figure 3) is the must important since all the cameras are located there and must be calibrated to each other.

Figure 3: PR2 overview

URDF

The URDF (**U**nified **R**obot **D**escription **F**ormat) is a XML format for robot model representation. The package also contains a C++ parser, and it can be visualized with RViz (see Figure 4). For more information: http://www.ros.org/wiki/urdf.

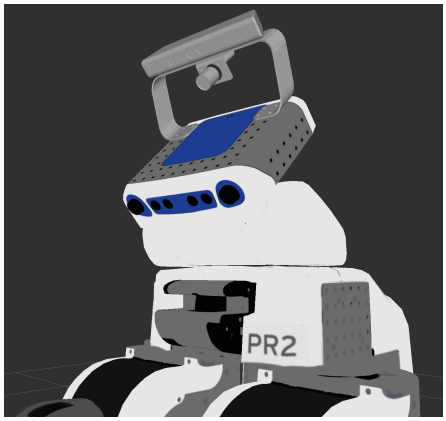


Figure 4: PR2 URDF model in RViz

/tf

The package /tf lets the user keep track of multiple coordinate frames over time. /tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, other geometric entities between any two coordinate frames at any desired point in time.

A robotic system typically has many 3D coordinate frames that change over time (see Figure 5), such as a world frame, base frame, gripper frame, head frame, etc. /tf keeps track of all these frames over time. For more information: http://www.ros.org/wiki/tf.

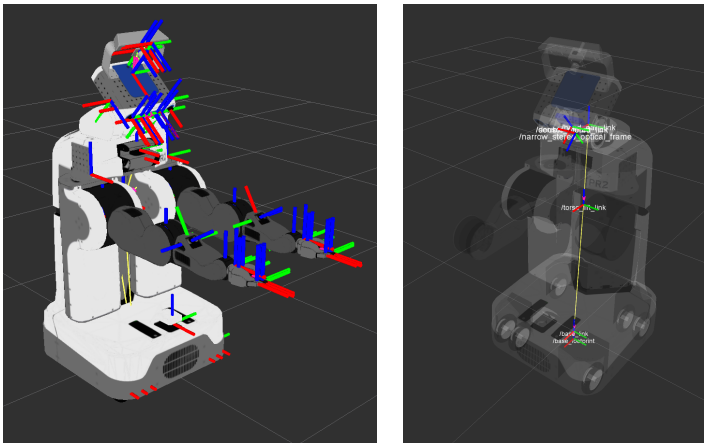


Figure 5: Example of /tf in RViz (in a particular time). On the left, all coordinate systems; on the right, /tf path from `narrow_stereo_optical_frame` until the robot root (`base_footprint`).

ROS bag and rqt

All the components for a simulated environment are implemented and, it is needed, a way to replay events/messages. This is performed by ROS bag, which is a set of tools for recording from and playing back to ROS topics. It avoids deserialization and reserialization of the messages. More info: http://www.ros.org/wiki/rosbag.

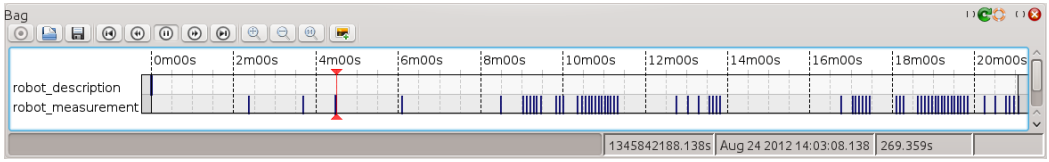


Figure 6: rqt: ROS bag interface.

A second tool described here is a rqt plugin which allows to play a ROS bag easily (Figure 6). The user will be able to see the ROS messages in a convenient way (Figure 7). More info: http://www.ros.org/wiki/rqt.

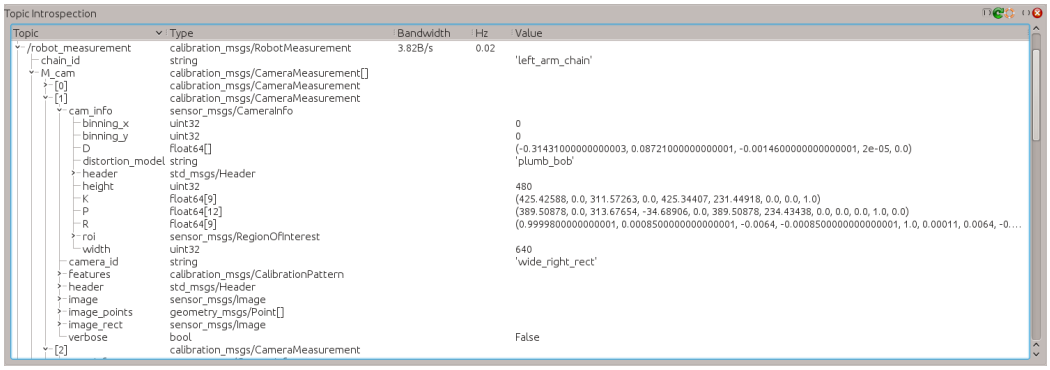


Figure 7: rqt: Topic introspection.

Image_proc_node

The node `image_proc` sits between the camera driver and vision processing nodes (see Figure 8). `image_proc` removes camera distortion from the raw image stream, and if necessary will convert Bayer or YUV422 format image data to color

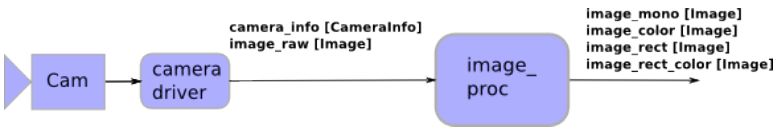


Figure 8: De-mosaics and undistorts the raw camera image stream.

All processing is on demand. Color processing is performed only if there is a subscriber to a color topic. Rectification is performed only if there is a subscriber to a rectified topic. While there are no subscribers to output topics, `image_proc` unsubscribes from the `image_raw` and `camera_info` topics.

More info: http://www.ros.org/wiki/image_proc.

Camera model in ROS

Figure 9 is a diagram of the camera coordinate system assumed. It is a right-handed system, with the world X and Y aligned with the image x and y.

The camera model used in ROS is the same coordinate system used in OpenCV. It differs from the coordinate system of Harley and Zisserman in [13], which has Z forward, Y up, and X to the left (looking towards +Z).

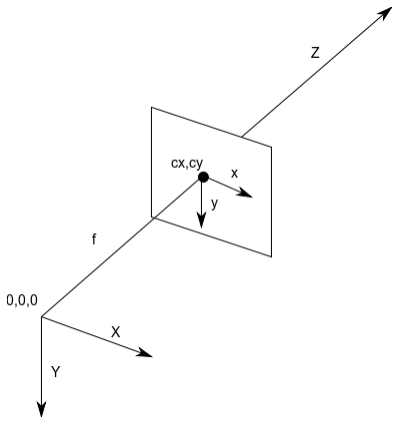


Figure 9: The camera coordinate system assumed.

More info: http://www.ros.org/wiki/image_pipeline/CameraInfo.

Derivation diagram

Figure 10 is a diagram of the various images that are conceptually derivable from the CameraInfo message[2]. The boxed variables are present in this message.

Projection onto the original image

Starting with an initial 3D point X, the position of the corresponding image point is found by going right-to-left in the upper half of the diagram 10. The process is summarized by the following equations:

$$\begin{aligned} X' &= [R,t]\,X && \texttt{transform} \\ sx &= X' && \texttt{projection} \\ x* &= d(x) && \texttt{distortion} \\ q &= Kx* && \texttt{pixel coordinates} \end{aligned}$$

This process uses the K camera matrix and D distortion vector. Ignoring the transform T, the projection of a point X' can be found into the original camera frame using these equations. First, the $3D$ point X' is projected onto the normalized, undistorted image via a projection

[2]CameraInfo message is a ROS message where all the camera information is saved.

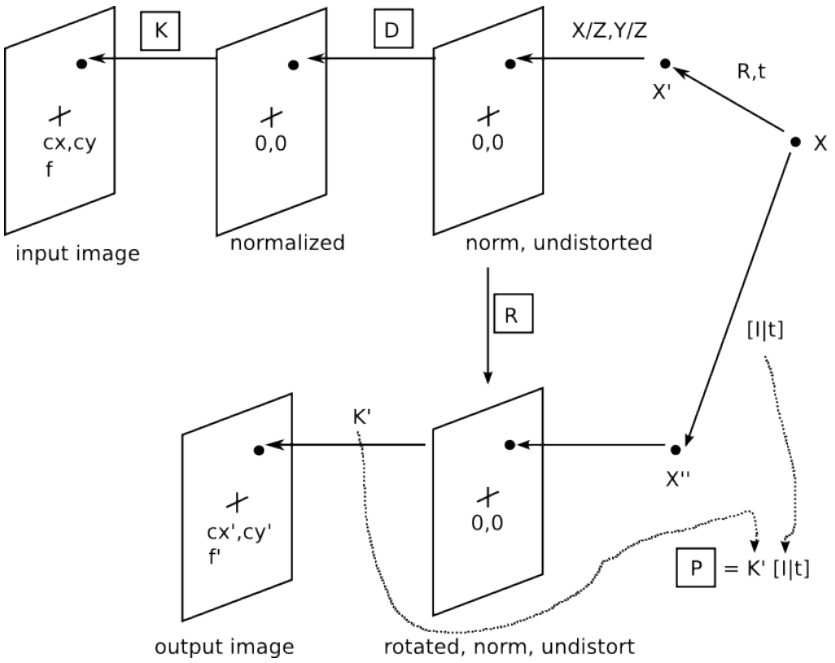


Figure 10: Derivation diagram

operation (division by Z). Then the distortion coefficients are used in the function $d()$ to move the point to its distorted position, still in a normalized image. Finally, the normalized image is converted to a pixel-coordinate image by applying the camera matrix to each image point.

Rectification

In this context, rectification is the process of transforming the input image into an output image with the distortion corrected (Figure 10), and optionally transformed by rotation, in-plane translation, and scale.

In the case of a **monocular rectification** with distortion correction, to transform a pixel in the input image into one in the output image, it is sent through the $K-D-R-K'$ series of transformations. $K-D$ gets to the normalized, undistorted image; the rotation R is the identity because we don't want to rotate the image; and then K' converts back to pixel coordinates in the output image (an example in Figure 11). In this case, since there is no rotation, translation, or scaling from the original image, $K = K'$, and only the K and D elements of CameraInfo are needed.

(a) **image_raw**: original camera image, Bayered and distorted

(b) **image_rect**: rectified image, de-Bayered and undistorted (amount of black border may vary depending on calibration)

Figure 11: Rectification (and de-Bayer) process performed by `image_proc_node`.

2.2.2 KDL

KDL (**K**inematics and **D**ynamics **L**ibrary) defines a tree structure to represent the kinematic and dynamic parameters of a robot mechanism (see Figure 12). *kdl_parser* provides tools to construct a KDL tree from an XML robot representation in URDf format.

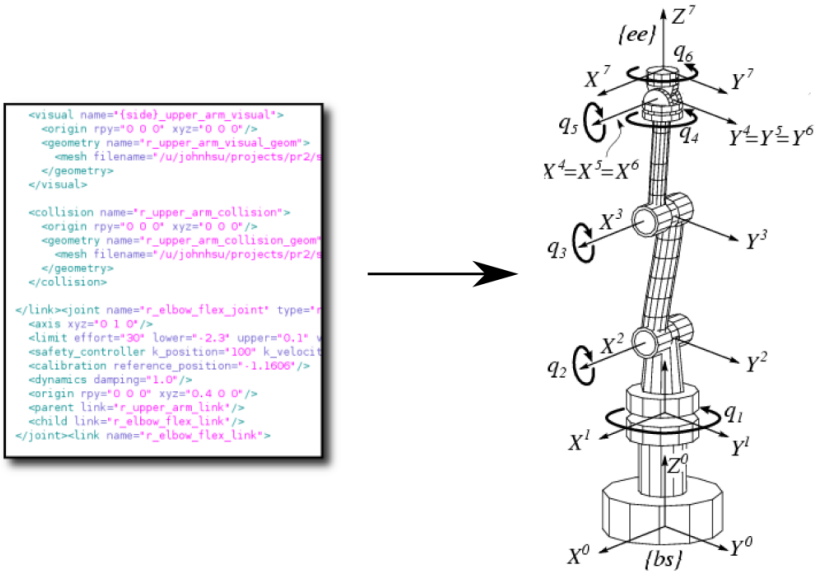


Figure 12: Example of KDL tree.

This library is mainly used for:

- 3D frame and vector transformations,
- forward kinematics (given joint angles compute the position of the end-effector[3]).

For more information: http://www.ros.org/wiki/kdl.

2.2.3 Ceres Solver

Ceres Solver [5] is a portable C++ library that allows modeling and solving large complicated **non-linear least squares** problems. It is used at Google to estimate the pose of Street View cars, aircrafts, and satellites; to build 3D models for PhotoTours; to estimate satellite image sensor characteristics, and more.

Ceres Solver is an important component in this thesis because it has very good features like, among others:

- A friendly API: build your objective function one term at a time.
- Automatic Analytic Derivatives.
- Specialized solvers for bundle adjustment problems in computer vision.

It will too long to make a good introduction to Ceres, but a briefly description of important points, obtained from the tutorial, can be found in the following. The excellent tutorial is available in: http://homes.cs.washington.edu/~sagarwal/ceres-solver/tutorial.html.

Modeling

Ceres solves robustified non-linear least squares problems of the form:

$$\frac{1}{2}\sum_{i=1}\rho_i\left(\left\|f_i\left(x_{i_1},...,x_{i_k}\right)\right\|^2\right). \tag{4}$$

The expression $\rho_i\left(\left\|f_i\left(x_{i_1},...,x_{i_k}\right)\right\|^2\right)$ is known as a `ResidualBlock`, where $f_i(\cdot)$ is a `Cost Function` that depends on the parameter blocks $[x_{i_1},...,x_{i_k}]$. In most optimization problems small groups of scalars occur together. For example, the three components of a translation vector and the four components of the quaternion that define the pose of a camera. Such groups of small scalars are called `ParameterBlock`.

ρ_i is a `LossFunction`. A `LossFunction` is a scalar function that is used to reduce the influence of outliers on the solution of non-linear least squares problems. As a special case, when $\rho_i(x) = x$, i.e., the identity function, it leads to the more familiar non-linear least squares

[3] An *end-effector* is the device at the end of a robotic arm, designed to interact with the environment, in our case the cameras.

problem.

$$\frac{1}{2}\sum_{i=1} \left\| f_i\left(x_{i_1}, ..., x_{i_k}\right)\right\|^2 .$$

Derivatives

Ceres Solver like most optimization packages, depends on being able to evaluate the value and the derivatives of each term in the objective function at arbitrary parameter values. Doing so correctly and efficiently is essential to getting good results. Ceres Solver provides a number of ways of doing so: Analytic and Numeric Derivatives. In order to use Automatic differentiation, it is necessary to define a **templated** cost functor[4].

2.2.4 OpenCV

OpenCV is integrated in ROS and used in many parts of this thesis, in special the function called solvePnP and Rodrigues's formula detailed next.

Rodrigues' formula

Converts a rotation matrix to a rotation vector[5] and vice versa. A rotation vector is a convenient and most compact representation of a rotation matrix (since any rotation matrix has just 3 degrees of freedom).

$$\begin{aligned}
&\theta \leftarrow \mathrm{norm}(r) \\
&r \leftarrow r/\theta \\
&R = \cos\theta I + (1 - \cos\theta) r r^T + \sin\theta \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}
\end{aligned}$$

Figure 13: Rodrigues' formula

SolvePnP

Without loss of generality, it is assumed the model plane (i.e., a checkerboard) is on $Z = 0$ of the world coordinate system. Let's denote the i^{th} column of the rotation matrix R by r_i.

[4] A functor is a class with an `operator()` member. And a cost functor is a functor which evaluate the residuals.

[5] This is an axis-angle representation, the vector module is used as the angle to describe the rotation magnitude about the axis.

From (1),

$$s \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{5}$$

Therefore, a model point **X** and its image **x** are related by a homography; and this can be used for solvePnP.

More precisely, SolvePnP gives a solution to the **PnP problem** –the estimation of the pose of a calibrated camera from n 3D-to-2D point correspondences– minimizing the reprojection error. In others words, it finds the rotation and translation from the checkerboard coordinate system to the camera coordinate system.

Chapter 3

Multi-view recalibration

The relation among the tools previously defined is provided here, explaining the way they work together in both the acquisition and estimation process.

This chapter is divided in three main sections: **acquisition**, **visualization**, and **estimation**. Recall that the goal is to recalibrate multiple cameras, using the PR2 as a real example, and focused on the estimation part.

3.1 Overview

In Figure 14, a high level chart flow is presented: **1.** camera measurements are collected (checkerboard corners, as explained in section 3.2); **2.** the estimation process optimizes a non-linear function in order to find the camera poses; **3.** the result is published to `/tf`.

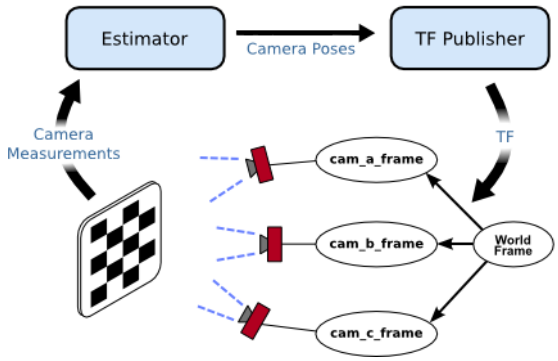


Figure 14: High level flow.

Note: point **3.** is optional. Since visual feedback was desired for the estimation process, it was necessary to consider this option from the beginning in the design. An alternative to **3.** is to

save the result as a new and unique URDF for the particular robot that is being calibrated.

Preliminary: cameras in the PR2

The PR2 has 6 cameras in its head:

- 2 narrow range. Topics: `wide_right_rect` and `wide_left_rect`.
- 2 wide range. Topics: `narrow_right_rect` and `narrow_left_rect`.
- 1 Kinect. Topic: `kinect_head`.
- 1 High definition. Topic: `Prosilica_rect`.

The initial camera configuration is shown in Figure 15, provided by the URDF. **Note**: this is the initialization and also the information to be calibrated.

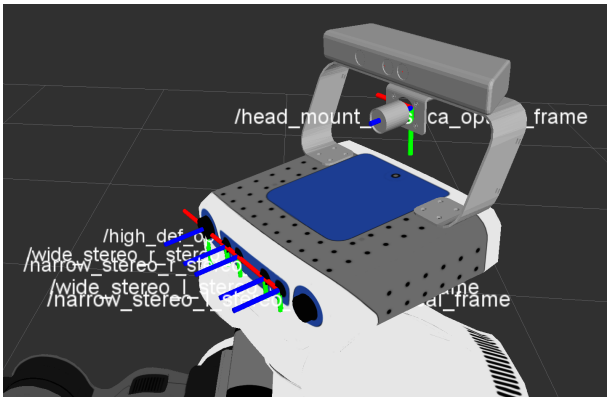


Figure 15: Coordinate system of all the cameras in the PR2 head.

3.2 Acquisition process

This is the *point **1.*** in the overview (Figure 14). The acquisition part was already implemented, and a full description can be found in the paper [11] and in the ROS tutorials for the PR2 calibration package.

3.2.1 Data collection

In order to sufficiently constrain the non-linear optimization (see [11]), it is necessary to collect a large amount of data, and manually collecting this calibration data can be tedious. Data acquisition is automated by having the PR2 hold a checkerboard in its gripper (Figure 16); a total of 30 checkerboard pose measurements are collected for each hand. Additionally, 4 samples

of a **large checkerboard** are captured far from the robot in order to help with far-field calibration (see Figure 17). This data is saved in a ROS bag (section 2.2.1) and later processed by the calibration package.

Notes:

- It is important to mention at this point that 2D measurements are obtained from rectified images (section 2.2.1). Therefore, working with distortion is thankfully avoided in the estimation process.
- It is assumed that all cameras have been intrinsically calibrated in a previous stage.

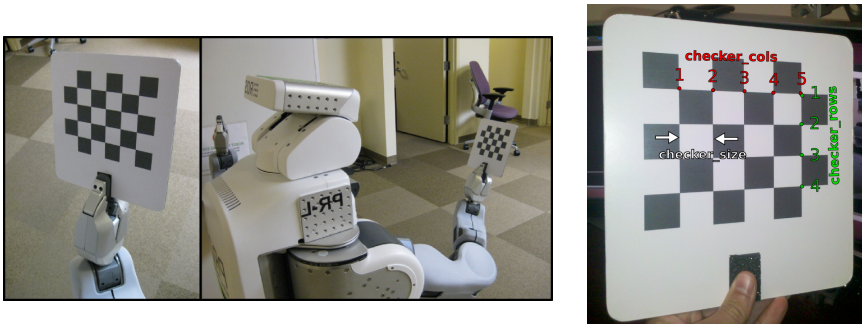


Figure 16: The PR2 holding a checkerboard pattern on the left.

3.3 Visualization

This is the *point **3.*** in the overview (Figure 14). This part is optional, but it was crucial to understand what was happening and to debug problems and solutions.

Even though publishing to /tf is simple, the overall task has been time consuming and more details will be given in chapter 4.

For now, let's us concentrate on 3D points visualization, more precisely RViz Markers.

3.3.1 Visual Markers

Since checkerboards are seen from multiple views, an idea to check if the system is calibrated is to overlap the 3D points obtained from individual cameras using solvePnP, described in section 2.2.4. Two examples can be seen in Figure 18. Different colors represent different cameras.
Note: applying solvePnP to individual cameras to compute 3D points is not the best approach. However, it is used here for a quick comparison of the calibration quality. A superior method is to use the measurements of all the cameras, minimizing an *algebraic error* (linear), which is

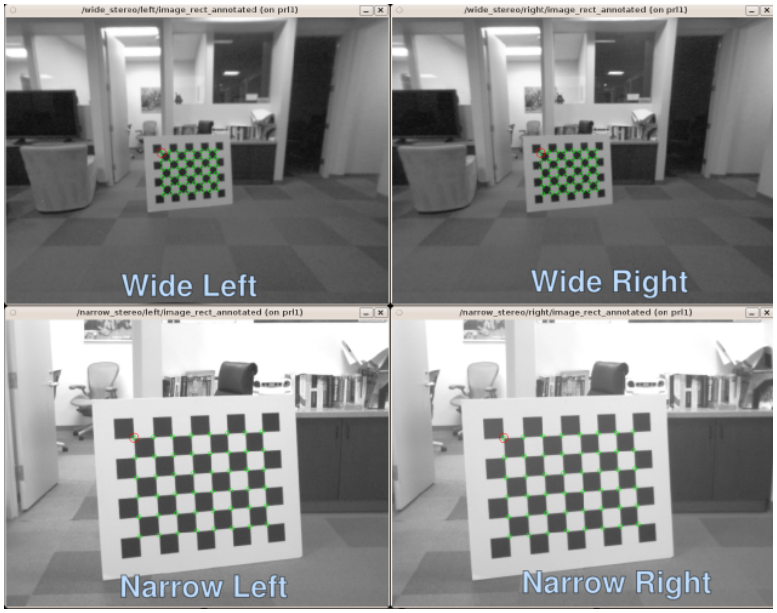
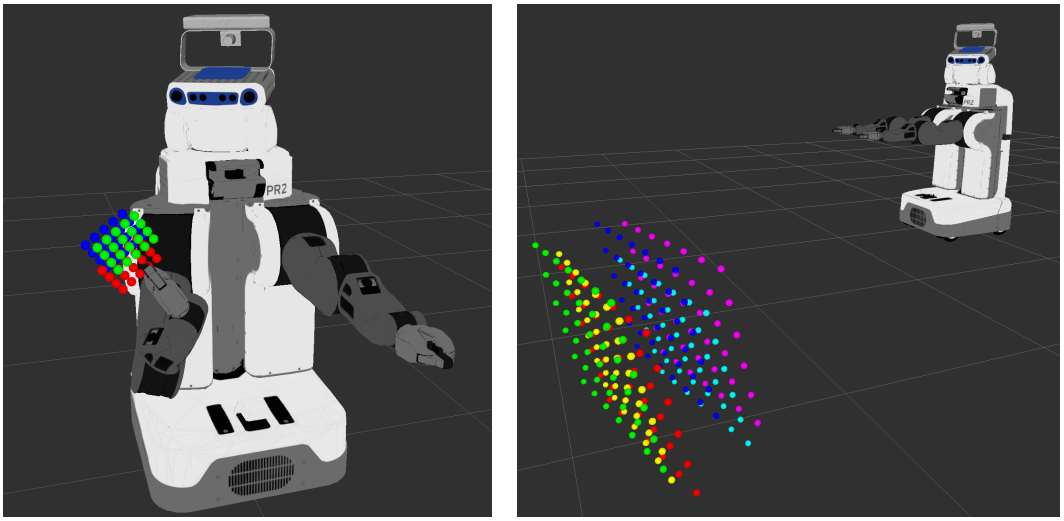


Figure 17: Example of *one* sample from 4 different views, using the large checkerboard.



(a) Checkerboard in hand.

(b) Checkerboard free in the world.

Figure 18: solvePnP results for the all the cameras (same checkerboard).

the n-view triangulation method (section 2.1.3), followed by a *geometrical error* minimization (non-linear) of the reprojection error. It is superior because it is optimal in the **maximum likelihood** sense.

3.4 Estimation process

This is the *point 2.* in the overview (Figure 14). The estimation process is in turn divided in three stages: **initialization**, actual **optimization**, and **updating** the URDF robot model with the results (see Figure 19).

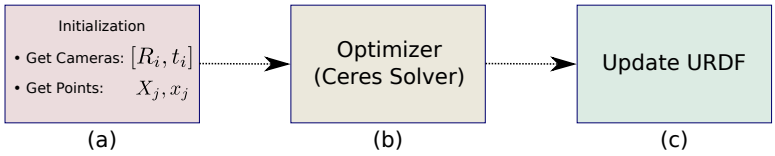


Figure 19: Estimation process.

3.4.1 Initialization

Get cameras

An **initial configuration** of the cameras is given by the URDF, which comes from the robot design.

Since the PR2 head moves during data collection, camera poses change with respect to the robot coordinate system in each calibration step, depending on the joint angles. Then, **forward kinematics** is needed to calculate the poses and a KDL (section 2.2.2) solver has been useful here. **Note**: even though the robot moves, the relative position of the cameras does not change. A rigid relationship exists between them since they are all located in the robot head, which is a rigid entity.

In order to understand Figure 20, some terms are introduced below:

- $\{\mathbf{Robot}\}$: robot coordinate system. It is the root of the URDF tree, usually called the `base_footprint` link.
- $\{\mathbf{C_i}\}$: camera coordinate system i. The *reference camera* is C_0.
- $\{\mathbf{T_i}\}$: transformation from $\{\mathbf{C_i}\}$ to $\{\mathbf{Robot}\}$.
- $\{\mathbf{H_i}\}$: transformation from $\{\mathbf{C_0}\}$ to $\{\mathbf{C_i}\}$. This is the relative position between them, from which rotation and translation matrices are obtained.

Then,

$$H_i = T_i^{-1}\, T_0 \tag{6}$$

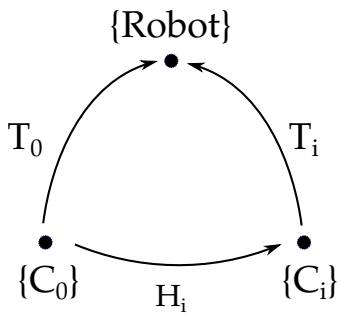


Figure 20: Relative transformations.

rotation R_i and translation t_i of the camera i can be extracted from H_i,

$$H_i = \begin{pmatrix} R_i & t_i \\ 0 & 1 \end{pmatrix} \tag{7}$$

Get points

Now, 3D points must be obtained from the measurements as an initialization. Two initializations are possible:

- using **solvePnP** (section 2.2.4) in the reference camera, 3D points will minimize reprojection error in the first camera. This is not an optimal initialization, as it is explained in the end of section 3.3,
- using **n-view triangulation** (section 2.1.3): by the moment of writing this thesis, this part is considered "*work in progress*". Therefore, results of this initialization are not provided; more details of the problems encountered are explained in the implementation chapter 4. Note: this is a natural extension of the triangulation method, which minimizes an algebraic error. This might be a better initialization for the structure, in comparison with the previous one.

Get projection matrices

As stated in section 2.1.2, BA takes **measurements**, **3D Points** (structure) and **projection matrices** P_i. The last step needed here is to multiply $[R_i \;\; t_i]$ by K_i (camera intrinsic matrix); it is assumed to have this information. Then,

$$P_i = K_i \, [R_i \;\; t_i]$$

3.4.2 Optimizer

A Bundle adjustment (section 2.1.2) implementation is used; details will be discussed in section 4.2.1.

3.4.3 Update URDF

In the robot tree, cameras are in the leaves: an example can be seen in Figure 5(b). The relation between the cameras and their immediate father is the real parameter to optimize. It will be explained soon how to extract this information from the estimation process result. Now, more notation is introduced in Figure 21:

- $\{\mathbf{f_i}\}$: immediate father of camera i.
- $\{\mathbf{T_{f_i}}\}$: transformation from $\{\mathbf{f_i}\}$ to $\{\mathbf{Robot}\}$.
- $\{\mathbf{T_{Rel_i}}\}$: transformation from $\{\mathbf{C_i}\}$ to $\{\mathbf{f_i}\}$.

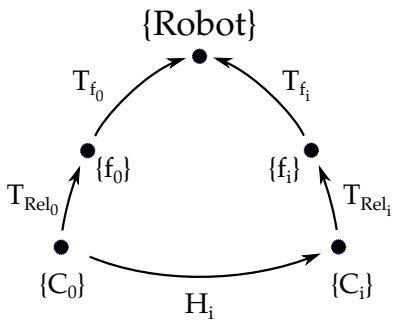


Figure 21: Introducing *camera father* notation.

The transformations defined in the initialization section 3.4.1 can be rewritten as

$$T_i = T_{f_i}\, T_{Rel_i} \tag{8}$$

$$H_i = T_{Rel_i}^{-1}\, T_{f_i}^{-1}\, T_0 \tag{9}$$

The ***update process*** is illustrated in Figure 22. A new H_i' is obtained from the optimization process, and new $\mathbf{T'_{Rel_i}}$ can be calculated as follows:

$$H_i' = T_{Rel_i}^{'-1}\, T_{f_i}^{-1}\, T_0 \quad \Rightarrow \quad \mathbf{T'_{Rel_i}} = T_{f_i}^{-1}\, T_0\, H_i^{'-1} \tag{10}$$

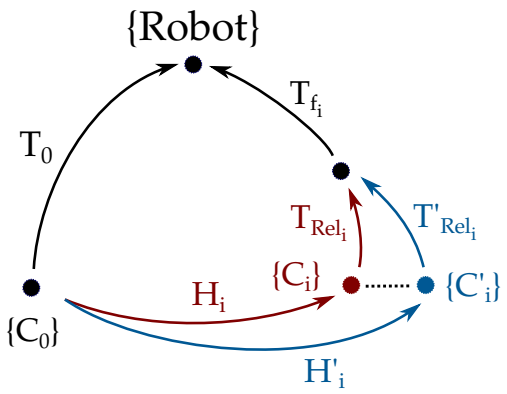


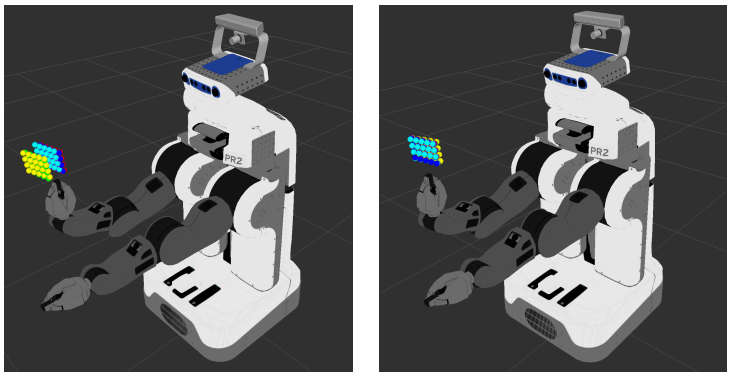
Figure 22: Old calibration (red) vs new calibration (blue). Dashed arrow is the *calibration error*.

3.4.4 Results

In this section, a result comparison is given. First, the results before and after the optimization process are compared visually using solvePnP, as described in section 3.3.1. Second, a quantitative comparison of the optimization process output is provided.

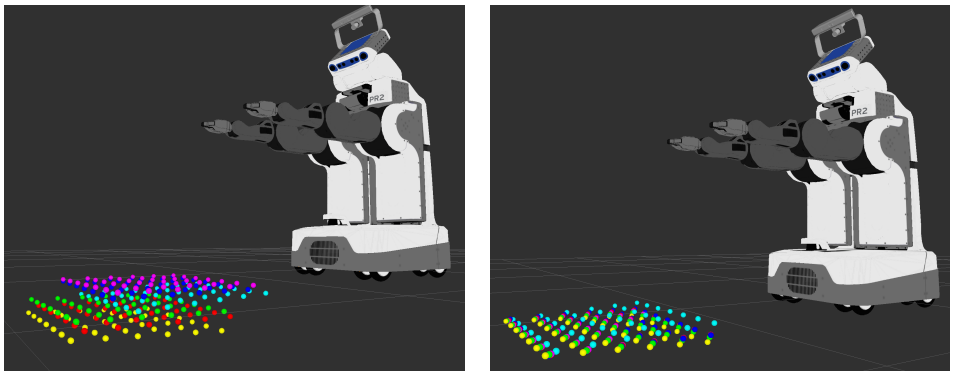
Visual comparison

Only two of a total of 64 checkerboard samples are shown here: a checkerboard in hand in Figure 23, and a checkerboard free in the world in Figure 24. It is easy to see that after the optimization process the checkerboards are closer to each other, indicating that a better calibration has been obtained. The next section gives a more exhaustive comparison of the estimation results.



(a) Before estimation. (b) After estimation.

Figure 23: Checkerboard in hand.



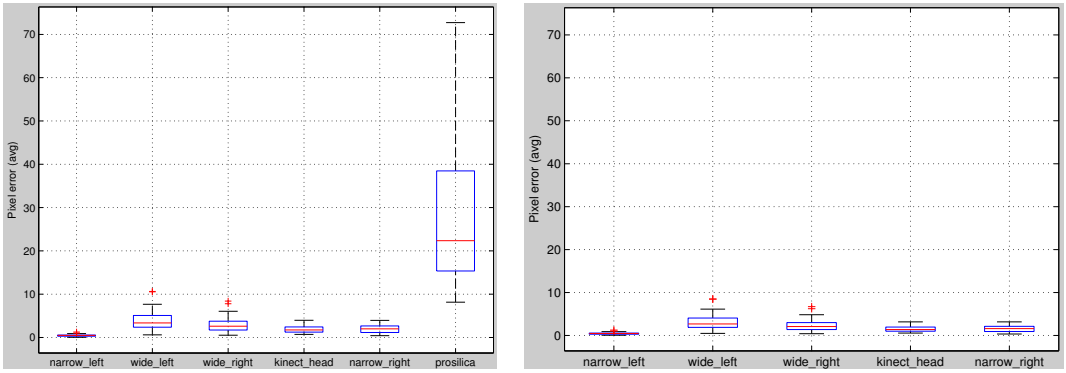
(a) Before estimation. (b) After estimation.

Figure 24: Checkerboard free in the world.

Note: one of the cameras does not show a good result (specially in Figure 24), corresponding to the `prosilica` camera; a likely factor of this problem will be mentioned in the next section.

Quantitative results

As mentioned above, the only finished implementation is the one which uses solvePnP to find 3D points in the first camera (reference camera). These points are used as initialization for the structure. It is not an optimal solution and exhibits a bias on the reference camera (`narrow_left_rect`): this camera has an excellent reprojection error for the first camera in comparison with the rest, as can be seen in Figure 25(a).



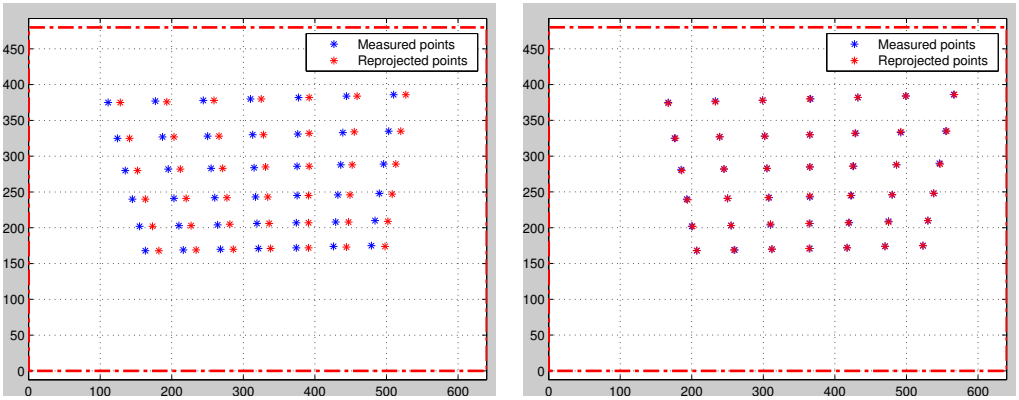
(a) With `prosilica` camera (b) Without `prosilica` camera

Figure 25: Visible bias on first camera (with and without `prosilica` camera)

In addition, it is possible to observe that the `prosilica` camera has a larger error than the

remaining cameras. The reason could be a wrong intrinsic calibration. In order to determine if the bias was produced for this camera, the same comparison has been repeated without taking into account the `prosilica` camera in the estimation process (Figure 25(b)). Even though the global reprojection improves in this case, the bias continues on the first camera.

A visible comparison of the bias can been noted in Figure 26, where reprojected points after optimization are plotted for two cameras: the `wide_left_rect` and the `narrow_left_rect` camera.



(a) `wide_left_rect` camera (b) `narrow_left_rect` camera (reference cam.)

Figure 26: Reprojection error after optimization.

More statistics (median, mean and standard deviation) can be observed in Figure 27.

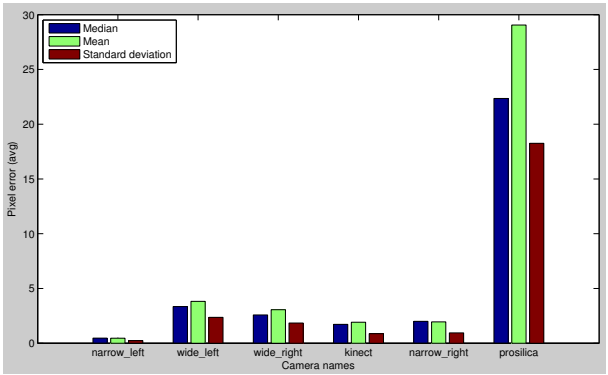


Figure 27: Median, mean, std after optimization.

Speed was not a priority of this project, but it is worth it to mention that the Ceres Solver takes only a few seconds to compute the result and less than 50 iterations to converge.

Chapter 4

Implementation details

Software engineering decisions, problems/solutions and resignations will be discussed in this chapter.

4.1 Software engineering decisions

4.1.1 Why to re-implement the calibration estimation package?

Willow Garage has an already implemented calibration package:

http://www.ros.org/wiki/calibration

It is a more generic package, it is able to calibrate joint angles and different type of sensors all together. My code is a github fork of this repository:

https://github.com/pablospe/calibration

Then, why to re-implement it? A number of points:

- Difficult to support new robot models.
- Poor performance.
- Implemented in Python. Rewrite in C++ was required in order to use Ceres Solver, KDL and URDF C++ parser.
- The old calibration package uses DH parameters[6], which has some problems with certain robots.

Mainly for these points, the estimation was decided to be rewritten from scratch.

[6]The Denavit-Hartenberg parameters (DH parameters) are the four parameters associated with a particular convention for attaching reference frames to the links of a spatial kinematic chain, or robot manipulator.

4.1.2 Why KDL?

As mentioned above, the old package uses DH parameters and it has its own kinematic library, instead of using the well tested KDL. Besides forward kinematic, KDL was useful for geometric frame transformations and conversion between quaternions and angle-axis (rotation matrix parameterization).

4.1.3 Why to re-implement `robot_state_publisher`?

`robot_state_publisher` is a package in ROS which publishes the state of a robot to `/tf`. It takes the joint angles of the robot as input and publishes the 3D poses of the robot links, using a kinematic tree model of the robot (KDL tree which is obtained from the URDF description).

There are two problems with this package:

- It does not allow to modify the URDF without constructing a new `robot_state_publisher` from the tree model. Note the importance of this because the main goal of calibration is to update the URDF. In addition, since it is an optimization process and it must be as fast as possible, it was not desired to send/receive messages during this stage.
- Another problem is that the fixed joints (non-mobile joints) are published half a second later (for some unknown reason), and when the URDF is updated strange behavior might occur.

4.1.4 Quaternions

One important observation is the use of quaternions. This decision was taken to avoid singularities that others rotation parameterization may have.

4.2 Code

4.2.1 Cost function

The Ceres Solver cost function implemented is simple, mainly because distortion is not needed to be considered, since the images are already rectified.

```
template <typename T>
void calc_residuals(double observed_x, double observed_y,
                    double fx, double fy, double cx, double cy,
                    const T* const camera_rotation,
                    const T* const camera_translation,
                    const T* const point,
                    T residuals[2])
{
  // camera_rotation are the quaternions
  T p[3];
  ceres::QuaternionRotatePoint(camera_rotation, point, p);

  // camera_translation is the translation
  p[0] += camera_translation[0];
  p[1] += camera_translation[1];
  p[2] += camera_translation[2];

  // Compute the projection
  T xp = p[0] / p[2];
  T yp = p[1] / p[2];
  T predicted_x = T(fx) * xp + T(cx);
  T predicted_y = T(fy) * yp + T(cy);

  // The error is the difference between the predicted and
     observed position.
  residuals[0] = predicted_x - T(observed_x);
  residuals[1] = predicted_y - T(observed_y);
}
```

Listing 4.1: Cost function

4.2.2 N-View Triangulation

This is a generalization of the triangulation method proposed in section 2.1.3.

```
// It is the standard DLT (for multiple views)
void nViewTriangulate(const Mat_<double> &x,
                      const vector<Matx34d> &Ps,
                      Vec3d &X)
{
  unsigned nviews = x.cols;

  CV_Assert(x.rows == 2);
  CV_Assert(nviews == Ps.size());

  Mat_<double> design = Mat_<double>::zeros(2 * nviews, 4);
  for (int i = 0; i < nviews; ++i) {
    for (int j = 0; j < 4; ++j) {
      design(i*2,   j) = x(0,i) * Ps[i](2, j) - Ps[i](0, j);
      design(i*2+1, j) = x(1,i) * Ps[i](2, j) - Ps[i](1, j);
    }
  }

  Mat X_homog;
  cv::SVD::solveZ(design, X_homog);
  homogeneousToEuclidean(X_homog, X);
}
```

Listing 4.2: Triangulation

4.3 Problems and solutions

4.3.1 Visible cameras

Problem: depending of the point of view, not all the checkerboards are visible by all the cameras (see Figure 28). Moreover, the order of the cameras in the ROS messages can change. This complicated the whole process adding unexpected complexity to simple tasks. It could have been avoided with a better design in the acquisition part.

Solution: A class called `View` was created which works as a data container with a well defined

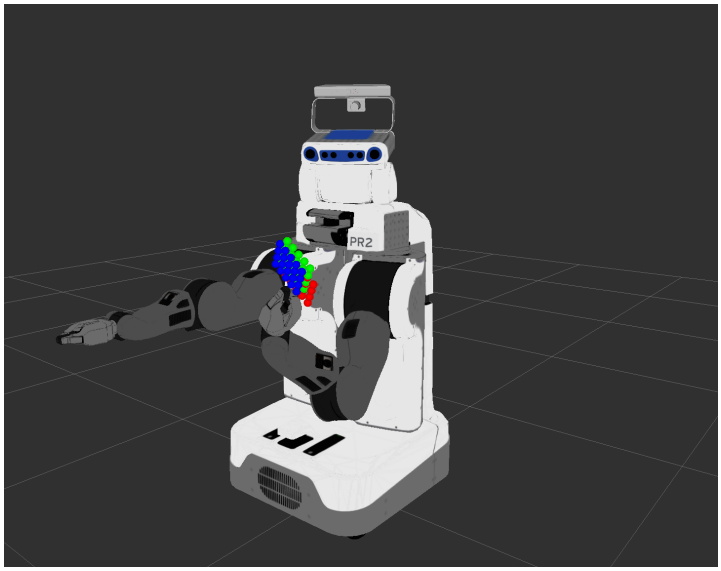


Figure 28: Checkerboard in hand visible only for 3 cameras.

camera order. It has some "mapping" between cameras and frame names (coordinate system names).

4.3.2 Visual Markers

Problem: if the number of checkerboard is reduced (less cameras can see the checkerboard), old markers will remain in the scene.
Solution: erasing all markers and publishing periodically with a timer.

4.3.3 Robot update

Problem: once the optimization process finishes, the result is not ready to be use. The rotation and translation matrices are relative to the first camera, and cannot be used to update the robot. A example in Figure 29, which was one of the initial "solutions".
Solution: an explanation of how to update the robot after the optimization process is given in section 3.4.3.

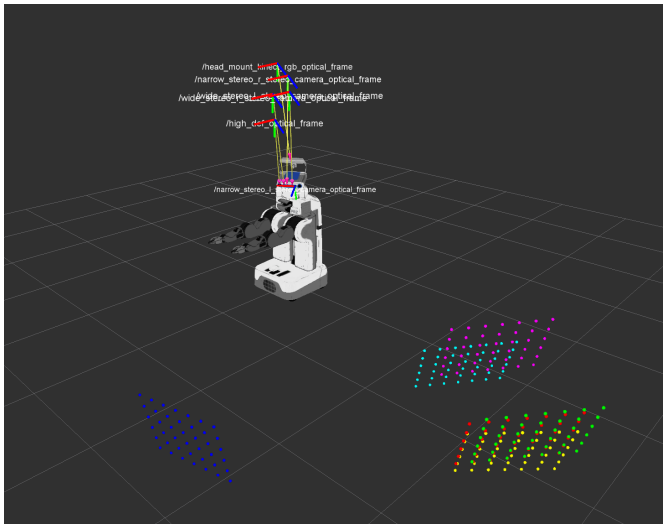


Figure 29: Initial versions

4.3.4 Multi-view triangulation

Problem: multi-view triangulation method is not as precise as expected, see Figure 30.
Solution: more analysis is required to find why the reprojection error is large.

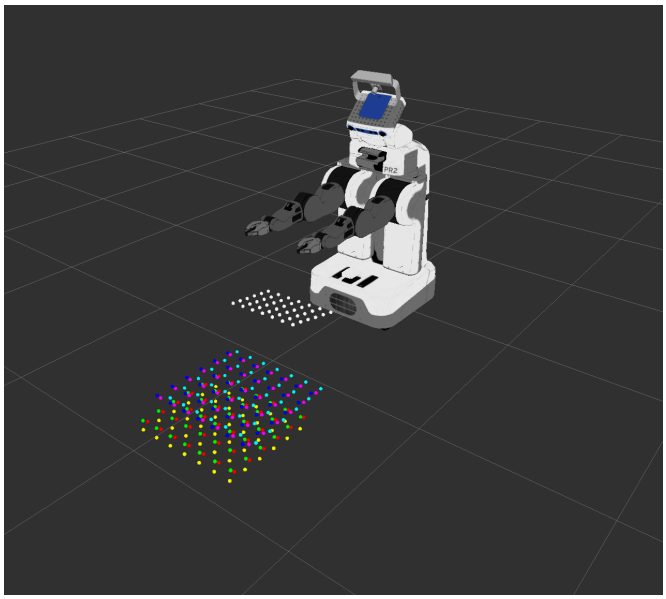


Figure 30: Triangulation points (white dots) must be closer to solvePnP solutions (color dots)

4.4 Stats

Just for fun, it took me 9,061 additions and 3,954 deletions in 99 commits until this point.

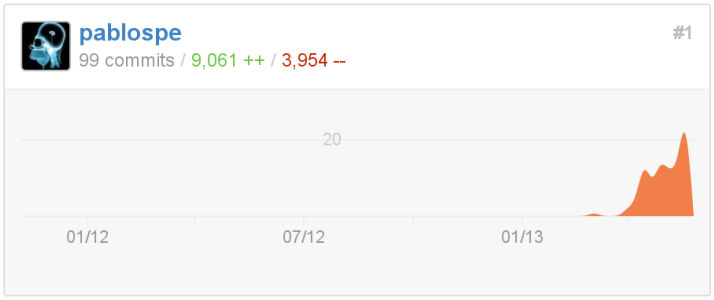


Figure 31: Github stats

Github repository: https://github.com/pablospe/calibration.

Chapter 5

Additional work

Two methods for structure initialization has been proposed (in chapter 3). However, a prior initialization for motion –rotation and translation– is required. This chapter deals with a motion initialization, based on measurements and on the beforehand initialization.

This chapter is purely theoretical and experiments must be performed in order to get validation. Nevertheless, it is worth further research. Knowledge of *linear algebra* and *epipolar geometric* will be assumed.

5.1 Preliminary

Some preliminary concepts of epipolar geometric are introduced as a reminder: essential matrix and camera resection.

5.1.1 Essential matrix

Now, consider a pair of camera matrices $P = [I\ 0]$ and $P' = [R\ t]$. The essential matrix has the form (see [13, p257]):

$$E = [t]_\times R \tag{11}$$

where

$$[t]_\times = \begin{pmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{pmatrix}, \qquad \text{if } t = (t_1, t_2, t_3)^T \tag{12}$$

5.1.2 Camera resection

Extraction of cameras from the essential matrix is possible using SVD, more details can be found in [13, p258-260, Result 9.19]. There are four possible solutions, except for overall scale, which cannot be determined. Figure 32 shows that a reconstructed point X will be in front of both cameras in only one of these four solutions. Thus, testing a single point, to determine if it is in front of both cameras, is sufficient to decide between the solutions for the camera matrix P'.

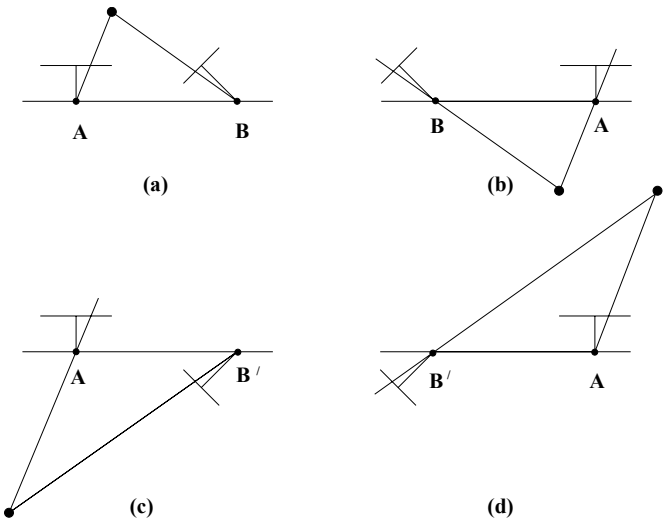


Figure 32: A reconstructed point X is in front of both cameras only in (a).

5.2 Proposed method

In this section, a proposed method is explained. This method is a generalization of camera resection (see section 5.1.2, or [13, Result 9.19]). In [13, p258, section 9.6.2] is stated that the overall scale cannot be determined, but it has been derived here that the scale is the norm of the original translation vector $\mathbf{t}$ (see theorem 1). However, the norm of translation vector $\mathbf{t}$ is usually unknown (recall that we want to recover the motion, $\mathbf{R}$ and $\mathbf{t}$, from $\mathbf{E}$), but for theoretical purposes let's suppose for a moment it is known. **Note**: part of the following theorem proof is postponed until Appendix A.

5.2.1 Demonstration

Theorem 1. *Suppose a given essential matrix $E = [\mathbf{t}]_\times R$, and the norm of $\mathbf{t}$ is known.*

$$\left.\begin{array}{l} E = [t]_\times R = S\,R \\ ||t|| = k \end{array}\right\} \Rightarrow \begin{array}{ll} E = U diag(k,k,0) V^T & \text{a SVD decomposition} \\ t = \;\; k\,u_3, & u_3 \text{ last column of } U \\ or\; t = -k\,u_3 & \end{array}$$

Proof.

$$\text{(from lemma 2, Appendix A)} \quad \Rightarrow \quad E = U diag(k,k,0) V^T \qquad \text{a SVD decomposition}$$

$$\begin{aligned} t\,[t]_\times = 0 \quad &\Rightarrow \quad t\,S = 0 \\ &\Rightarrow \quad t \text{ is left null-space of } S \\ &\Rightarrow \quad t \text{ is left null-space of } E, \quad \text{since } E = SR \\ &\Rightarrow \quad t = \alpha\,u_3, \qquad \alpha \in \mathbb{R} \qquad (13) \\ &\qquad\qquad\;\; u_3 \text{ last column of } U \\ &\qquad\qquad\;\; ||u_3|| = 1 \qquad (14) \end{aligned}$$

We know that

$$\begin{aligned} ||t|| = k \overset{(13)}{=} ||\alpha\,u_3|| = |\alpha|\,||u_3|| \quad &\overset{(14)}{\Rightarrow} \quad |\alpha| = k \\ &\Rightarrow \quad \alpha = k \text{ or } \alpha = -k \end{aligned}$$

then

$$\begin{aligned} t = \;\; ||t||\,u_3 \text{ or} \\ t = -||t||\,u_3 \end{aligned}$$

□

5.2.2 Resection generalization

The actual generalization is presented in the following theorem:

Theorem 2. *For a given essential matrix $E = U\,diag(k,k,0)\,V^T$, where $||t|| = k$, and first camera matrix $P = [I\;\;0]$, there are four possible choices for the second camera matrix P',*

namely

$$\begin{aligned} P' &= [\ UWV^T\ |+k\,u3]\ \textit{or} \\ &= [\ UWV^T\ |-k\,u3]\ \textit{or} \\ &= [\ UW^TV^T\ |+k\,u3]\ \textit{or} \\ &= [\ UW^TV^T\ |-k\,u3]. \end{aligned}$$

where

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{15}$$

Proof. The proof is similar to [13, Result 9.18 and Result 9.19], the exception is the part where the translation is calculated, theorem 1 should be used. The full demonstration is left as an exercise for the reader[7]. **Notes**: **1.** four solutions are possible since there are two options for the rotation matrix and two for the translation: $t = ||t||\,u_3$ or $t = -||t||\,u_3$; **2.** as before, it is possible to determine the correct solution by testing with a single reconstructed point X. □

5.2.3 Scenario

Consider the following hypothetical scenario: **1.** only stereo pair will be taken into consideration instead of a multi-cameras setup; **2.** all cameras have the same K intrinsic matrix, the identity: $K = I$; **3.** a **small** perturbation is the motion is produced by external causes, provoking a loss in the calibration. Figure 33 illustrates these assumptions:

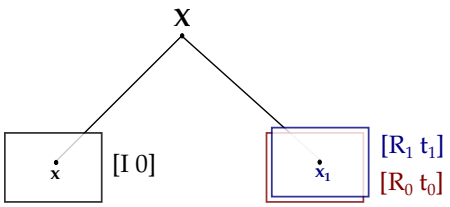


Figure 33: Stereo pair with and without perturbation.

where

- $\mathbf{R_0}$, $\mathbf{t_0}$: previous known motion (prior calibration or initialization).
- $\mathbf{R_1}$, $\mathbf{t_1}$: new motion (unknown, but measurements in $[R_1\ t_1]$ coordinate system.).

[7] Please consider this as a revenge.

5.2.4 Application

From the assumption number **3.** (small perturbations),

$$||t_0|| \approx ||t_1|| \tag{16}$$

Now, suppose that resection has been applied to the essential matrix E obtained from the measurements between $[I\ 0]$ (left camera) and $[R_1\ t_1]$. Then, it is possible to obtain

$$E \rightarrow R^*, t^* \tag{17}$$

For the exposed in [13] and theorem 2, we can recover the motion

$$\begin{cases} R_1 = R^* \\ t_1 = ||t_1||\, t^* \end{cases} \tag{18}$$

But $||t_1||$ is unknown; then, using the approximation (16),

$$\begin{cases} R_1 = R^* \\ t_1 \approx ||t_0||\, t^* \end{cases} \tag{19}$$

And finally, the **proposed method** can be found in the last equation (19). In word, can be summarized as: **1.** find essential matrix using the measurements; **2.** apply camera resection to recover motion, but with unknown scale factor in the translation; **3.** use the previous translation norm (our prior knowledge) to approximate the scale factor using theorem 2 and the assumption of small perturbations.

5.2.5 Advantages / disadvantages

Advantages

- Rotation matrix is can be calculated any ambiguity.
- Robust estimation for essential matrix exist; therefore, it can be robust to outliers.

Disadvantages

- It is needed to calculated the essential matrix and camera resection. It is more expensive computationally, in contrast with the use of the prior knowledge directly (i.e., previous motion: $[R_0\ t_0]$).
- It has not been yet validate, see "open questions" below.

5.2.6 Open questions

As mentioned before, all the experimentation have been focused in multi camera calibration. Therefore, there are still open questions, e.g.: 1. will it be possible to recover the scale factor of $||t_1||$ with this method? 2. In that case, will it be better this initialization that the one already provided (i.e., previous motion)? Further research and experiments are needed to validate the proposed method.

Chapter 6

Future work

Since the ROS package describes in this thesis is still in development stage many things need to be improved, some points are listed here:

- The highest priority is to fix triangulation in order to get a better structure initialization, this will lead to a considerable improvement in the whole calibration system.
- User interface is required to make the package more useful.
- The system should be robot agnostic. There is a wizard in the new Moveit! package that can be used it to create the configuration files.
- Export the URDF to a file. I had to fixed some bugs in the URDF Dom package, they will allow to create this functionality.
- Validate the proposed method in chapter 5 for motion initialization.

Chapter 7

Conclusions

This project combines the strongest of several **open-source** components: KDL, Ceres Solves, ROS, OpenCV and PR2 (open-source platform, see Figure 34). It cannot be less to contribute in the same way to the community, making the code freely available as a ROS package (even though it is still in development stage).

Figure 34: PR2 pleased to be free.

The above mentioned software components are well designed, with excellent interfaces and documentation. Even though this fact, **combining** them in an unique product is a **time consuming** task, which requires different type of abilities; for example, solve compilation problems with CMake or Catkin (new compilation system in ROS), to mention one.

Regarding the thesis itself, **robot calibration** is an important problem to be solved in any robot, in order to allow an effective interaction with the environment. And this thesis has been focused in **multiple camera calibration** mounted in a robot. The work has been done by experimenting with real data, real robots, real problems.

Two different methods have been proposed for the **structure initialization** (3D points). Initialization used for the bundle adjustment optimizer (Ceres Solver). These methods are: solvePnP –3D points in the camera reference– and n-view triangulation method.

The improvement in calibration has been shown **visually** –for a human-friendly comparison– and **quantitatively** –for a more rigorous analysis–. A bias has been observed in the solvePnP solution, which will lead to future research with the aim to improve results using n-view triangulation method.

In addition to the practical experimentation, a theoretical development has been also elaborated in the framework of epipolar geometry, in which a method for motion initialization is discussed.

To conclude with this thesis, it has been a **proud** to work with two excellent person like David Fofi and Vincent Rabaud, in two excellent places like Le2i and Willow Garage.

Appendix A

Lemma demonstrations

This Appendix has auxiliary lemmas used in the theorem (1) proof. Knowledge of linear algebra will be assumed.

Lemma 1.

$$\left.\begin{array}{l} A = Q\,\Lambda\,Q^T \\ \Lambda \textit{ diagonal matrix} \\ Q \textit{ symmetric matrix} \end{array}\right\} \Rightarrow A \textit{ and } \Lambda \textit{ have same eigenvalues.}$$

Proof. See [15] for a demonstration. □

Lemma 2.

$$\left.\begin{array}{l} E = [t]_\times R \\ ||t|| = k \end{array}\right\} \Rightarrow E \textit{ has two singular values equal to } k \textit{ and the last one is zero.}$$

Proof. From [13, Result A4.1 (ii)],

$$\begin{array}{lcll} S = [t]_\times \text{ is skew-symmetric} & \Rightarrow & S = \alpha\,U\,Z\,U^T, & \text{with } U \text{ orthogonal} \\ & \Rightarrow & S = U\,Z_\alpha\,U^T & \end{array}$$

where

$$Z = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad Z_\alpha = \begin{pmatrix} 0 & \alpha & 0 \\ -\alpha & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{20}$$

Taking

$$\begin{aligned} S\,S^T &= U\,Z_\alpha\,U^T(U\,Z_\alpha\,U^T)^T \\ &= U\,Z_\alpha\,(U^TU)\,Z_\alpha^T\,U^T \\ &= U\,Z_\alpha\,Z_\alpha^T\,U^T \end{aligned} \tag{21}$$

From (21),

$$S\,S^T \text{ and } Z_\alpha\,Z_\alpha^T \text{ are similar matrices} \quad \Rightarrow \quad tr(S\,S^T) = tr(Z_\alpha\,Z_\alpha^T) \tag{22}$$

By expansion,

$$S\,S^T = \begin{pmatrix} t_3^2+t_2^2 & 0 & 0 \\ 0 & t_1^2+t_3^2 & 0 \\ 0 & 0 & t_2^2+t_1^2 \end{pmatrix} \tag{23}$$

then,

$$tr(S\,S^T) = (t_1^2+t_2^2+t_3^2)+(t_1^2+t_2^2+t_3^2) = 2\,||t||^2 = 2\,k^2 \tag{24}$$

In the other hand (by expansion),

$$tr(Z_\alpha\,Z_\alpha^T) = tr\begin{pmatrix} \alpha^2 & 0 & 0 \\ 0 & \alpha^2 & 0 \\ 0 & 0 & 0 \end{pmatrix} = 2\,\alpha^2 \tag{25}$$

From (21), (24) and (25),

$$\alpha^2 = k^2 \tag{26}$$

and also

$$Z_\alpha\,Z_\alpha^T = \begin{pmatrix} k^2 & 0 & 0 \\ 0 & k^2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{27}$$

Now, since $Z_\alpha\,Z_\alpha^T$ diagonal, and U is orthogonal (symmetric), it is possible to use lemma (1) then,

$$S\,S^T \text{ has the same eigenvalues than } Z_\alpha\,Z_\alpha^T \tag{28}$$

Now, we start from the hypothesis:

$$E = S\,R \quad \Rightarrow \quad E\,E^T = S\,R\,(S\,R)^T = S\,R\,R^T\,S^T = S\,S^T \tag{29}$$

From (28) and (29),

$$E\,E^T \text{ has the same eigenvalues than } Z_\alpha\,Z_\alpha^T \tag{30}$$

Then,

$$E \text{ has two singular values equal to } k \text{ and the last one is zero.} \tag{31}$$

and,

$$E = U\,diag(k,k,0)\,V^T \qquad \text{a SVD decomposition} \tag{32}$$

□

Bibliography

[1] ROS: Robot Operating System. http://www.ros.org/.

[2] RViz: 3D visualization tool for ROS. http://www.ros.org/wiki/rviz/.

[3] Keenan A. Wyrobek, Eric H. Berger, H. F. Machiel Van der Loos, and J. Kenneth Salisbury. Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot. In *2008 IEEE International Conference on Robotics and Automation*, pages 2165–2170. IEEE, May 2008.

[4] R. Smits. KDL: Kinematics and Dynamics Library. http://www.orocos.org/kdl.

[5] Sameer Agarwal and Keir Mierle. *Ceres Solver: Tutorial & Reference*. Google Inc.

[6] Qilong Zhang. Extrinsic calibration of a camera and laser range finder. In *In IEEE International Conference on Intelligent Robots and Systems (IROS*, page 2004, 2004.

[7] Ranjith Unnikrishnan and Martial Hebert. Fast extrinsic calibration of a laser rangefinder to a camera. Technical report.

[8] Radu Horaud and Fadi Dornaika. Hand-eye calibration.

[9] C.T. Huang and O.R. Mitchell. Dynamic camera calibration. In *Computer Vision, 1995. Proceedings., International Symposium on*, pages 169–174, 1995.

[10] Y. Furukawa and J. Ponce. Accurate camera calibration from multi-view stereo and bundle adjustment. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, 2008.

[11] Vijay Pradeep, Kurt Konolige, and Eric Berger. Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach. In *International Symposium on Experimental Robotics (ISER)*, New Delhi, India, 12/2010 2010.

[12] Bill Triggs, P. McLauchlan, Richard Hartley, and A. Fitzgibbon. Bundle adjustment – a modern synthesis. In B. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372. Springer-Verlag, 2000.

[13] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[14] PR2 calibration package. http://www.ros.org/wiki/pr2_calibration.

[15] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 1993.