Bachelor thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Two Arm Robot Manipulation with Garments

Jan Vítek
Cybernetics and Robotics

May 22, 2014
Supervisor: Ing. Vladimír Smutný

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Jan V í t e k

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Manipulace s oděvy dvourukým robotem

Pokyny pro vypracování:

1. Seznamte se s operačním systémem ROS, s robotem Clopema a s existujícími přístupy k manipulaci s oděvy.
2. Seznamte se s nástroji, které již byly vytvořeny v projektu Clopema.
3. Navrhněte a implementujte nástroje pro manipulaci s oděvem pro projekt Clopema.
4. S vytvořenými nástroji proveďte experimenty a vyhodnoťte je.

Seznam odborné literatury:

[1] Stephen Miller, Jur van den Berg, Mario Fritz, Trevor Darrell, Ken Goldberg, Pieter Abbeel: A Geometric Approach to Robotic Laundry Folding. In the International Journal of Robotics Research (IJRR), first published on December 20, 2011.
[2] Ping Chuan Wang, Stephen Miller, Mario Fritz, Trevor Darrell, Pieter Abbeel: Perception for the Manipulation of Socks, In the proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011.
[3] https://portal.fnpconnect.com/documents/10113/4412513/Garment+Folding+Methods
[4] Li Sun, Gerarado Aragon-Camarasa, Paul Cockshott, Simon Rogers, J. Paul Siebert: A Heuristic-Based Approach for Flattening Wrinkled Clothes (to be published).

Vedoucí bakalářské práce: Ing. Vladimír Smutný

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Jan V í t e k

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Bachelor Project: Two Arm Robot Manipulation with Garments

Guidelines:

1. Get familiar with the ROS operating system, Clopema robot, existing approaches to the garment manipulation.
2. Get familiar with already implemented tools within the Clopema project.
3. Design and implement the tools for garment manipulation within Clopema project.
4. Make experiments and evaluate results.

Bibliography/Sources:

[1] Stephen Miller, Jur van den Berg, Mario Fritz, Trevor Darrell, Ken Goldberg, Pieter Abbeel: A Geometric Approach to Robotic Laundry Folding. In the International Journal of Robotics Research (IJRR), first published on December 20, 2011.
[2] Ping Chuan Wang, Stephen Miller, Mario Fritz, Trevor Darrell, Pieter Abbeel: Perception for the Manipulation of Socks, In the proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011.
[3] https://portal.fnpconnect.com/documents/10113/4412513/Garment+Folding+Methods
[4] Li Sun, Gerarado Aragon-Camarasa, Paul Cockshott, Simon Rogers, J. Paul Siebert: A Heuristic-Based Approach for Flattening Wrinkled Clothes (to be published).

Bachelor Project Supervisor: Ing. Vladimír Smutný

Valid until: the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2014

Acknowledgement / Declaration

I would like to thank my thesis supervisor Ing. Vladimír Smutný for the guidance of my work and his advices about the computer vision and 3D stereo image reconstruction procedures. A special thanks goes to Ing. Libor Wagner for the priceless answers on my questions about CloPeMa and ROS and assistance with implementation of the outcome of my thesis to the CloPeMa systems.
The thesis could not have been created without the support of my girlfriend Jana, my family and their understanding for spending my nights with MATLAB and OpenNI.

I hereby declare that this thesis is my own work and I have quoted each source I have used in compliance with "Metodický pokyn o dodržování etických principů při přípravě vysokoškolských závěrečných prací".
In Prague, May 22, 2014

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.
V Praze dne 22. května 2014

.....

Abstrakt / Abstract

Tato bakalářská práce se zabývá kalibrací 3D senzorů založených na technologii vyvinuté firmou PrimeSense. Ke kalibraci je použit postup, který byl již dříve navrhnut a popsán Ing. Janem Smíškem, pro kalibraci senzoru Kinect od firmy Microsoft. Tento postup byl v rámci práce upraven pro použití se senzory ASUS Xtion a PrimeSense Carmine. Pro aplikaci kalibračních dat bylo nutné upravit zdrojový kód open-source ovladače OpenNI a vytvořit balíčky pro linuxové distribuce a platformy používané v rámci projektu CloPeMa. Výsledná kalibrace senzorů byla ověřena jak teoreticky, tak prakticky.

Klíčová slova: ASUS; Xtion; PrimeSense; Carmine; OpenNI; kalibrace; počítačové vidění; 3D senzor; CloPeMa; ROS

Překlad titulu: Manipulace s oděvy dvourukým robotem

The thesis deals with calibration of 3D sensors using depth measuring technology developed and patented by PrimeSense. The calibration is based on Microsoft Kinect calibration procedure earlier developped by Ing. Jan Smíšek and slightly modified to work with ASUS Xtion and PrimeSense Carmine. It was necessary to modify the OpenNI drivers to accept the calibration data and build packages for each Linux distribution and platform used in the CloPeMa project. The resulting calibration of sensors attached to the CloPeMa robot was verified both theoretically and in practice.

Keywords: ASUS; Xtion; PrimeSense; Carmine; OpenNI; calibration; computer vision; 3D sensor; CloPeMa; ROS; structured light range finder

Contents

Tables / Figures

Chapter 1
Introduction

For fine manipulation it is good to equip a robot with some kind of vision. This requirement gets more important if the object of manipulation changes its shape and its initial position is not accurately defined. Then you need to find and identify the manipulated object prior to the manipulation. In project CloPeMa 3D sensors made by PrimeSense and ASUS are used for localization of a garment on a table or to find a grasping point on the garment. Later, when the garment is grasped, the success depends on the accuracy of the position measurement. My task is to improve accuracy of the sensors and to make the garments manipulation more predictable and reliable.

To achieve the best results I will use a calibration procedure developed by Jan Smíšek and described in [12, 8]. To calibrate RGB-D sensor (color camera and range finder) the procedure takes a set of RGB and IR camera images and disparity map of the scene. Jan Smíšek used `libfreenect` driver which does not support our sensors therefore I need to discover whether our drivers are configurable as we need, otherwise modify them for our needs.

Figure 1.1. Depth sensor PrimeSense Carmine 1.09

1.1 Devices

CloPeMa robot uses two types of 3D sensors branded by ASUS and PrimeSense. The ASUS Xtion Pro Live sensor is made to be used for longer distance measurements — returns depth values from 46.6 cm — while the PrimeSense Carmine 1.09 sensor returns depth values starting at 25.8 cm. Both of them use technology developed by PrimeSense (also used by Microsoft Kinect[13]) and OpenNI2 drivers. The sensor projects IR pattern on the scene, captures IR image and triangulates the 3D data. [6] A detailed description how do the sensors work is in [12, 15].

The disparity (inverse depth) data are computed in the device using PrimeSense PS1080 SoC. These data are related to the IR image and if we want to colorize the scene we need to assign RGB picture pixels to the proper IR pixel. This feature is also implemented in the PS1080 SoC and it is called the hardware registration. It means that the sensor returns pre-processed data and I am not able to modify the

disparity calculation if it is inaccurate. On the other hand, the hardware registration could be turned of, and replaced by ROS `openni2_camera` software registration, if it was necessary.

1.2 Device Drivers

OpenNI are open-source drivers developed with participation of PrimeSense, currently in version 2.2.0.33. It is the officially recommended API for Xtion sensors by ASUS[4]. The drivers are discussed in chapter 3, where I present the basic structure and function of the drivers, and my modifications of the code.

1.3 Robot Operating System (ROS)

The CloPeMa robot and its peripherals are driven by the Robot Operating System. It solves basic robotic tasks like inverse and forward kinematics, path planning, collision avoidance etc. The whole system is not driven centrally by one piece of software but it is divided to many subsystems called ROS Nodes.[2] These nodes do its own job (i.e. compute trajectory, distribute camera images) and communicate with each other using publish-subscribe pattern managed by the ROS Master. [7, 1]

The `openni2_camera` package provides us with nodes which publish information about the sensors itself, raw sensor data and processed data — depth registered images — set of points with their coordinates in the space and its color assigned from RGB camera.

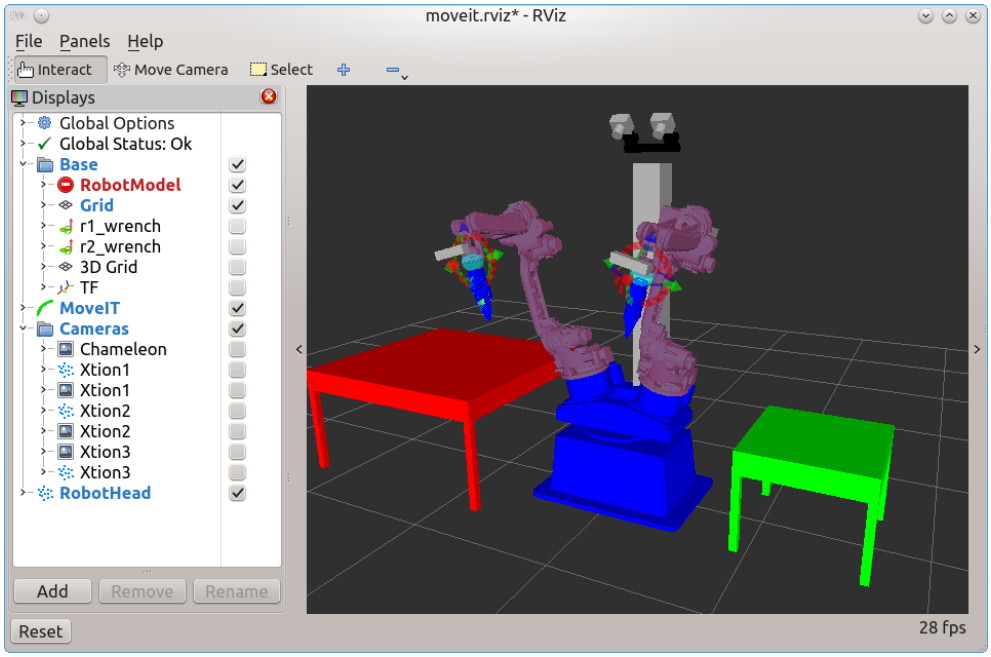


Figure 1.2. View of the CloPeMa robot model in RViz — the ROS visualization tool. RViz lets you manipulate with the robot and visualize RGB-D data captured by the depth sensors that are mounted on the robot's arms.

Chapter 2
Calibration Description

In this chapter I will summarize chapter 4.1 of [12]. I will introduce you the facts that are necessary to be understood to be able to perform the calibration. The calibration uses Jan Smíšek's algorithm which I have slightly modified so that the usage is more comfortable.

2.1 Camera calibration

For the calibration it is necessary to find intrinsic (focal length, distortion, principal point and pixel size) and extrinsic parameters (respective position of IR and RGB camera). Pixel size is given by the resolution and the size of the CMOS[15] and other parameters are approximated using images of the calibration chessboard.

For the actual run of the 3D measurement only intrinsic parameters are passed to the driver. However I have found this as a very important part of the calibration because default focal lengths are noticeably inaccurate. The default values according to the ROS `openni2_camera` package configuration files are 575 px for both IR and RGB lenses although the RGB camera DoF is visibly wider than the IR camera field of view. This lets the RGB camera see the most of the IR camera DoF and it guarantees that there will be RGB data for the whole depth image. So the smaller focal length of the RGB lens is intentional but it is not reflected in the calculations.

The other thing that is completely omitted is the image distortion. It has been found that the radial distortion can cause displacement up to 10 pixels in the image corners[12]. This results in inaccurate measurements if the object (or a part of it) is close to the edge of the image.

After the camera calibration I have made a series of simple measurements to see whether it has any effect on the scene reconstruction. I have measured the width of 26 cm wide box and the distance to it using the PrimeSense Carmine 1.09 sensor. You can see how did the scene look like in Fig. 2.1. The data are summarized in the Table 2.1 and you can see significant improvement of the accuracy of the short range (∼40 cm) measurement. When the measured object was moved to the greater distance distance (∼1.7 m) the accuracy of the measurement decreased however the effect of the camera calibration is still observable.

distance	width [mm]		depth [mm]		width error [mm]	
[mm]	before	after	before	after	before	after
355	245.3	260.1	361	362	147	0.1
421	244.1	258.5	427	427	159	1.5
488	244.2	259.2	493	493	158	0.8
172	235.4	248.7	1670	1670	246	113

Table 2.1. Simple verification of the camera intrinsic parameter calibration. A box 26 cm wide was measured in different distances to verify whether the camera intrinsic parameter has an effect on the measurements.

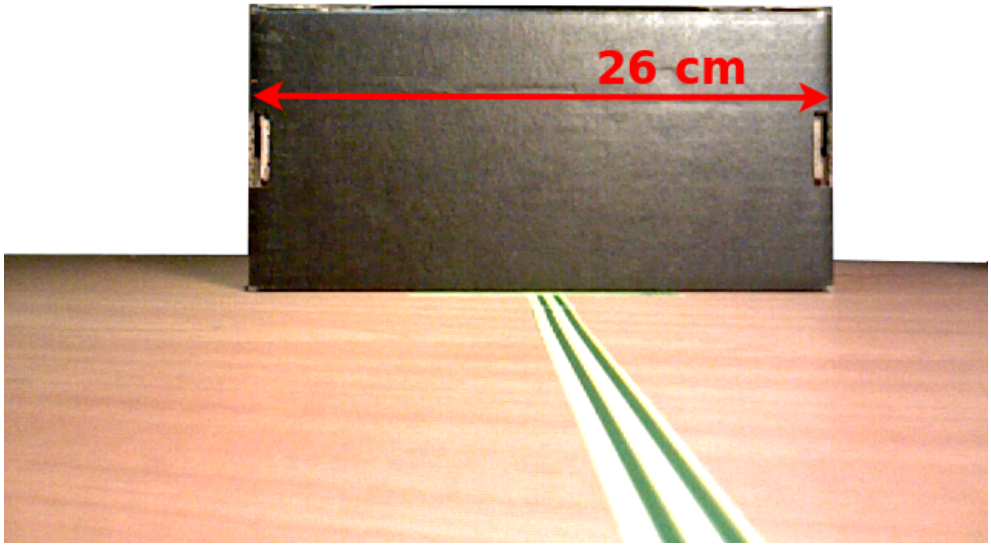


Figure 2.1. A simple camera intrinsic parameters verification scene as seen by the sensor. The 26 cm wide box was measured using the RGB-D sensor to prove an effect of the camera intrinsic parameters calibration.

2.2 Range finder calibration

The structured light 3D sensor measures disparity values which are converted to the depth values in millimetres using equation (1)[12]. This value is later used to calculate `x,y,z` coordinates of the captured points.

$$d = \frac{fb}{c_1 r + c_0} \tag{1}$$

d	...	depth in mm
f	...	focal length in mm
b	...	baseline between IR camera and IR projector (75 mm)
r	...	raw disparity value
c_1, c_0	...	coefficients of the model

The MATLAB range finder calibration algorithm from [12] uses the RGB and IR camera images and disparity map of the scene. The IR and RGB images are used for a stereoscopic reconstruction. The coordinates of chessboard corners are extracted from the reconstructed 3D data and their depth is calculated. The stereoscopic depth values are the ground truth data for the depth calibration. Now the program finds a corresponding disparity value for each chessboard corner. This way we will get one or more disparity values for various depth values. The program uses the least-square method to fit the values to equation (1).

When the data are fitted, we can use equation (1) to calculate a depth in millimetres for each disparity value that sensor can return. The results are stored to a `.csv` file which can be loaded by the OpenNI drivers. Meaning of the file and its structure is described in 3.2.

2.3 Verification

The MATLAB calibration script automatically verifies the results it has calculated. It takes the disparity values on the positions of calibration chessboard corners, calculates the depth using the calibrated model and compares the results to the measurements computed using the stereoscopic scene reconstruction. The output of the verification are two graphs that show the error size according to the depth.

2.4 Output

Here is the list of files which are produced by the MATLAB calibration script. The meaning of the files is discussed in respective chapters and their structure is shown in Appendix B.

1. `depth_PS1080_PrimeSense.yaml`
   The file holds intrinsic parameters of the IR camera. It is used by the ROS `openni2_camera` wrapper and the usage is described in 3.3.2.
2. `rgb_PS1080_PrimeSense.yaml`
   Similar to the previous file with the intrinsic data for the RGB camera.
3. `depth_model.txt`
   There are two numbers which can be substituted for c_0 and c_1 in equation (1).
4. `S2D_table.csv`
   The Shift to Depth table which is described in 3.2.
5. `kin_residues.eps` & `kin_residues_hist.eps`
   The verification results.

Chapter 3
OpenNI

This chapter describes the function of the OpenNI driver for the PrimeSense sensors and my modifications to its source code. Although the official website of the OpenNI was closed in April 2014 the source code is still accessible on the OpenNI GitHub page[5]. I have used the code from the OpenNI2 repository which supports the sensors based on the PS1080 SoC.

The OpenNI consists of more drivers for various depth sensing devices. Both Asus Xtion and PrimeSense Carmine sensors use the PS1080 driver. The driver itself is modular and initializes just the part of the sensor that is used. There are some restrictions such as you cannot access the color and the IR stream simultaneously so it is good to initialize just the modules you really need.

3.1 Disparity vs. Depth

There are two formats of the depth channel output — disparity map (inverse depth) or the real depth in millimetres. The sensor returns the disparity value and the driver is by default configured to convert this value to the real depth. The disparity of the pixel represents the shift of a point in two stereoscopic images[11] (analogically for the pattern projected by the IR projector and captured by the IR camera). In the PrimeSense terminology, the disparity is called a shift value.

3.2 Shift to Depth table (S2D table)

The conversion from Shift to Depth is implemented in the `ShiftToDepth.cpp` file. In the initialization procedure there is created an integer array of 2048 values. The index of the array refers to the shift value returned by the sensor and the corresponding value is the depth value in millimetres. The metric values are computed during the initialization and they are recomputed each time when the relevant configuration parameters changes. When the disparity image comes from the sensor, the driver just finds the depth value for each pixel in the table and it is not necessary to compute it each time.

This associative array is called the Shift to Depth table (abbr. S2D table). During the calculation of the S2D table is also stored the Depth to Shift table which could be used to get the disparity values from existing depth images which were saved with the metric values. However I do not recommend to use such reconstructed shift values for the calibration because the Shift to Depth is not an injective function.

3.3 Original Configuration Options

The processing of the calibration data can be configured using two types of configuration files, each of them is determined to configure different part of sensing. First possibility is to configure the OpenNI using `.ini` files. The other possibility is to configure `openni2_camera` wrapper using configuration files placed in `~/.ros/camera_info/`.

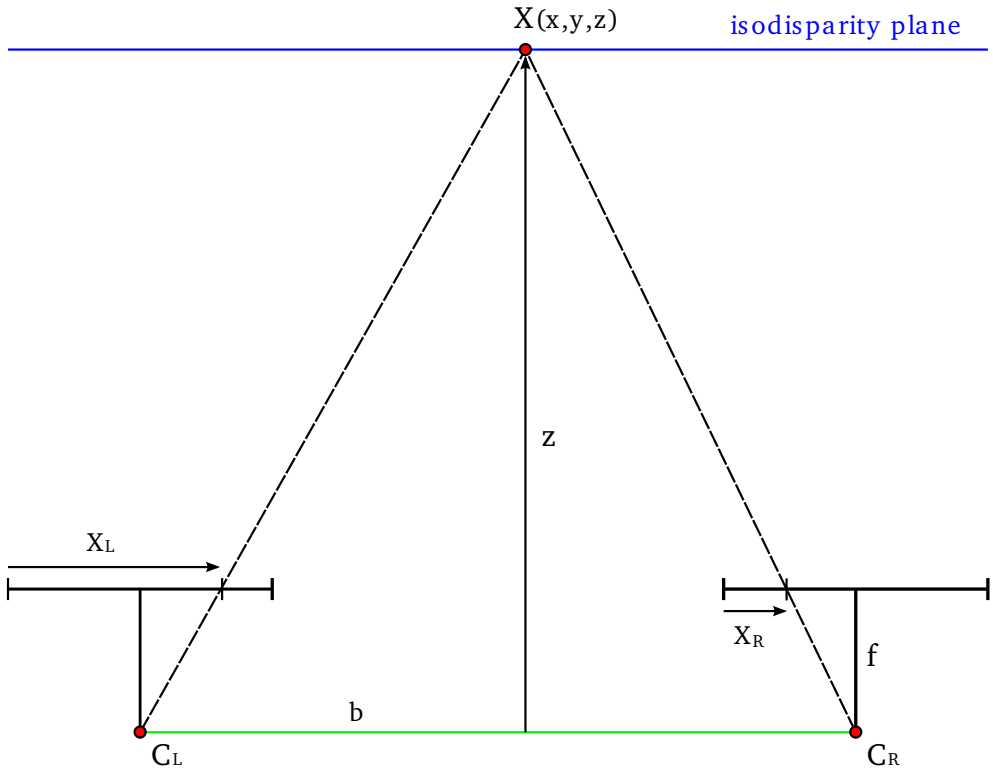


Figure 3.1. Illustration of disparity depth measurement. The `z` value value is computed based on the difference between X_L and X_R — the position of the same object in left and right camera images. Figure is taken from [12].

If you have installed OpenNI using the package manager the configuration `.ini` files are automatically created in `/etc/openni2` directory. If you did not use the packages to install the OpenNI, and you got the libraries with a program, the configuration files must be placed in the same directory as the redistributable libraries [14].

3.3.1 OpenNI configuration

The openni `.ini` configuration files allows us to configure default settings for the device and its streams. Each option in the file is well documented and you should not have a problem with them. You can set for example resolution of the cameras, FPS rate, white balance filter and so on. However the most important parameter for the calibration is the output format of the Depth stream. Make sure it is set to 102 which means the device will return disparity (shift) values of the scene. By default it is set to 100 and the Shift values are converted to the depth in millimetres.

3.3.2 ROS OpenNI Wrapper configuration

The wrapper allows us to configure parameters important for the 3D scene reconstruction. You can configure the intrinsic parameters of the IR and RGB cameras. The configuration files are not created automatically with the installation of the `ros-hydro-openni2-camera` package. I have not found any official documentation of these configuration files. However I have found some files on a GitHub page of another robotic project and I have used their structure of the configuration files[9].

By default the wrapper looks for files named `depth_PS1080_PrimeSense.yaml` and `rgb_PS1080_PrimeSense.yaml` in `~/.ros/camera_info/` directory. The file url can

be different for each sensor and it can be configured as the `depth_camera_info_url` and `rgb_camera_info_url` ROS Node parameters[3].

3.4 Changes made to the driver

Unfortunately the OpenNI drivers does not allow you to use different configuration files for two or more sensors although the sensors parameters are stored separately when they are loaded. I have made changes to the `XnSensor` class which represents the device and creates the data streams. The url to the `PS1080.ini` configuration file is stored in the `m_strGlobalConfigFileName` variable and it is resolved by `ResolveGlobalConfigFileName()` function. I have changed this function to check whether the `/etc/openni2/PS1080-{serial#}.ini` file exists, and if it does, to return it as the global configuration file. Otherwise the `/etc/openni2/PS1080.ini` is used. This is illustrated in Fig. 3.2

There is just one problem and it is that the global configuration file is common for device and streams configuration and you cannot get the serial number of the sensor unless the device is initialized. I have solved this by adding a second call to the file resolving function after the device initialization. It means the device is initialized using `PS1080.ini` file and the streams are initialized with values from the `PS1080-{serial#}.ini` file if it exists. This is sufficient for passing the unique parameters to the Depth stream.

The original OpenNI does not let you to change any parameters related to the Shift to Depth calculation. I have decided not to load the constants for the conversion formula but to load a pre-computed S2D table. This allows you to completely change the formula without any changes to the driver. I have added `CustomS2DTableID` paremeter to the Depth section of the `PS1080.ini` file. The parameter holds the integer value used to distinguish between prepared S2D tables. The sensor tries to load the `/etc/openni2/PS1080-{CustomS2DTableID}.csv` file which consists of semicolon-separated Shift and Depth values. If such a file does not exist the S2D table is computed the old way. The `CustomS2DTableID` value defaults to 0.

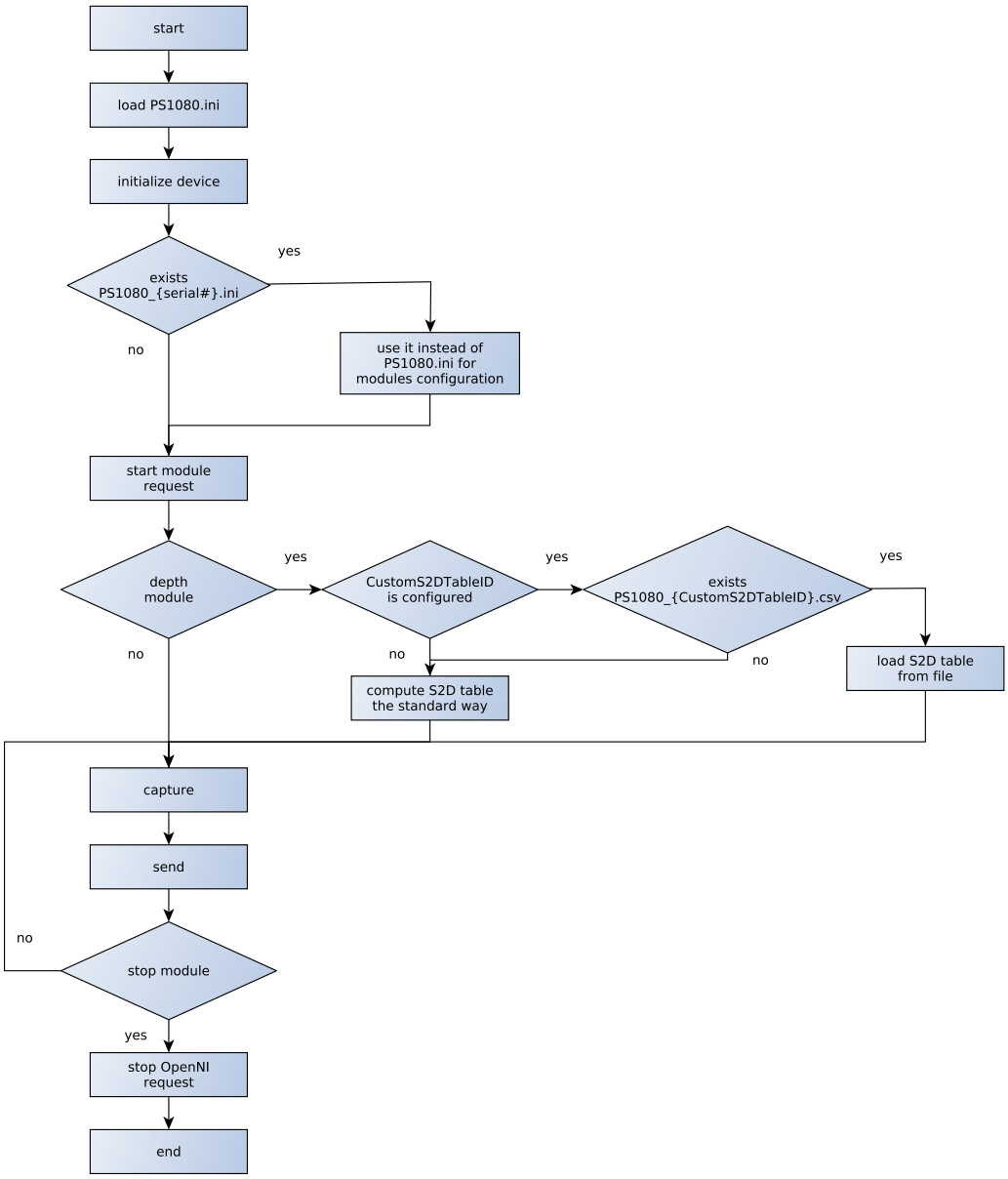


Figure 3.2. OpenNI life cycle flowchart illustrating loading of the configuration files.

Chapter 4
Calibration Data Capturing

It is needed to get a set of RGB, IR and disparity data for various positions of the calibration chessboard. The chessboard positions should be spread over the whole camera field of view and the range of common distances in later application. There are used 14 different poses in [12] with total number of 2205 calibration points. I have used 18 poses of 7×6 chessboard fields that made total number of 630 calibration points which made an improvement in the sensor accuracy however the result was worse (discussed in Chapter 6).

The input files for the calibration MATLAB script should be named `iri_{i}.png` for IR pictures, `rgb_{i}.png` for color pictures and `dep_00{i}.yml` for disparity values where `i` is a number between 1 and total number of image sets. It is not allowed to skip a value. The `.yml` file has a strict format of its header and it has to look the same as the example in Appendix B.

4.1 Capturing Program

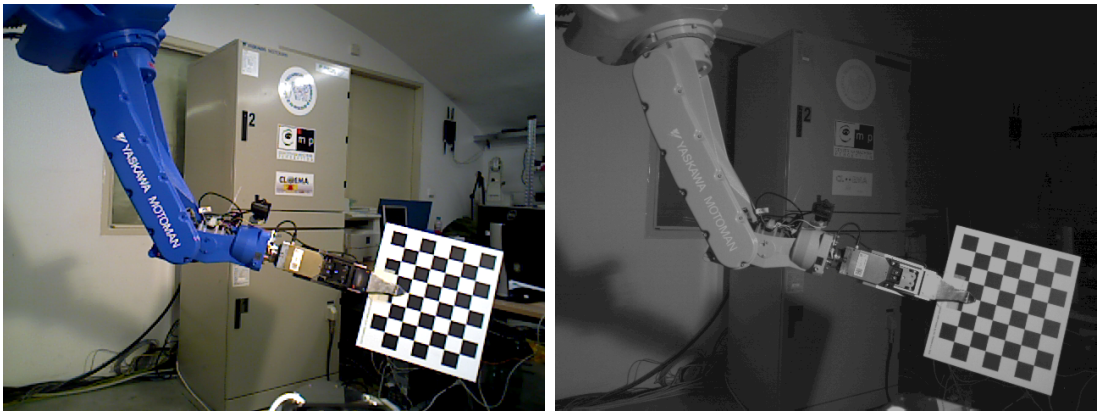
I have created a simple C++ program which can be used to get the data required for the calibration. It is based on SimpleRead program which comes with OpenNI and reads the depth value in the centre of a view. I strongly recommend to have just one sensor connected to the PC where you run this program because it takes the data from a random sensor connected to the system. The program captures IR, RGB and depth data in this order and waits for a button press after each type of measurement.

I have found it is not good to capture pictures right after the sensor initialization. The reason was clearly visible in the case of the RGB camera. If I took the first picture after the sensor initialization the auto white balance filter has not been applied yet, and that resulted in less contrast of the checkerboard pattern. Thus the program waits 1 second after the initialization and then captures the picture.

The program uses OpenNI API for communication with PrimeSense sensor and OpenCV for saving the data. The data are saved in `.png` and `.yml` formats. The captured images are also shown on the screen during the process.

4.2 IR Image

To get a clear IR image it is necessary to cover the IR projector and illuminate the scene with a lamp with high IR emission (e.g. halogen lamp). [12] Then you will get 16 bit output image from the IR camera. The image seems to be all black but the information is there and the chessboard corners extraction works well. The image is saved to the 16 bit `iri_0.png` file. For immediate preview I save and display the image stored to the 8 bit png which is on the other hand not suitable for the corner detection.



a) RGB image b) IR image

Figure 4.1. Example of an RGB and IR image pair. The calibration procedure uses these image to get the ground truth data based on a stereoscopic reconstruction of the scene.

4.3 RGB Image

The RGB image is saved to 3×8 bit `rgb_0.png` image. The RGB image capturing does not need any other adjustments to the sensor however it is good to keep the halogen lamp illuminating the chessboard to get high contrast and low noise image.

4.4 Disparity Map

To reduce the noise in the depth channel it is recommended to take at least 5 pictures and use median for each pixel in the picture[12]. The program does so and produces the output of 5 pictures as taken from the sensor and the 6th picture as the median value for each pixel. This relevant file is saved as `dep_000.yml`. The other files are stored as `depth_(0-4).{png,yml}`. The `.png` files are good for a visual preview if you raise the brightness.

Important: It is necessary to uncover the IR projector and turn off (point away) the halogen lamp prior to taking the depth images.

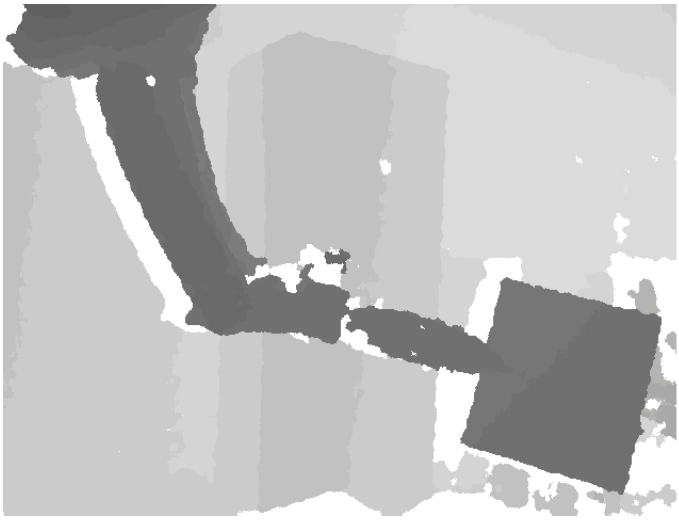


Figure 4.2. Disparity map correspoding to the camera image pair from Fig. 4.1. The depth sensor returns 11 bit values where the maximum value is used for pixels where the device was not able to compute disparity.

Chapter 5
Running the Calibration Procedure

The calibration procedure consists of 3 parts — data collection, MATLAB script execution and application of the output files. In this chapter I will describe the way how go through the calibration procedure and avoid the mistakes I have made before I have found the correct way. I assume you will use my data collecting program and Jan Smíšek's MATLAB calibration script modified by me to directly provide configuration files for OpenNI and `openni2_camera` ROS Node.

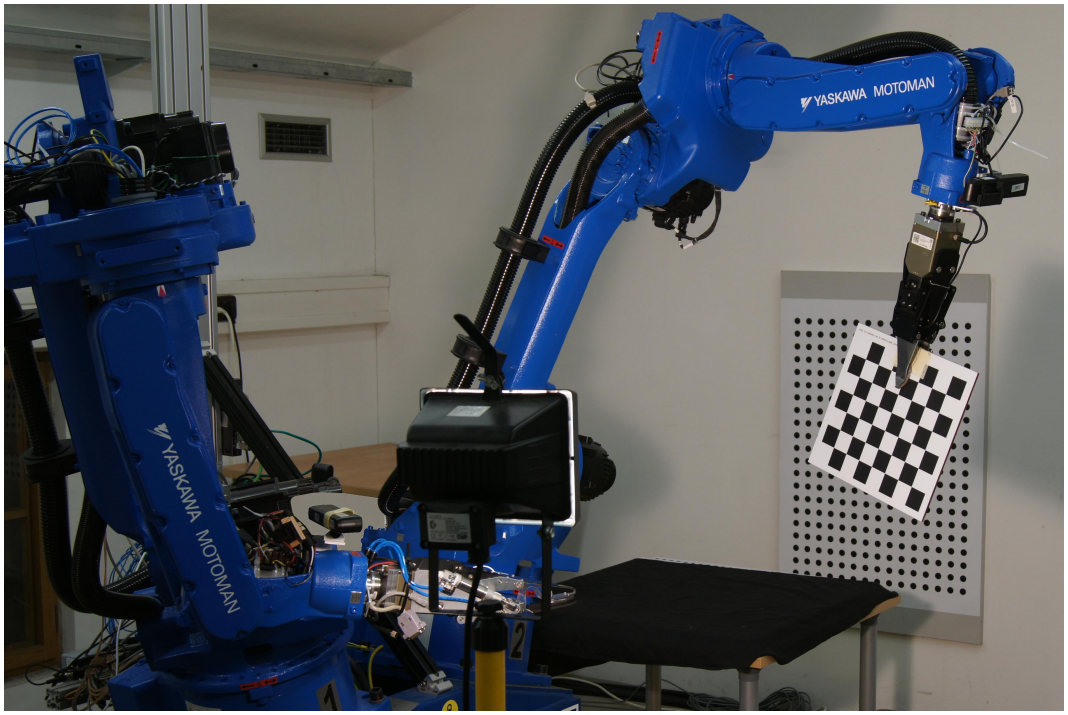


Figure 5.1. Calibration of the Asus Xtion sensor mounted on the arm number one, while the second arm is holding the chessboard. The halogen lamp is used as a source of an infra-red light.

5.1 Step by Step Description

1. Data Collection

    1. Make sure you have just one sensor connected to the system.
    2. Cover the IR projector and light up the chessboard with a halogen lamp.
    3. Run the `DataCollection` program.
    4. Wait until you will see two windows on the screen — 8b and 16b IR image.
    5. Press any button and wait for the RGB image.
    6. Uncover the IR projector and point away the halogen lamp.

7. Press any button and wait until you will see the disparity images (the images seems to be all black because they are 16b `.png` images with maximum value 2047).
8. Press any button to terminate the program.
9. Change the checkerboard position and repeat the procedure unless you think you have enough data.

2. MATLAB Calibration

   1. Change indexes of `iri_0.png`, `rgb_0.png` and `dep_000.yml` files to be ascending from 1 to n (number of images) and place them all in one directory.
   2. Run the MATLAB script: `kinect_calib_2('path_to_data', './')`

3. Output usage

   1. Find out the serial number of your sensor (it is printed on a sticker on the sensor or on the box, use just the part after the dash)
   2. Move the `rgb_PS1080_PrimeSense.yaml` file to ~`/.ros/camera_info/rgb_{serial#}.yaml` (e.g. `rgb_1206040264.yaml`)
   3. Move the `depth_PS1080_PrimeSense.yaml` file to ~`/.ros/camera_info/depth_{serial#}.yaml`
   4. Copy the `/etc/openni2/PS1080.ini` to `/etc/openni2/PS1080-{serial#}.ini`
   5. Move the `S2D_table.csv` file to `/etc/openni2/PS1080-{random#}.csv`
   6. Edit the `/etc/openni2/PS1080-{serial#}.ini` file and set CustomS2DTableID parameter to the random number value from the previous step.
   7. Edit your ROS `.launch` file to load the calibration data:

```
<arg name="rgb_camera_info_url"
   default="file://${ROS_HOME}/camera_info/rgb_{serial#}.yaml" />
<arg name="depth_camera_info_url"
   default="file://${ROS_HOME}/camera_info/depth_{serial#}.yaml" />
```

5.2 Common Problems

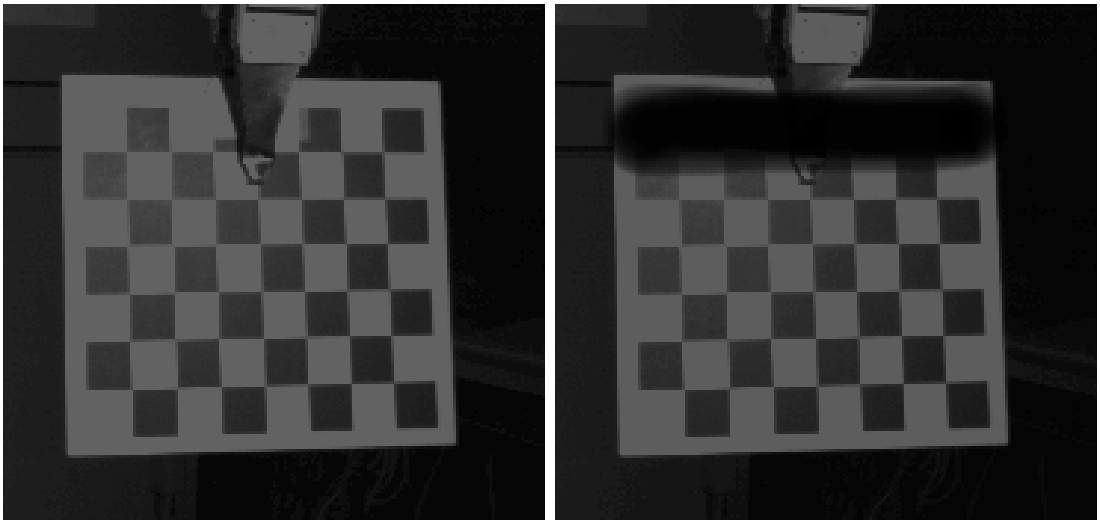
5.2.1 Different count of chessboard corners in RGB and IR images

Sometimes happens that a part of the chessboard is covered for one of the cameras or there can be a shadow on the chessboard. Then the chessboard detected in the IR and RGB images can be different (a row may be missing in one image). When this happens you can see error message like the one below. To fix this use MATLAB Details window to check dimensions of `x_{i}` matrices in files `Calib_Results_IR.mat` and `Calib_Results_RGB.mat`. Then you can remove the affected images from the calibration set or fix them in graphics editor. The sample situation is shown in Fig 5.2.

```
Gradient descent iteration: 1...Error using  -
Matrix dimensions must agree.

Error in calib_stereo (line 343)
            ekk(2*Nckk+1:end) = xrkk(:) - xr(:);

Error in kinect_calib_my (line 88)
calib_stereo;
```



a) problematic image b) simple solution

Figure 5.2. Example of a problematic chessboard image. Part of the chessboard is covered by the gripper. The corner detection algorithm then does not detect the first line in the rgb picture however the line is detected in IR picture with the covered point displaced. The problem can be easily fixed in an image editor.

5.2.2 Abnormally high error in verification

If the result of the calibration verification shows high error (∼50 cm) it is likely caused by presence of a corrupted image of in the calibration data. You need to identify the image set containing the corrupted image. If you do not find anything unusual in the data you need to remove the set from the calibration data. Example of the calibration result when this error occurs is in Fig. 5.3.

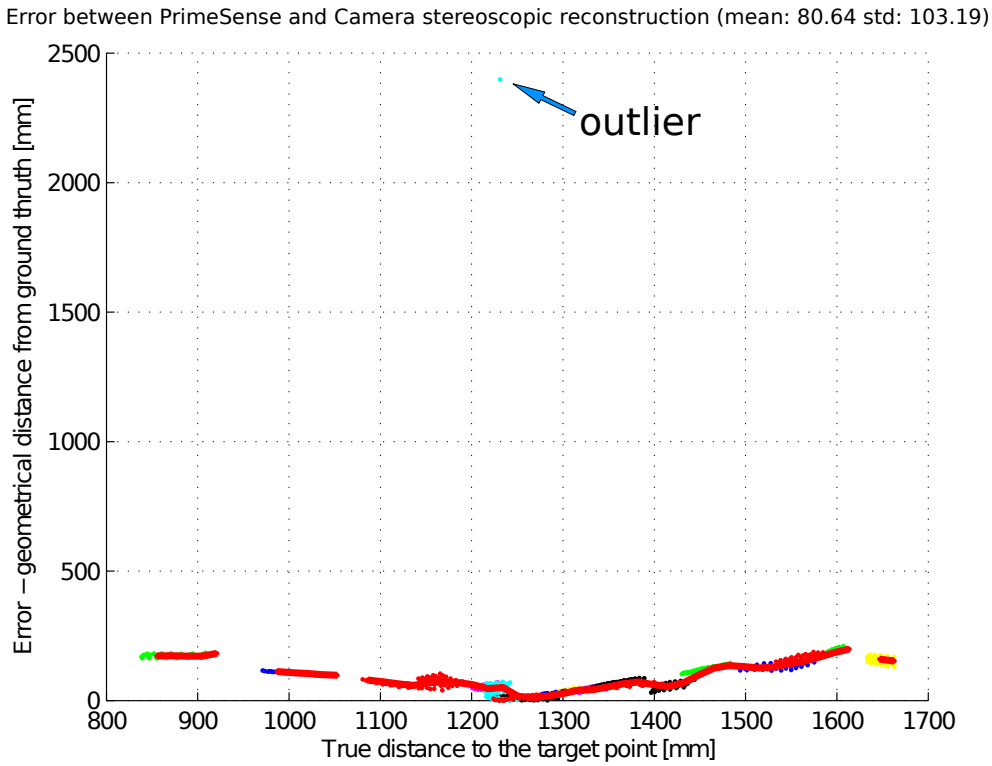


Figure 5.3. When one of the calibration images is corrupted, the result of the calibration can look like this. Solution is removing the corrupted data set from the calibration data.

Chapter 6
Accuracy of the Calibrated Sensor

The accuracy can be verified theoretically and practically. The theoretical verification is a part of the Jan Smíšek's MATLAB calibration script and it is described in 2.3. However the more important are the calibration results in practical measurements. I have created a ROS script for collection of the verification data which are evaluated by a MATLAB script.

6.1 Practical Verification Method

For the practical verification I have used the ROS model of the CloPeMa robot. The depth sensors are included in the model and you can display the view of the sensor in the virtual robot scene. ROS transforms coordinates of RGB-D data to the coordinate system of the robot model. Then you can make measure the end effector position using two different ways. The first way is to use the forward kinematics solver and the other way is to find the end effector in the depth image. I assume the forward kinematics solver is accurate and the position extracted from the depth sensor value should be close to it.

The position of the end effector in depth image is selected manually. This is not a good approach because in the distance where the measurement is made the width of one pixel corresponds to few millimetres. Therefore it is hard to identify the right spot and the measurement error is close to the error of the sensor itself. However it is precise enough to show us whether the calibration has positive effect on the measurement. An improvement of the verification procedure can be a subject of future research.

6.2 Verification Data Collection

The ROS script lets you control the robot using the interactive model and when the robot is in a desired position the script stores the joint positions and end effector position based on the forward kinematics and measurement of the sensor with default calibration. To get the sensor measured value use "Publish point" function of the RViz environment and select the end effector position in the depth sensor point cloud. The published point is caught by the script and saved. This way you can store as many end effector positions as you need for the verification (I have used 10 positions). The collected data are continuously stored to `verification_data.json` file in your home directory.

When you have finished collecting the data measured by the sensor with default calibration, you have to load your configuration to the sensor. Change the `.launch` file of the sensor ROS node to load the camera intrinsic parameters and edit `OpenNI.ini` file to make OpenNI load the custom S2D table. The script for data collection does not need to be restarted. If you have stopped you can run it again and it lets you load the previously collected data.

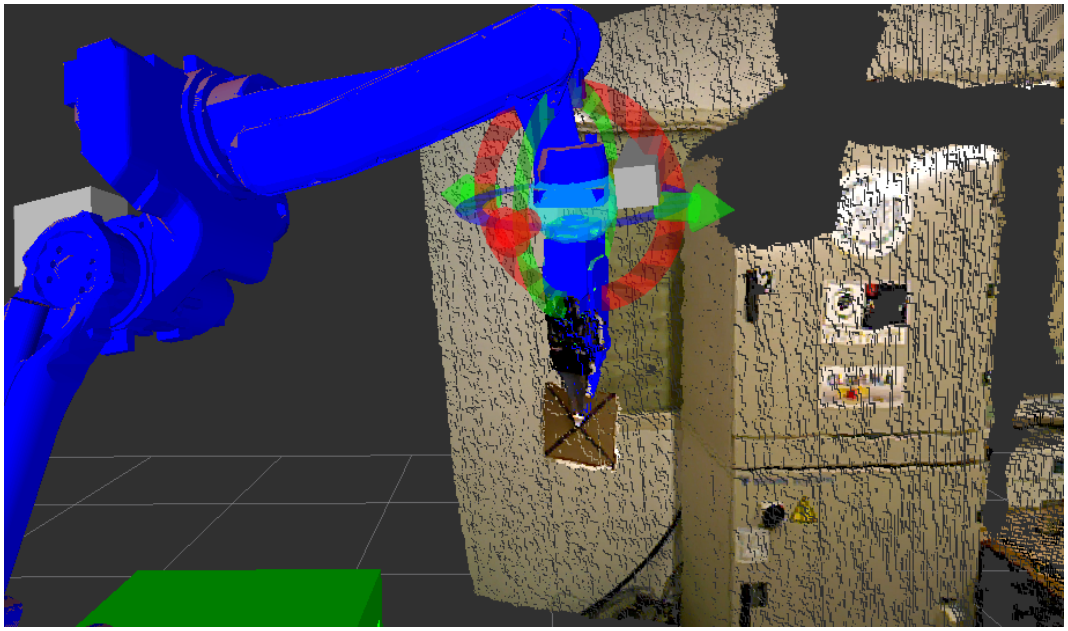


Figure 6.1. RGB-D data measured by the depth sensor can be displayed in the RViz environment in the same scene as the robot model. Data are projected to the scene based on the position of the sensor and you can compare the depth measurement with the virtual robot model.

Now the script will move the robot based on the stored joint positions and will ask you to select the sensor measured coordinates the same way, using the "Publish point" button, as you did before. When all the previously stored positions were reached the script is stopped and the collected data can be found in the `verification_data.json` file in the home directory.

6.3 Verification Data Processing

The output file of the collection script can be loaded by the `verification.m` MATLAB function, which takes the path to the file as the first parameter. The data are loaded using the JSONlab MATLAB toolbox[10] and displayed in a 3D figure. In the figure you can check whether the collected data are correct because sometimes happens you capture the wall behind the target instead of the end effector's position.

To verify the calibration effect, distances between the actual position of the end effector and the position measured by the depth sensor are computed and compared. The output of the script is an average, minimal and maximal value of the error for both measurements. If you would like to see the measured errors for each point, you can call the function with parameter `'output'` set to `'verbose'` e.g. `verification.m('./data.json','output','verbose')`.

6.4 Example of the Calibration Results

For the illustration of the calibration capabilities I have calibrated one of the ASUS Xtion sensors attached to the CloPeMa robot. The calibration and verification procedure followed the technique described in this thesis. Table 6.1 shows the example values of the calibration results and it should help you find abnormalities in you results.

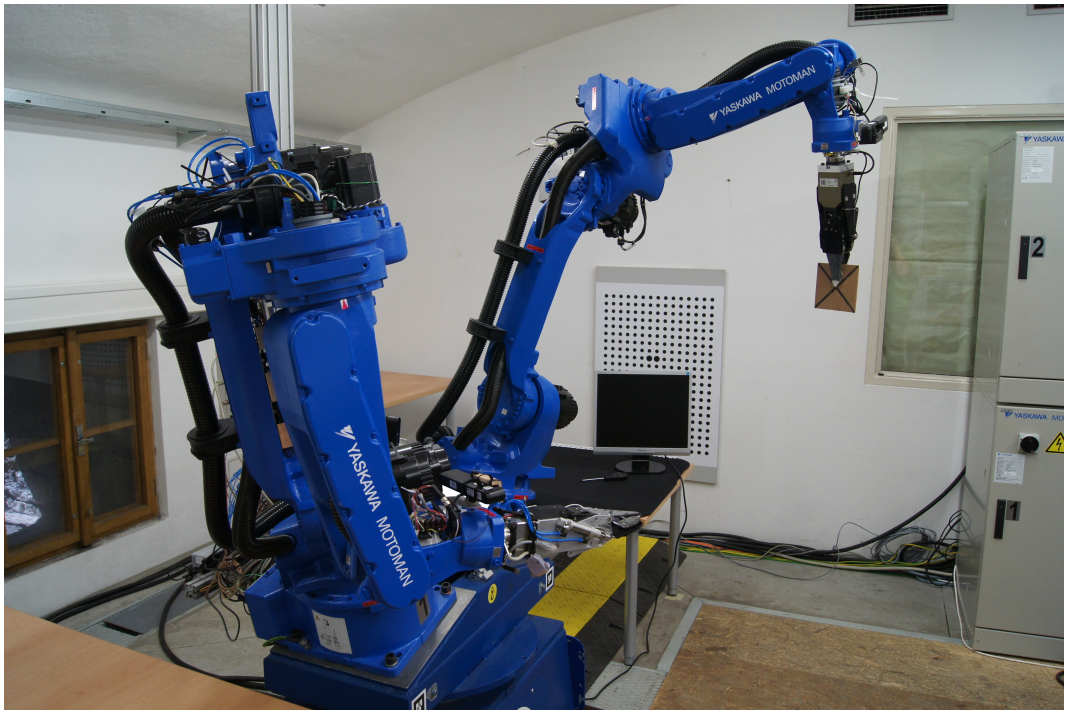
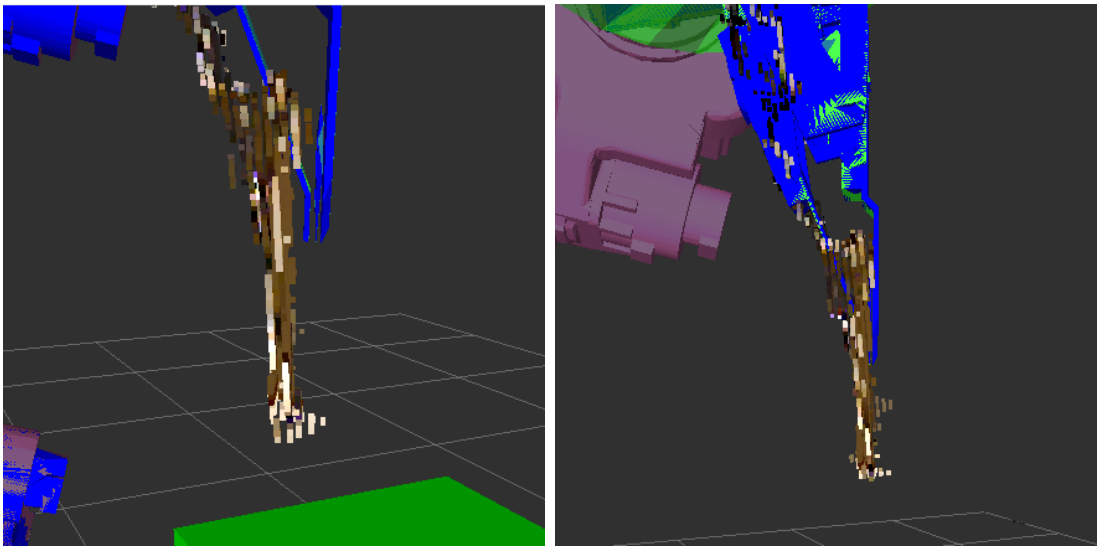


Figure 6.2. The verification data capturing. For better recognition of the end effector the robot is holding a plate with a cross.

The calibration was made using 18 images chessboard with 6×7 fields placed in the distance from 80 to 160 cm. The verification used 10 position of the end effector in a similar distance range. According to the theoretical verification I have achieved depth error mean value approximately 6 mm which is higher than 3 mm error achieved in [12] but I think a better result can be achieved with more calibration points. In practice the average deviation of the values measured by the sensor was decreased from 2 cm to 1.33 cm.

parameter	value
intrinsic parameters	
RGB camera focal length [px]	551.01445
RGB camera principal point [px]	315.09043; 241.21808
RGB camera distortion coefficients	[0.06283, -0.16341, 0.00177, 0.00092, 0.00000]
IR camera focal length [px]	584.53120
IR camera principal point [px]	309.71607; 244.49387
IR camera distortion coefficients	[-0.01008, -0.01601, 0.00286, -0.00004, 0.00000]
depth model parameters	
c_1	$-1.2844266 \cdot 10^{-3}$
c_0	1.4065634
mean value of error [mm]	5.9066

Table 6.1. Simple verification of the camera intrinsic parameter calibration.



a) before calibration b) after calibration

Figure 6.3. The improvement of the accuracy is visible on the position of the plate that the robot is holding. The images shows the depth sensor data projected to the RViz environment. The robot is in the same just the calibration data are different.

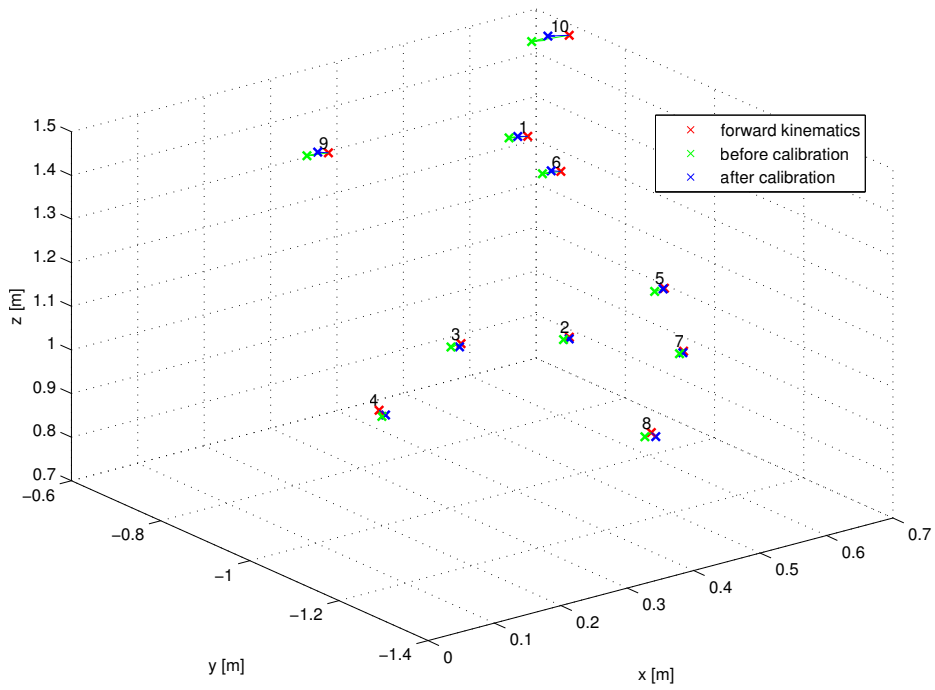


Figure 6.4. The verification data displayed in a 3D plot. Red points are the ground truth points computed by forward kinematics solver. The green points are data measured by the depth sensor before its calibration and the blue points are measured after the calibration.

Chapter 7
Conclusion

Although the PrimeSense sensors are used primarily for a human motion recognition where the best accuracy is not required, with a custom calibration they can be used in robotic applications as sufficiently accurate RGB-D sensors. My primary task was to study the structured light range finder calibration method presented in [12] and to find a way how to use it for the sensors used in project CloPeMa. To be able to perform the calibration and apply the results I have studied principles of RGB-D sensors, ROS environment and OpenNI API.

I have created a standalone program for the data collection. The program is written in C++ and it helps you to collect all data needed for the calibration. With a basic data set I was able to make experiments with the calibration script developed in [12]. I have slightly modified the script to make its use more comfortable and to produce the output files in a format required by the other applications. I found it quite easy to apply the camera calibration data however the more difficult it was to apply the range finder calibration. I had to modify the source code of the OpenNI drivers in order to make them configurable as I needed. Finally with all these adjustments I was able to use computed calibration data within the ROS environment.

To find out whether the calibration was successful there is a verification algorithm in the MATLAB calibration script. However it is good to make verification similar to the real application of the sensor and so I did. I proved the accuracy of the sensor was increased and I think better results can be achieved with more calibration data.

Further improvement of the accuracy is related to the future research which should bring an ability of recognition other calibration patterns, not just chessboards. Also the suitability of equation (1) could be discussed and with a large set of calibration data it might be replaced by an associative array computed using only the calibration data and not fitted to a function. There definitely are possibilities how to improve the calibration procedure and resulting accuracy of the sensor and my thesis, which made the sensor more configurable, enables the the future research in this topic.

References

[1] Ken Conley; Jostein Austvik. Understanding ROS nodes, February 2012. http://wiki.ros.org/Master.

[2] Ken Conley. Nodes, February 2012. http://wiki.ros.org/Nodes.

[3] Piyush Khandelwal; Michael Ferguson; Austin Hendrix; Chad Rockey; Pattrick Mihelich; Carlos Jaramillo; Yiping Liu; Ken Conley; Andrew Sommerville; Greg Olmscheng; Stephane Magnenat; Tully Foote. openni_camera, November 2013. http://wiki.ros.org/openni_camera.

[4] ASUSTeK Computer Incorporated. Support for Xtion PRO Live, December 2013. http://support.asus.com/Download.aspx?SLanguage=en&m=Xtion+PRO+LIVE&p=19&s=11.

[5] Open Natural Interaction. OpenNI, November 2013. https://github.com/OpenNI.

[6] Michael J. Miller. Primesense: Motion control beyond the Kinect, May 2011. http://forwardthinking.pcmag.com/gadgets/282321-primesense-motion control-beyond-the-kinect.

[7] Cory Cross; Chris Lalancette; Jack Thompson; Isaac Saito; William Woodal; Dustin Webb; Chad Rockey; Thibault Kruse; Alex Bravo; Dereck Wonnacott; Melonee Wise; Chris Alexander; Patrick Bouffard; Ken Conley; Tim Field; Steffi Paepcke. Understanding Ros nodes, May 2014. http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes.

[8] Jan Smíšek; Michal Jančošek; Tomáš Pajdla. 3D with Kinect. Technical report, Czech Technical University, 2011. ftp://cmp.felk.cvut.cz/pub/cvl/articles/pajdla/Smisek-CDC4CV-2011.pdf.

[9] Andrew Price. Hubo head description, September 2013. https://github.com/hubo/hubo_head_description/blob/master/launch/.

[10] PhD Qianqian Fang. JSONlab, February 2014. http://iso2mesh.sourceforge.net/cgi-bin/index.cgi?jsonlab.

[11] Jay Rambhia. Disparity map, March 2013. http://www.jayrambhia.com/blog/disparity-maps/.

[12] Jan Smíšek. 3D camera calibration. Master's thesis, Czech Technical University, 2011. https://support.dce.felk.cvut.cz/mediawiki/images/1/18/Dp_2011_smisek_jan.pdf.

[13] Dean Takahashi. Beyond kinect, Primesense wants to drive 3D sensing into more everyday consumer gear, 2013.

http://venturebeat.com/2013/01/20/beyond-kinect-primesense-wants-to-drive-3d-sensing-into-more-everyday-consumer-gear/.

[14] Tomoto Washio. Search path for redist files, 2012. http://community.openni.org/openni/topics/search_path_for_redist_files.

[15] William Wong. How Microsoft's primesense-based Kinect really works, March 2011.
http://electronicdesign.com/embedded/how-microsoft-s-primesense-based-kinect-really-works.

Appendix A
Abbreviations

ROS	Robot Operating System
CloPeMa	Clothes Manipulation and Perception
OpenNI	Open Natural Interaction
RGB	Red Green Blue (color image)
IR	Infra Red
RGB-D	Red Green Blue - Depth — 3D color model of a scene
S2D	Shift to Depth
SoC	System on a Chip
PS	PrimeSense
API	Application Programming Interface

Appendix B
Examples of Files Used in the Calibration

B.1 Depth YAML File

```
%YAML:1.0
depth_data: !!opencv-matrix
   rows: 480
   cols: 640
   dt: w
   data: [ 644, 644, 644, ... 2047, 2047, 2047 ]
```

B.2 Camera intrinsic parameters configuration file

The file structure was found in [9].

```
image_width: 640
image_height: 480
camera_name: rgb_PS1080_PrimeSense
camera_matrix:
  rows: 3
  cols: 3
  data: [551.01445, 0, 315.09043, 0, 551.01445, 241.21808, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.06283, -0.16341, 0.00177, 0.00092, 0.00000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [551.01445,0,315.09043,0, 0,551.01445,241.21808,0, 0,0,1,0]
```

B.3 Custom S2D Table

```
1;324
2;325
3;325
4;325
```

```
5;326
6;326
7;326
.
.
.
1050;7872
1051;8051
1052;8238
```

Appendix C
Content of the Enclosed CD

- `/BP-2014-Vitek_Jan.pdf` — digital version of this text
- `/text_src/` — TEX sources
- `/OpenNI/` — modified source codes of the OpenNI drivers
- `/OpenNI/dist/` — Ubuntu packages of the modified OpenNI
- `/DataCollector/` — source codes of the program for calibration data collection
- `/SmisekCalib/` — modified version of files from [12]
- `/ROS/` — source files for ROS
- `/data/` — example of the calibration data set and results