Laboration 2: RGBD-cameras

Sensors and Sensing

Marek Bečica, Tom Olsson

December 3, 2015

All code for this exercise can be found at
https://github.com/tgolsson/sensors-laboration2-xtion

Contents

Code listings

List of Figures

List of Tables

1 Theory and motivation

1.1 RGBD-cameras

RGBD-cameras, short for *Red-Green-Blue-Depth*-camera, is a type of low-cost camera commonly used for robot vision. The concept became widely popular with the release of the Microsoft Kinect in late 2010.

These cameras consist of two separate parts: one normal color-based camera, and one infrared sensor with accompanying projector. The sensing consists of projecting a deterministic pattern onto the scene, and then unprojecting them by comparing to previously captured patterns at known depths. By interpolating through these patterns, a full depth-image is generated.

1.2 Noise

A common problem in any type of sensing is the introduction of noise into the system. This noise can come from many sources, and be predictable or unpredictable. Examples of noise sources could be frequency hum from electric circuits, flickering lights, air pollution or pure inaccuracy. This noise can skew the results of sensors that make algorithm much more error prone.

There are many approaches to reduce noise. Proper calibration and good testing environments is a good start, but this can only reduce external noise. Internal noise of the sensor needs to be analyzed and minimized on a much lower-level such as by using specially constructed algorithms. For sensors, that generate some sort of sequence one very naive (but nonetheless effective) approach is the use of smoothing.

2 Implementation

The purpose of this exercise is to calibrate an RGBD-camera and investigate its characteristics. Then, several smoothing algorithms shall be evaluated for the depth data.

2.1 Hardware and environment

This exercise was performed using an *ASUS Xtion Pro*. The camera was connected over *USB2* to a laptop running Linux kernel 4.2.5. The communication to the camera is done using the *Robot Operating System* [ROS] version *Indigo Igloo*. All packages used are compiled directly from GitHub development branch for Indigo Igloo.

Other software used includes the OpenCV libraries, version 2.4.12.2-1.

2.2 Camera setup

2.3 ROS setup

Instead of using RVIZ to view the data as before, a custom ROS-node can be used. As there are three types of data - color image, depth image, and pointcloud, there are three listeners setup to receive this data. An example of the data on these topics can be found in the embedded files below.

Pointcloud file RGB-image file

2.4 Camera calibration

```
image_width: 640
image_height: 480
camera_name: rgb_PS1080_PrimeSense
camera_matrix:
  rows: 3
  cols: 3
  data: [546.589906, 0, 319.850785, 0, 545.054549,
   240.484512, 0, 0, 1]
```

```
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.073113, -0.09862, 0.002178, 0.001978, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [546.70282, 0, 320.499416, 0, 0,
   547.7890619999999, 240.692826, 0, 0, 0, 1, 0]
```

2.5 Noise characterization

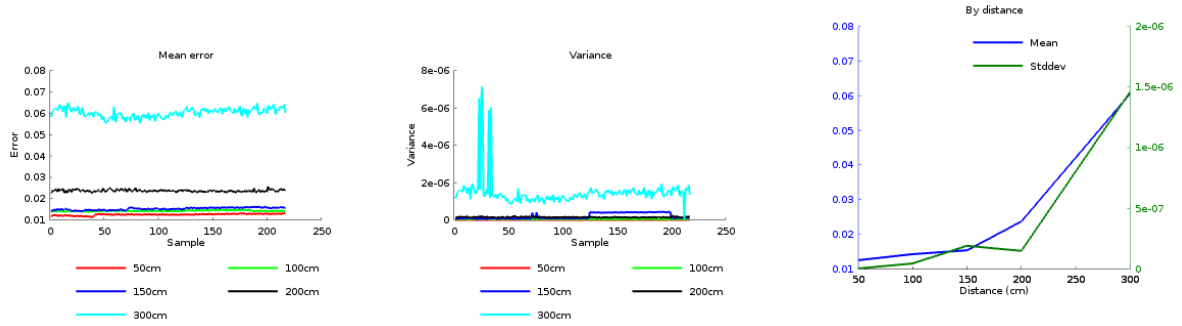

Figure 1: The error and variance for a 20x20 pixel window at the center.

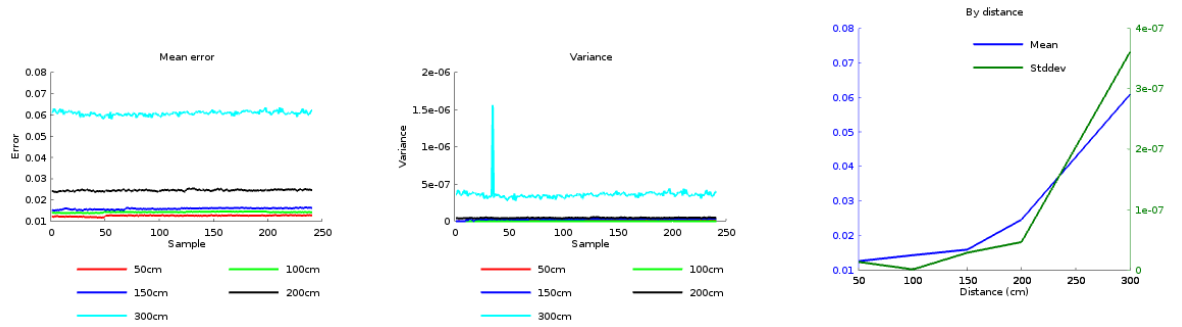

Figure 2: The error and variance for a 40x40 pixel window at the center.

2.6 Noise filtering

3 Results

References

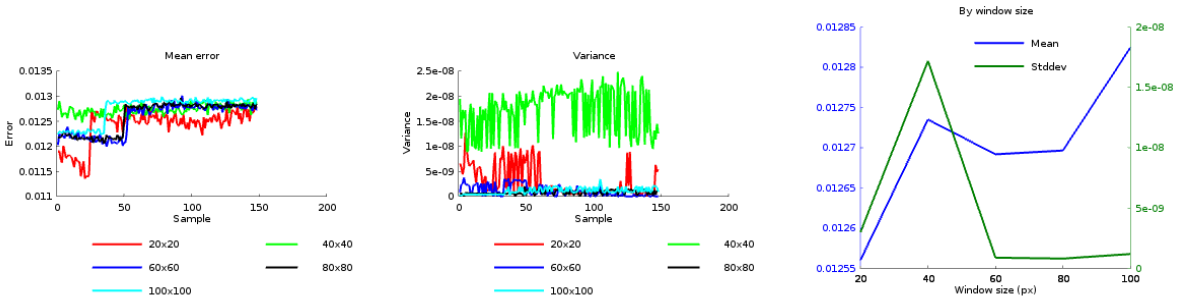

Figure 3: The error and variance at 50cm for various window sizes.