DT00BQ89

Multidimensional Sensing Techniques

Lab exercise 3 - Camera calibration and LiDAR point projection

Petteri Laitinen ÅA21965

Harri Kemppainen ÅA1801751

7.10.2019

[We did the in-class-lab together with Zahid Choudhury, Vid Sustar and Masinde Mtesigwa because of struggle with OpenCV and C++ environment. We continued with the codes after the class on our own, and this report is our own text.]

Introduction

The purpose of this experiment was to learn to calibrate a camera with openCV and project LIDAR data on a frame/video.

Preparations before the classroom lab (assignment A)

Before the lab, we had installed openCV on the laptop. We had also downloaded a provided source code from moodle, unzipped it and familiarized ourselves a bit with the code.

Lab assignment B

There was a digital video camera in the lab that we connected to our laptop. The purpose of the assignment was to do camera calibration, i.e. find parameters to correct the camera picture so that it would produce a device-independent picture that could be used as measurement or source to sensor fusion.

We determined the intrinsic matrix and distortion coefficients for a provided camera (using the provided CameraCalibration/camera_calibration.cpp in the unzipped code)

We changed some of the parameters in the default.xml -file, like number of pictures taken, which was changed to 70, and the calculation method was changed to “Chessboard”.

The command to compile code was:

g++ -std=c++11 camera_calibration.cpp ‘pkg-config --libs --cflags opencv’ -o Calibration

Then we ran the code using command ./ Calibration.cpp. Pressing ‘g’ stopped the program, and we could read the results from an output.

Results

For camera calibration, we calculated values of intrinsic matrix and intrinsic distortion coefficients.
Intrinsic matrix has the form

$$\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

where c_x and c_y give the focal center point of the camera (in pixels), and f_x, f_y tell focal length in x and y-direction in terms of pixels. The focal length coefficients that we got from the program were multiplied by 1,5 because the true resolution 720px was higher than the defaulted 480px (That is how we understood the oral instructions during the lab). (We got $f_x, f_y = 646.48$ - both x and y axis.) The resulted intrinsic matrix was:

$f_x = 970$	0	$c_x = 480$
0	$f_y = 970$	$c_y = 360$
0	0	1

We were really unsure why the focal length values were multiplied, but not the focal center point values. It turn out later that knowing the camera resolution is important, as in happy case the focal center point is in the middle of the screen, and the the focal lengths along x- and y-axis would be equal.

Intrinsic distortion can be characterized by four equations, two for radial and two fro tangential distortion.
Distortion coefficients for radial distortion:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

, and
distortion coefficients for tangential distortion:

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

,
where $r^2 = x^2 + y^2$, both for radial and tangential distortion.

Collecting a set of camera-specific parameters for these equations:
distortion coefficient vector D= $D = (k_1 \ k_2 \ p_1 \ p_2 \ k_3)$.

Intrinsic distortion coefficients obtained:

$k_1 = -0.4662177$

$k_2 = 0.1638718$

$p_1 = 0$

$p_2 = 0$

$k_3 = 0.2586795$

This means that there was no tangential distortion, but there was some radial distortion. In fact, the values of k_2 and k_3 are so high that either the picture is really a lot distorted, or we should have re-done the calibration using the chesserboard.

Lab assignment C

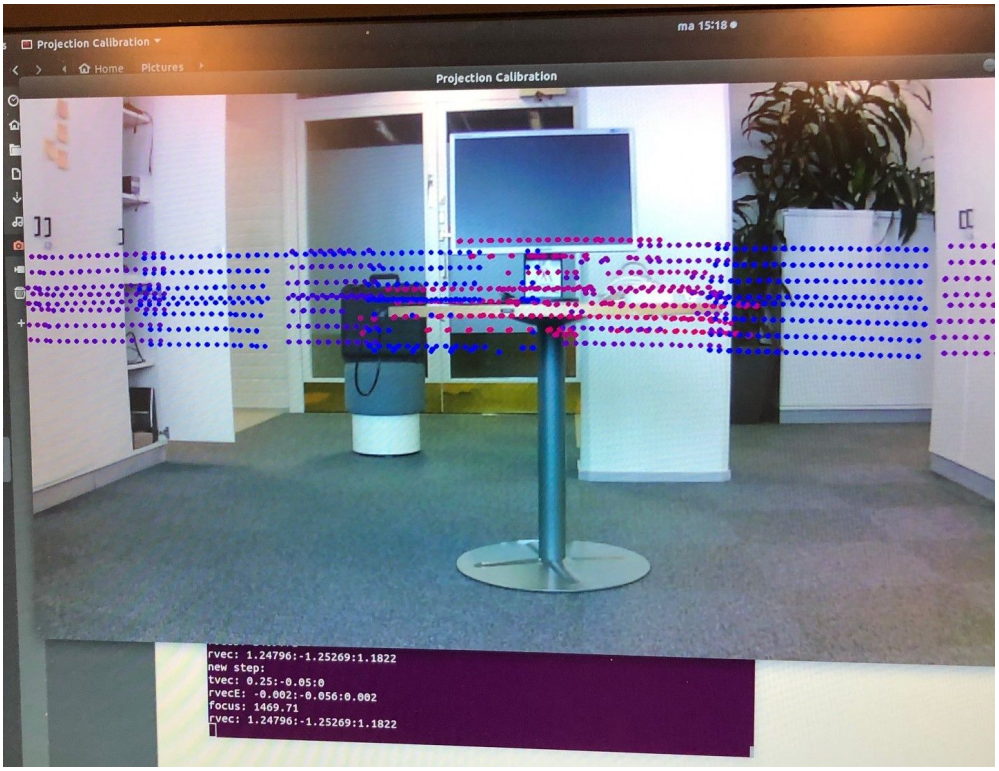
We then inserted the obtained calibration data into the provided code in Calibrate2.cpp, and used it to determine the translation parameters to align the LIDAR point on the image plane of the camera. This was done on one frame (probe.png)

We read the Lidar point data from file LidarPoints.txt, and compiled the code Calibrate2.cpp
g++ -std=c++11 Calibrate2.cpp ‘pkg-config --libs --cflags opencv’ -o Calibrate2

Then we ran the code using command line ./Calibrate2.cpp

We adjusted the lidar points to match the picture. Opposite could have been possible, but it had required more calculations.

Command keys “WASD” and “UHJK” were used for adjusting the lidar points - to calibrate the picture. The calibration was hard, we did not just quite get the lidar points to right places on the picture.



Results

After positioning the lidar points as well as we could during the lab, we got following values for the extrinsic matrix:
Translation vector (tvec): [0.55, -0.05, -0.05]
Rotation vector (rvec): [-0.012, -0.062, 0.116]

The values of translation vector we took as distance between lidar and camera in x-, y- and z-directions. However, we did not know the measurement unit (felt small in terms of pixels!). The values of rotation vector correspond to rotation matrix.

We also got following parameters, but they were not used:
focus=1419.71
rvec: [1.19222, -1.3449, 1.25746], after using Rodrigues' method to transform rotation matrix to rotation vector.

As we were not very happy with the positioning of the lidar points on the image, we tried to have a "second best set" of parameters:
Translation vector tvec: [0.05, -0.1, -1.05]
Rotation vector rvec: [0, -0.03, -0.074]

These values were not good (found out in later stage of the exercise). So we started again (with kind help of the instructor) with values from the video of the exercise. Distortion matrix

there was [[674*2, 0, 320*2][0, 674*1,5, 240*1,5][0, 0, 1]]. With those values in Calibrate2 we got

tvec: -0.1:-0.25:0
rvecE: 0:-0.018:0.014
focus: 1000
rvec: 1.21589:-1.23273:1.21104

Lab assignment D

Finally, we inserted all the previously obtained data (calibration and alignment) into a provided code ProjectionExcercise.cpp. Our final version of this code is in Github.

After we downloaded the video from Moodle we could run ProjectionExcercise. It shows lidar points over the video. Calibration is not perfect, but it's close enough to match found object with the actual object in the video.

Besides point lidar gives rectangles over objects, but rectangles are on a different plane than video. So we need to go through all points and find rectangle they are closest to. From those points we can create a bounding (box) rectangle. We went through all points and matched it to a rectangle. Then from all points matched to a rectangle we created a bounding box by taking min(x1,...,xn) and min(y1,...,yn) from all point. That gives the top left corner for the bounding box. Similarly max(x1,...,xn) and max(y1,...,yn) gives the bottom right corner.

Box was enlarged with 5 pixels each way and drawn over the video.

Results

So old code draws lidar points over the video. Points are off the real objects due to less than optimal calibration, but close enough in most cases to know which object was actually detected. Added code creates bounding boxes over the points and draws them too. Points are matched to objects detected by lidar so that there are separate bounding boxes for each detected object.
Detection is not perfect. There is kind of flashing of points and rectangles at parts of the video. It's hard to say if there's an error in new code or what causes it. Couple of times bounding box is extremely large and it seems that a point that doesn't belong to lidar rectangle is included. That can be an error in new code.

Reflections

- We got quite good overview of camera calibration, learnt the concepts of intrinsic and extrinsic matrix, and tangential and radial distortion.
- We got an understanding how to calibrate a camera using chessboard pattern.
- We got an understanding what it takes to use two camera sources and combine them.
- Finally, we got an overview how to implement these using OpenCV.