3. Assignment, Introduction to Robotics WS17/18 - Ver 1.02

Prof. Daniel Göhring, Martin Dreher, Jakob Krause
Institut für Informatik, Freie Universität Berlin
Submission: online until Tuesday, 14 Nov 2017, 11:55 a.m.

Please summarize your results (images and descriptions) in a pdf-document and name it, e.g., "RO-03-<surnames of the students - group name>.pdf".
Submit your python code file and the rosbag-file (max. 1 MB) in addition
Only one member of the group must submit the necessary files.
Do not copy solutions to other groups.
Every group must contain two people.
Only submissions via KVV will be accepted.

Each group (not each person) should go to https://doodle.com/poll/dx53hqpeffgrxncd to sign in for ONE time slot in the robotics lab. There should not be more than 5 groups in the lab in parallel.
If the time in the lab does not suffice, you have two options:
1. Take a rosbag file to take the data home. Or
2. Come back to the lab again, just by chance there might be a car available to work with. We encourage you to work with the model cars as much as you want (whenever the lab is open).

1. Connect to the model car via SSH (1 Point)

Connect an ethernet cable between your computer and the model car. Read the relevant Wiki pages for connecting to the Odroid and Network configurations:
https://github.com/AutoModelCar/AutoModelCarWiki/wiki

Then create an SSH connection to the model car 1. via ethernet as well as 2. via WLAN. If you don't have an SSH client application installed on your machine, now is a good time to install it.

Create a text file "hello_car_XY.txt" in the model car's **/root/** folder, where XY is the name of your group. Open the file with a terminal text editor like **nano** or **vim**, write the current date and time into the file and save the file. After that, from a new **local** terminal view (not from an ssh-terminal on the car), copy the file to your machine using the **scp** command.

Create some screenshots showing that you performed these steps.

2. Create a repository (1 Point) (Mandatory)

Fork the repository https://github.com/AutoModelCar/catkin_ws_user.
It would be nice if some of your results may be merged into the AutoModelCar repository at the end of the course.

If you don’t want to have a publicly available repository during this course, you can (and have to) create a private one and provide access to the tutors. As an alternative to github you can (and have to) create a repository at https://git.imp.fu-berlin.de. Make sure the content of the current **version-3** branch is included in your repository, too.

Write the url of your repository as a solution of this task.

3. Prepare the field (if it has not been prepared so far) (1 Point)

Using white tape add six marks on the floor and fix the car position in the field. You should choose the distance between the marks. Locate the marks in a symmetrical way in order to get the real world coordinates easily (see fig 1). Measure the distance of the car relative to the points when you take the picture. Record a rosbag file in case you want to use the image data later at home.
It is not allowed to use another group’s data. Each group has to record its own data! Submit the rosbag-file (not larger than 1 MB).

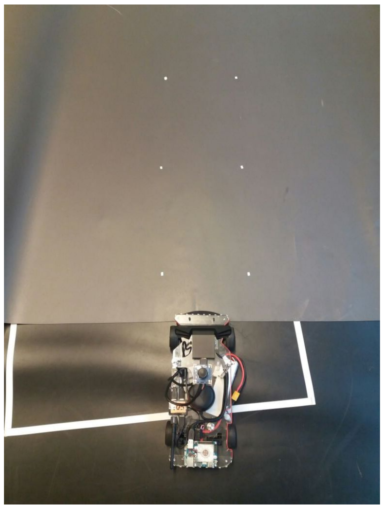
Fig 1. Field.

4. Monochrome (grayscale) image (1 Point):
Take a look at the code of “py_image_processing_example.py” in https://github.com/richrdcm/catkin_ws_user/tree/master/src/py_image_processing.
Using the cv_bridge library, obtain a RGB image from the camera and transform it to grayscale. Publish the grayscale image in a new ros topic. (see fig. 2)

5. Black and white image (1 Point):
Use the OpenCV library “threshold” function, turn your grayscale image from the previous step to a black/white image. Play with the interval parameters until you find the nicest threshold - where white marks and black background are left in the image only. Publish your image and copy it into your submission pdf. (see fig. 2)

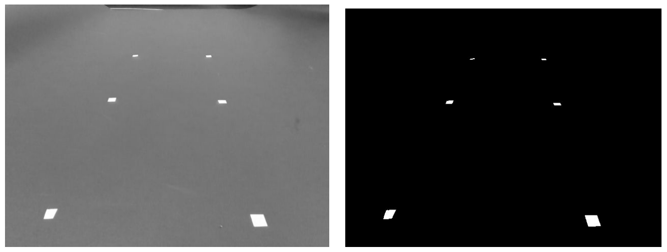
Fig. 2 Gray and black/white image.

4. Find the white points in the image (3 Points):

You can use simple “for” loops to search for the white pixels by scanning the complete image. Remember that the coordinates of the image are like follows:

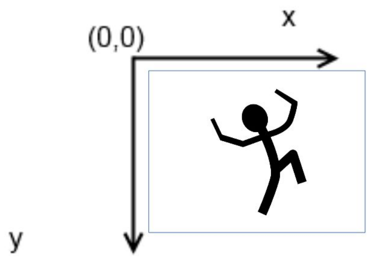

Fig 3. Image coordinates.

Where the “x” axis is the width of the image (in this case 640 pixels) and the “y” axis is the height of the image (in this case 480 pixels). Save the detected white points in an array. Print the points in the terminal.

5. Compute the extrinsic parameters (3 Points):

Here you will have to use “solvePnP” function of OpenCV to calculate the extrinsic camera parameters using the points obtained in the last part. Take a look at the documentation in:

http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

Define a 3x3 cv::Mat matrix for the intrinsic parameters and use the following values:

fx = 614.1699
fy = 614.9002
cx = 329.9491
cy = 237.2788

Define a 4x1 cv::Mat vector for the distortion parameters and use the following values:

k1 = 0.1115
k2 = -0.1089
t1 = 0
t2 = 0

Define an array to save the real world coordinates for each point in the same correspondence that you use to save the pixel coordinates in the previous part. You have to define which of your point will be the World frame ([0,0,0]).

Create two 3x1 empty cv::Matrix, one for the rotation vector and the other for the translation vector that we are going to receive from “solvePnP” function.

Print your results in the terminal.

6. Finding the camera location and orientation (2 Points):

Use the “rodrigues” OpenCV function in order to obtain the rotation matrix from the rotation vector obtained in the previous step.

Now calculate the inverse of the Homogeneous transform using the rotation matrix and the translation vector.

Calculate the angles (roll pitch and yaw or euler angles) from the rotation matrix.

Print your results in the terminal and check the if the location and orientation of the camera is correctly related to the defined “World Origin”.

Hint: There is an implementation which performs tasks 4-6 available in C++ :

https://github.com/richrdcm/catkin_ws_user/blob/master/src/camera_pose/src/camera_pose.cpp

To watch the rosbag-file, start the roscore, play the rosbag file, e.g., in a loop, then run the following command to see the images:

```
rosrun image_view image_view image:=/app/camera/rgb/image_raw
```