Introduction to Robotics WS18/19

4. Assignment: Finding camera position and orientation using extrinsic camera parameters.

Objective: In this exercise you will code a program to read white points in the floor with the RGBD camera of the car and use those points to compute the location and orientation of the camera in the space. In order to achieve this objective, functions from OpenCV library must be used.

1. Prepare the field :

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.)

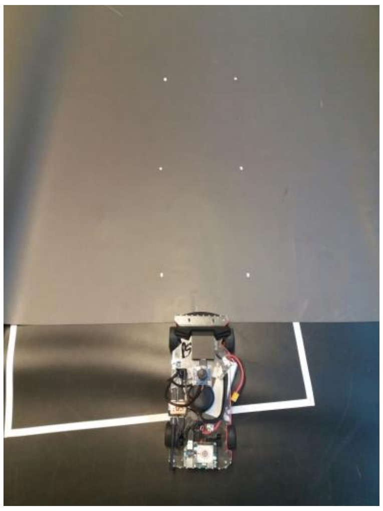

Fig 1. Field.

2. Gray image (1 Points):

Take a look at the code of "py_image_processing" in https://github.com/richrdcm/catkin_ws_user/tree/master/src/py_image_processing . Using cv_bridge library obtain the RGB image from the camera and transform it to gray. Publish your image. (see fig. 2)

3. Black and white image (1 Points):

Using OpenCV library "threshold" turn your gray image from the previous step to a black/white image. Play with the intervals until you find the nicest threshold where only the white marks are left in the image. Publish your image. (see fig. 2)

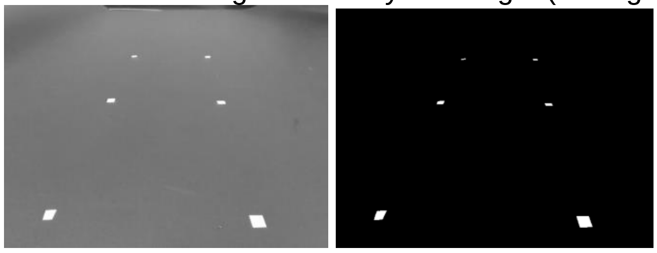

Fig. 2 Gray and black/white image.

4. Find the white points in the image (2 Points):

You can use simple "for" loops to search for the white pixels scanning the complete image. Remember that the coordinates of the image are as follows:

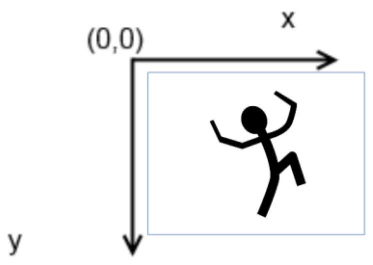

Fig 3. Image coordinates.

Where the "x" axis is the with 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 points you found in an array. Print the point coordinates in a 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 numbers:

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 numbers:

k1 = 0.1115;
k2 = -0.1089;
p1 = 0;
p2 = 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 the "solvePnP" function.

Print your results in a terminal.

6. Finding the camera location and orientation (3 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 a terminal and check the if the location and orientation of the camera are correctly related to the defined “World Origin”.

Example code for solvePnP:

```
fx = 614.1699
fy = 614.9002
cx = 329.9491
cy = 237.2788
camera_mat = np.zeros((3,3,1))
camera_mat[:,:,0] = np.array([[fx, 0, cx],
                              [0, fy, cy],
                              [0,  0,  1]])
k1 =  0.1115
k2 = -0.1089
p1 =  0
p2 =  0
dist_coeffs = np.zeros((4,1))
dist_coeffs[:,0] = np.array([[k1, k2, p1, p2]])

# far to close, left to right (order of discovery) in cm
obj_points = np.zeros((6,3,1))
obj_points[:,:,0] = np.array([[00.0, 00.0, 0],
                              [21.8, 00.0, 0],
                              [00.0, 30.0, 0],
                              [22.2, 30.0, 0],
                              [00.0, 60.0, 0],
                              [22.0, 60.0, 0]])

retval, rvec, tvec = cv2.solvePnP(obj_points, img_points,camera_mat, dist_coeffs)

rmat = np.zeros((3,3))
cv2.Rodrigues(rvec, rmat, jacobian=0)
```