Naviagtion with Robobot using ArUco

Tommy V. Hansen, s102339

Januar 2019

Contents

1 Introduction

The purpose of this project to investigate to properties of the OpenCV library in respect to an ArUco marker and the possibility of robot navigation from an ArUco marker with a single camera and implement this on the Robobot. Furthermore, a camera calibration is necessary and a way to reuse set camera calibration as the camera parameters only change if the camera is changed or a different lens is added.

2 ArUco

ArUco (Augmented reality mark) is a library in OpenCV for detecting square feducial marks in an image. Furthermore, the library can estimate pose from single or multiple markers in the image. It have the ability to draw markers and create marker dictionaries or use from all ready established marker dictionaries to detect markers[1].

3 Camera calibration

To be able to get a good pose estimation of the ArUco and be able to determine where the camera or Robot in respect to the marker camera calibration is needed. From a calibration we need a camera-matrix A and a vector of distortion coefficients. This tells everything we need to know about the intrinsic parameters for the camera. For calibration there are some different options in the ArUco library in which a chessboard finder is used or a dot finder can be used. The chessboard method is the one used here. For this the size of the chess board is very important. When talking about size of the chessboard it is the inner dimensions of the board we are interested in as it the corners of the chessboard we are looking for. Additionally, we need the precise size of each square on the board. The sizes of the chessboard used for the calibration of the camera is seen in table 1.

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$Distortioncoef = (k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, \tau_x, \tau_y]]]]) \tag{2}$$

Inner-chessboard size	7x9
Square size	20x20mm

Table 1: Chessboard used for calibration dimensions

For the calibration routine we will create a live video-feed from the camera which looks for the corners of the chessboard. And if it is able to find the corners draw lines between them so we can determine if the image we are looking at is good for calibration and we can capture it and save it for the calibration. For a decent calibration we need at least 15-20 pictures and before we have this we will not make a calibration. For this we be using some subroutines from opencv2/calib3d.h library namely, "findChessboardCorners" and "drawChessBoardCorners". "FindChessboardCorners" needs an image, the inner chessboard dimensions and two flags "CALIB CB ADAPTIVE THRESH" and "CALIB CB NORMALIZE IMAGE" which either apply adaptive thresh holding to the image and converts into black and white or normalize the image gamma before applying adaptive thresh holding. It then returns the found points which we input into the draw function which then draws between every inner corner found and shows it live in the camera feed.

When +20 pictures, the more pictures the better, have been taken it is possible to calibrate the camera to get the camera-matrix and the distortion coefficients. In figure 1 we see how the calibration routine draws the points to the chessboard when it have successfully found a chessboard it can use for calibration.

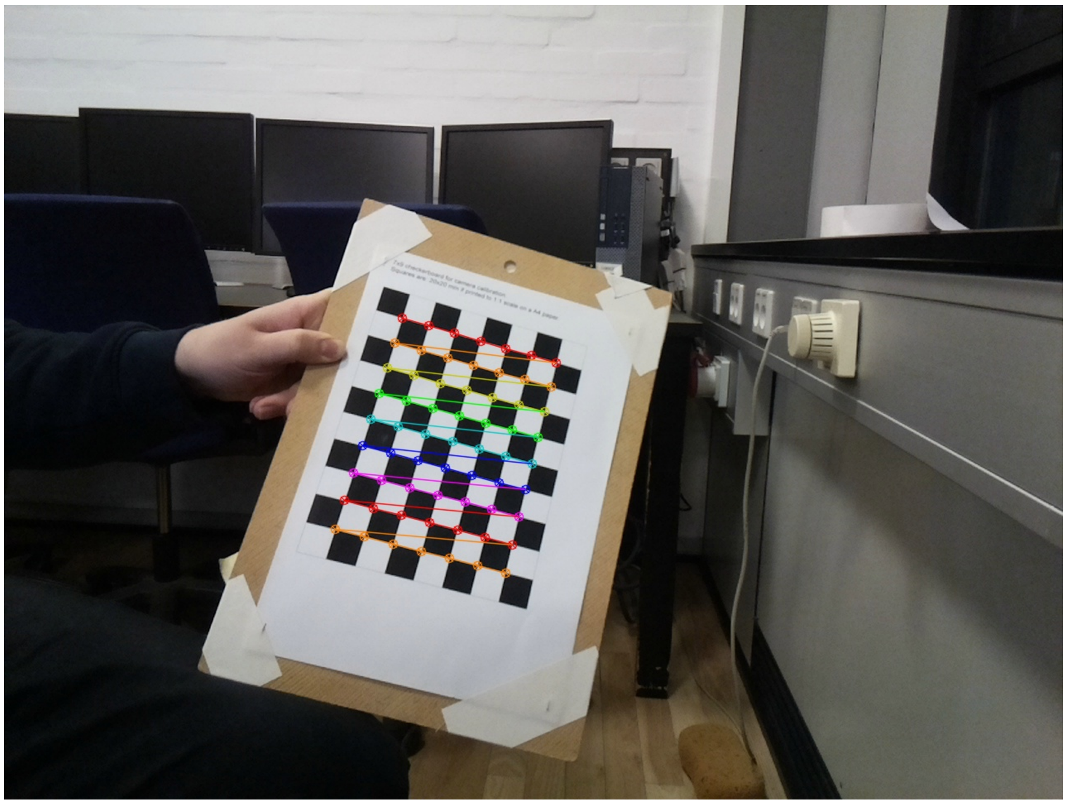

Figure 1: Picture of a successful found chessboard with point drawing

4 Navigation

The goal is for a robot to be able to get an idea of its location in a world coordinate system based on a fixed location of a ArUco marker and thus we need to be able to locate the markers and estimate the robot or camera pose from this detection.

4.1 Marker detection

To be able to detect a ArUco marker a marker dictionary is necessary, the marker dictionary used is a 4x4 50, meaning the marker is a 4 by 4 mark with black edges and the ids in the dictionary goes from 0 to 50. To be able to detect markers live we use the detectMarkers function from the aruco library which finds a marker from an image and returns the marker corners its and its ID based of the marker dictionary.

4.2 Pose estimation

To be able to do pose estimation we will use estimatePoseSingleMarkers function from the aruco library. This functions needs the marker corners, the aruco-sqaure total dimensions, the camera matrix and the distortion coefficients from the calibration and it return a translation-vector(**tv**) and a rotation-vector(**rv**) of the marker.

From the Pose estimation we get a rotation (**rv**) and a translation vector (**tv**) of the marker position from the camera in camera coordinates. However what we are interested in is to get the camera pose in respect to the marker in marker coordinates and thus we need to use the Rodrigues rotation formula[2] to get a Rotation-matrix which we can use to calculate the translation vector i.e. the XYZ-coordiantes we need for navigation. To obtain the Rotation matrix we input the rotation vector we got from the detected marker pose estimation. From the properties of the rotation matrix we know $-R^T = R^{-1}$ and thus we can now calculate the translation vector tv_{mark} , we need in equation 3.

$$tv_{mark} = -R^T * tv \tag{3}$$

When the camera pose is obtained in marker coordinate system we are able to calculate the exact angles and distance to the marker from the camera and thus the robot, as we are given a XYZ-coordinate. Furthermore, from this distance and knowing the offset of the camera position to the center of the robot we are able to calculate the exact position of the robot. However, for simplicity i will just demonstrate the distance the to marker from the camera given coordinates using equation 4. For the robot to move in a world-coordinate system XY-coordinates would only be used unless the robot would move in the z-direction i.e. on plateaus and be able to see the markers on the floor still.

$$CameraDistanceToMarker = \sqrt{x^2 + y^2 + z^2} \tag{4}$$

5 Tests and Results

Here we will place the Robot camera in a known distance from the marker and see how well the it estimates the position of the camera in respects to the marker. In the appendix we see the pictures

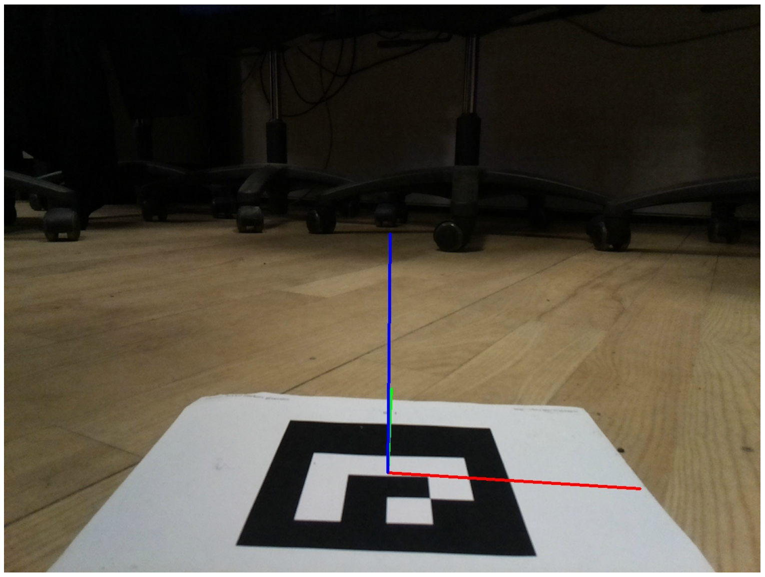

Figure 2: Measurment 5 from results to see how a marker is detected and the marker coordinates from the cam in cam coordinate system is drawn. X:red, Y:green, Z:blue

corresponding to the individual measurements a sample of a picture can be seen in figure 2 in table 2 we see the results for 5 different marker measurements with the same marker though.

Estimated XYZ-coordinates	Calculated distance	Measured distance to center
[0.1891, -0.4687, 0.1319]	0.5223m	0.50 m
[-0.1232, -0.3598, 0.1320]	0.4026m	0.39m
[-0.2291,0.5168,0.1385]	0.5820m	0.52m
[0.0233,-0.7072,0.1307]	0.7196m	0.67m
[0.0020, -0.2482, 0.1255]	0.2781m	0.27m

Table 2: Calculated distances from XYZ-estimated position vs. hand-measured distances

5.1 Results Discussion

From the results we see we get a slight accuracy offset this can be due to poor camera calibration as it was done with only 20 images and 5 included distortion coefficients. For a more precise camera calibration up to 50 images in all XYZ-directions would improve the accuracy much more. The fact that the camera is angled a bit on the robot could also have an impact on the accuracy of the distance measure and if that was included in the positioning calculation it could also improve it. In the Appendix section we see the pictures from the results and in figure **??** we see that the closest the robot can get to the feducial is within 27cm to get a detection and the furthest is about 70cm as we see in figure **??** as the marker is so small and angled so much that the detector have a hard time seeing it on the image. This means we have an approximated range for the camera to detect the marker from 27-70cm. The results also shows us that the rotation of the marker have no immediate impact on the detection or the position estimation.

6 Conclusion

Calibration of the raspberry pi camera has been achieved, and creation of a subroutine to reuse the calibration have been made possible. ArUco marker detection, recognition and pose estimation have been implemented for single markers and multiple markers in the same image. From this Robot position have been calculated, with single marker in image. However with some offset on the distance measure due to poor camera calibration. As time was an issue an implementation onto the Robobot besides programming on the Raspberry pi and using the Pi-camera mounted on the bot an implementation onto the Robobot software was unsuccessful.

Appendix

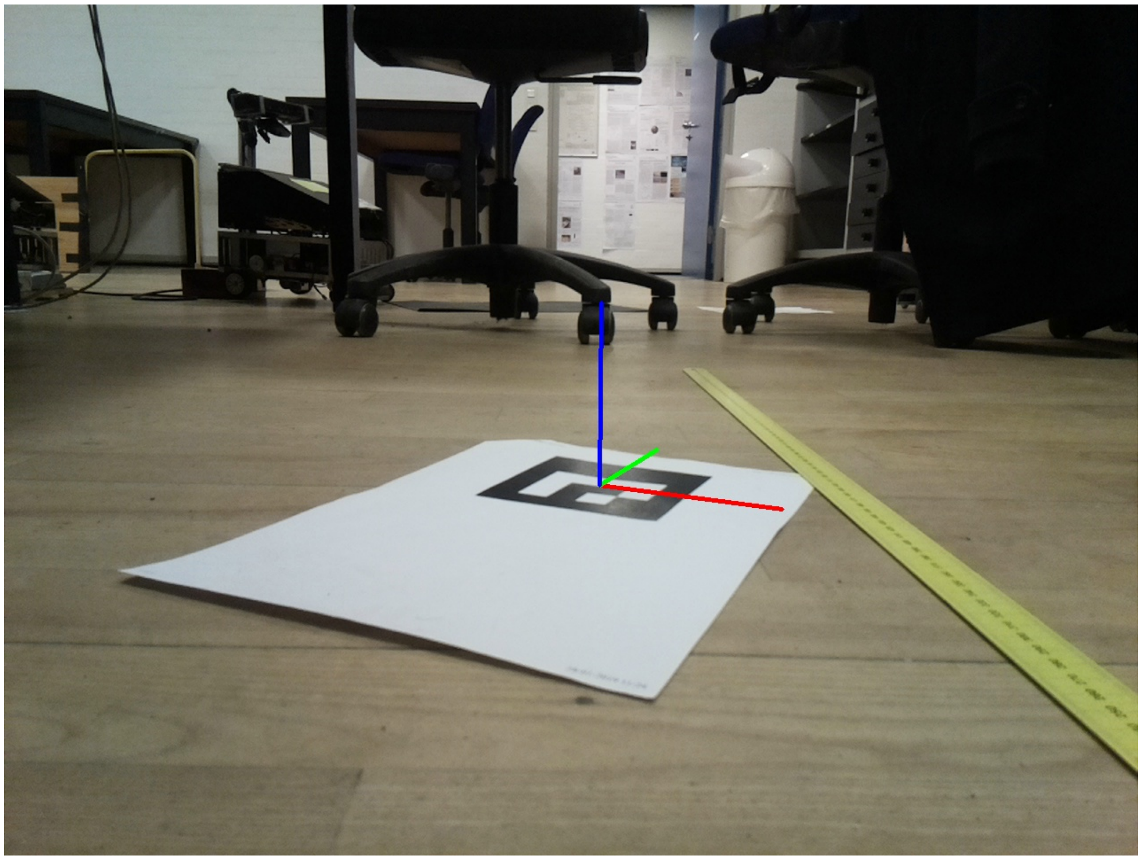

Figure 3: Picture for measurement 1

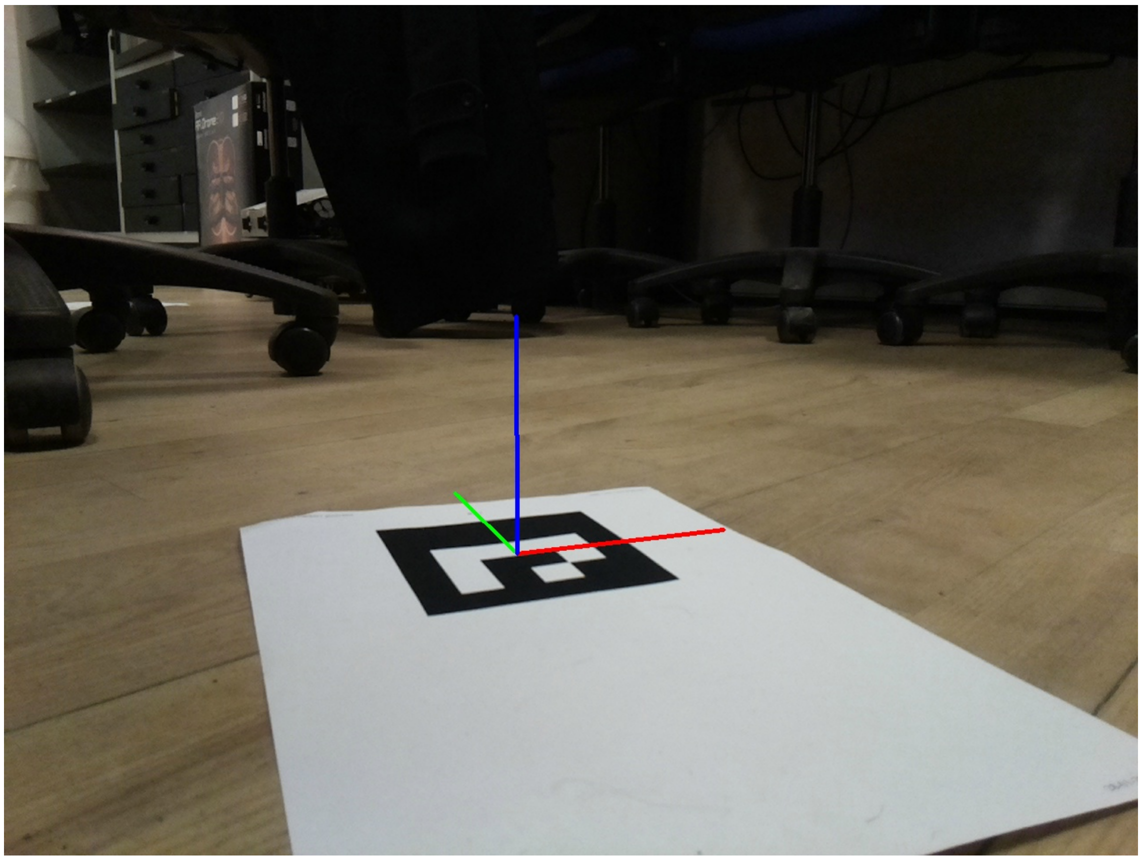

Figure 4: Picture for measurement 2

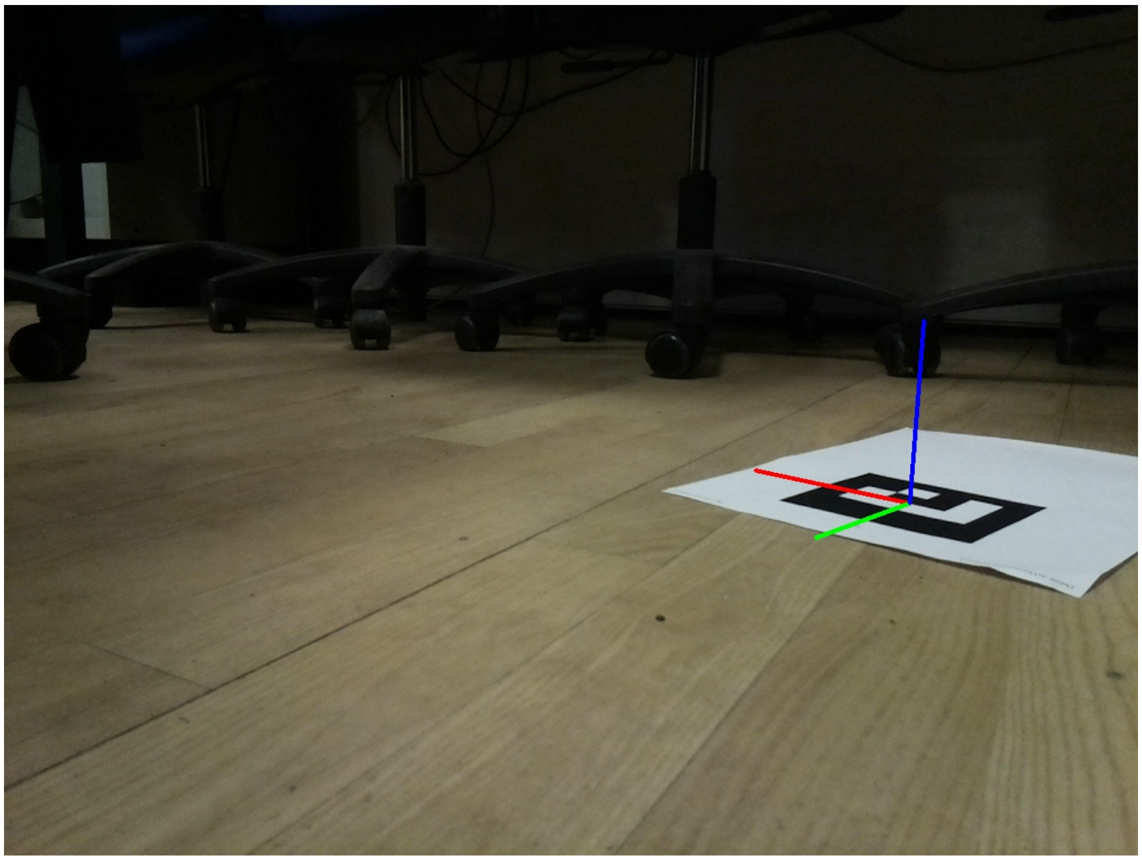

Figure 5: Picture for measurement 3

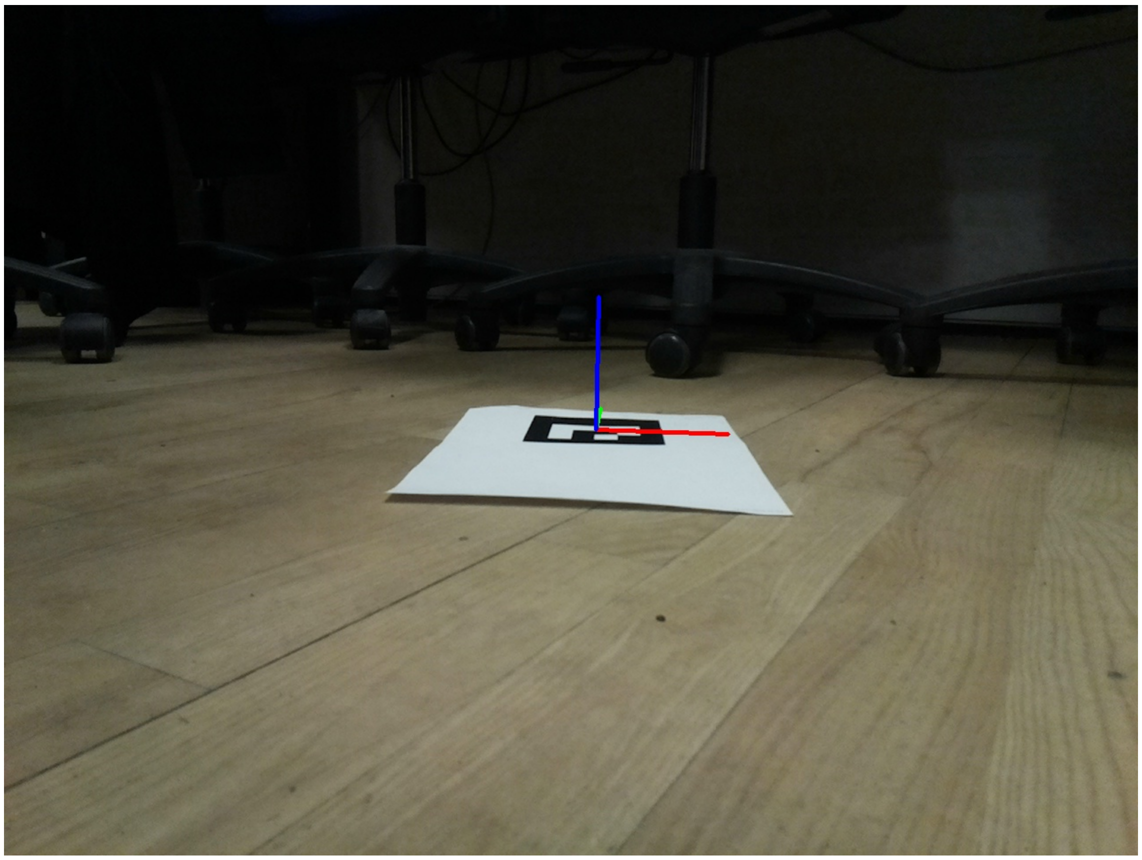

Figure 6: Picture for measurement 4

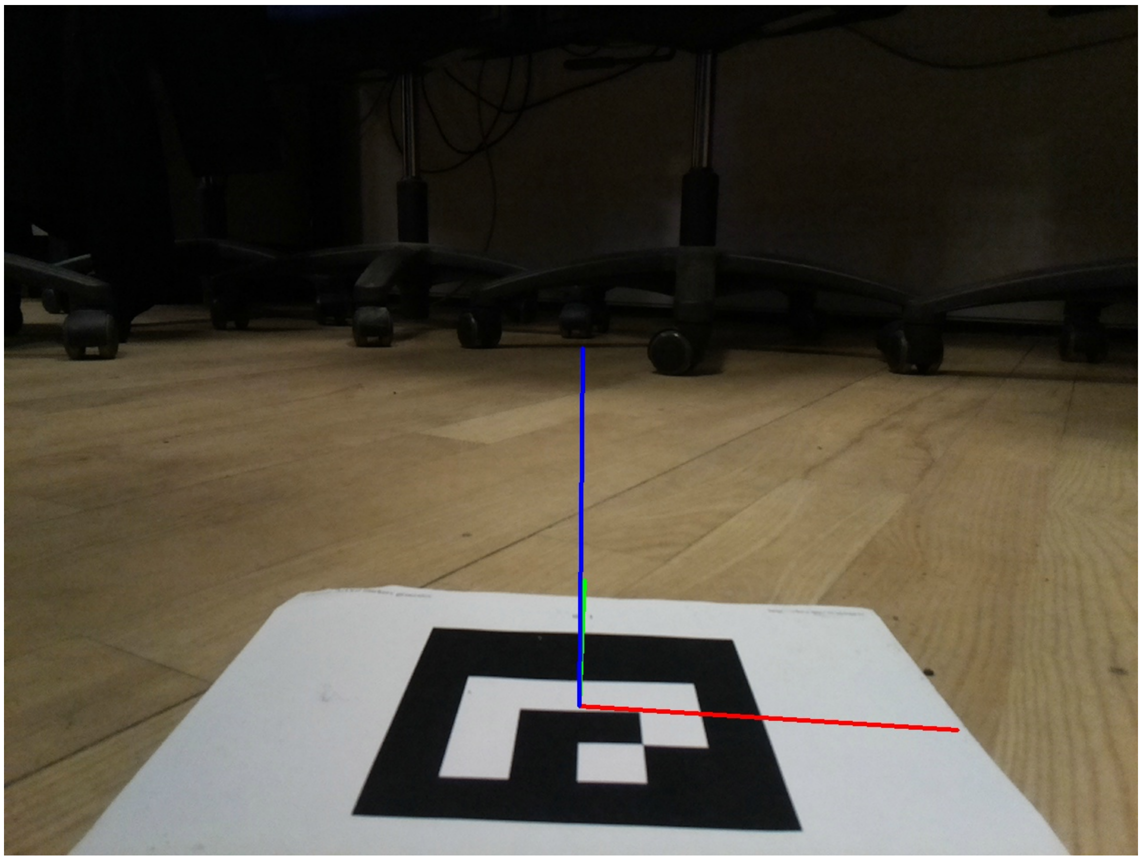

Figure 7: Picture for measurement 5

References

[1] OpenCV. *OpenCV Aruco Library*. https://docs.opencv.org/3.4.5/d5/dae/tutorial$_{a}ruco_{d}etection.html$.

[2] OpenCV. *OpenCV Aruco Library Calib3d*. https://docs.opencv.org/3.4/d9/d0c/group$_{calib3d.htmlga61585db663d9da06b68e70cfbf}$