**INFOMCV Assignment 1**

**Bas de Blok & Joris Heemskerk (group 69)**

**Calibration**

**Run 1:**

|  |  |
| --- | --- |
| Image resolution = 1280 x 720  *Distance says 0.404* |  |

**Run 2:**

|  |  |
| --- | --- |
| Image resolution = 1280 x 720  *Distance says 0.607* |  |

**Run 3:**

|  |  |
| --- | --- |
| Image resolution = 1280 x 720  *Distance says 0.351* |  |

**Discussion of the three runs**

Prior to performing choice task 2, we believed that the run with the most images would likely be the most accurate. See choice task 2 below on why we later changed our minds. Regardless, we first expected the accuracy of task 1 to be best, then task 2 and then task 3, as each has less and less images. Especially the 5 images the algorithm could not automatically detect, which we assumed to be more valuable towards calibration, as they hold more challenging views. While some of these assumptions can be interpreted as correct by looking at the focal lengths, they may not tell the entire story. For example, looking at the focal lengths of the x and y components in run 1, we see very little difference, this is to be expected if there is not too much distortion. While in run 2 we see a very large difference in the x-component of the focal length as compared to run 1. In run 3 we see changes in both components, but now growing smaller rather than larger, and to a much lesser extent. Over the different runs, we see the d values to not change too drastically, only growing slightly larger as the runs decrease in image counts.

While looking at the test images, we see that the cubes in run 1 and 3 look most convincing, with the distance estimation of run 3 being the least off. This would seem to suggest that run 3 is most accurate. Again, we reflect on this in choice task 2.

A person holding a checkerboard

AI-generated content may be incorrect.**Manual calibration tool**

Since this part is worth 15 points, it might be worth writing about, even though this was not asked for. We made a tool that allows you to zoom into a pixel (right click) you want to select as the corner pixel. An example can be found to the right. Once a corner is selected (left click), a text appears that shows you where you clicked. Once 4 corners have been selected, the image can be closed and our code will evenly divide all the inner chessboard corners on the image. For this, we manually interpolate the points using math. See the code.

**Explanation of the calculation of hue and value**

To calculate the value we first calculated the distance between the camera and the middle of the top cube plane in 3D space. We do so by first transforming the 3D world coordinates to 2D camera coordinates using the extrinsic parameters we got using *solvePnP*. This was done using the formula , where R is the rotation matrix, t is the translation vector and c is the camera coordinate system and w the world coordinates. By taking the Euclidean distance of this point we obtained the distance between the camera and the point. This distance value was then scaled between 0 and 4 meters over the possible values that value can take (all integer values between 255 and 0), if the distance exceeds 4 meters then a value of 0 is chosen. We also display the distance on the image itself, along with a dot where the point is.

Sadly, we later discovered that a windows webcam can have an auto focus option turned on which might have influenced these results (as they were not obtained with OpenCV, but with the internal camera app). This could affect our measurement accuracy particularly the distances, as they may not have been all that consistent throughout the calibration. We found out at a later stage of the process and decided not to change this. We still believe the steps we performed are correct and reflect our efforts.

Second, the colour hue was supposed to be calculated as a change in angle. When discussing with Ronald, he told us it did not really matter along which axis the rotation was measured. We first, again, got the camera extrinsics using *solvePnP*. Rotation matrix R contains the rotation information. Using the 2nd arctan, we extracted the angle of chessboard tilt in radians: . We then converted this to degrees and scaled it between 0 and 45 over the possible values that hue can take (all integer values between 255 and 0)[[1]](#footnote-1), if the angle exceeds 45 degrees then a value of 0 is chosen.

**Choice tasks**

**Choice task 1**

Using OpenCV’s *VideoCapture*, we also made our code able to handle video inputs. Be that camera or just random video files. The results can be found in “*example\_video.mp4*”.

**Choice task 2**

For this task, we first put all our 25 images through the *calibrateCamera* function. This function returns a re-projection error. For all 25 images this was 3.522, which was used as a baseline. Then, step by step, we took out individual images and calibrated the other 24 together. This also gave an error. If the calibration was worse than before, this image was labelled as rejected. After testing each individual image, we left out all images that negatively impacted the error and calibrated again. The new calibration gave an error of only 1.057. Out of 25 images, 10 negatively impacted the error and were removed. Three of these images were manually calibrated. The resulting matrix K looks as follows:

Note that this matrix most closely resembles run 3.

**Choice task 5**

This task took by far the most time, as we apparently have really challenging images. For example, one of them is so skewed and far away that each square is only about 20 by 4 pixels. We attempted a lot of image enhancements, starting with blurring, to smooth edges and avoid gaps (Functions like *medianBlur, GaussianBlur,* and *AddWeighted*). Sadly, none of these made the 5 images detectable. Then we tried gamma correction using *LUT*, which did not help either. We also tried *createCLAHE* and *filter2D* to try and manipulate the images in our favour. Lastly we tried *findChessboardCornersSB* instead of *findChessboardCorners*, which is a more exhaustive method. Sadly, all of this, except the new *findChessboardCornersSB* did not end up being adopted into the final code.

It is important to note that we also tried the combinations of all these techniques, including in all different orders. Sometimes a combination led to 1 of 5 images being corrected, but never more.

Until we added the final filter, which was thresholding. All pixels above and below values of 200 were pushed to their extremes (0 and 255, as seen from a grey scaled image). This yielded the best results, with now being able to automatically detect 2 out of 5 images. Sadly, all other filters were removed and are no longer findable in the current version of the code. If evidence is needed that they did at some point, please contact us and we will show the Git logs.

**Choice task 6**

For this task we run all images through the calibration and for each, extract the R and t matrices/vectors. The coordinate of the camera is then calculated by . And the direction the camera looks at is the reverse of the camera coordinates, as it points towards (0,0,0). We also plotted the outline of the corners of the chessboard in black lines. We did not use external code for this part, as we were already familiar with Matplotlib. The resulting plot is as follows:

Afbeelding met diagram, lijn, ontwerp

Door AI gegenereerde inhoud is mogelijk onjuist.

**Any GenAI prompt used**

**Tool**: ChatGPT  
**Prompt**: I'm calibrating a camera on chessboard corners and this is the result I'm getting from opencv's calibratecamera function, does this make sense? ret = 2.8003920460223046 mtx = array([[949.3544538 , 0. , 632.29329938], [ 0. , 950.25521848, 342.30391943], [ 0. , 0. , 1. ]]) dist = array([[ 0.01308269, 0.00830235, -0.00569856, -0.00197573, -0.07354867]]) rvecs = (array([[-0.02249589], [ 0.96791941], [ 2.95791207]]), array([[-0.28472393], [-0.36600851], [ 0.097076 ]]), array([[ 0.08090293], [ 0.23423288], [-0.04143436]]), array([[-0.34284721], [-0.78286334], [-2.5990608 ]]), array([[0.01436033], [1.49111563], [2.69694279]]), array([[-1.2020515 ], [-0.73456981], [-2.74927711]]), array([[-1.32214319], [-0.71582873], [-2.72065512]]), array([[-0.01042112], [-0.73417606], [-3.08282249]]), array([[-0.77557792], [-0.96354568], [-2.76967579]]), array([[0.04458953], [1.99562545], [2.37763468]]), array([[ 0.69954609], [-1.33751776], [-2.15106206]]), array([[ 0.04737814], [-0.96547143], [-2.76776792]]), array([[0.1911455 ], [1.31245308], [2.33740533]]), array([[ 0.11651925], [-1.50941256], [ 2.70418832]]), array([[ 0.16061017], [-1.07070278], [ 2.86438501]]), array([[0.78830349], [0.22170946], [2.92119716]]), array([[ 0.17211444], [-0.79030873], [ 2.95848013]]), array([[-0.20686103], [-1.49680647], [ 2.62704821]]), array([[-0.05175465], [-2.27179329], [-2.13559935]]), array([[1.86619636], [0.60984159], [2.29089539]]), array([[ 0.53370667], [-0.7484461 ], [ 2.6418231 ]]), array([[ 0.95266271], [-0.73844584], [-2.83864623]]), array([[-0.12116627], [-1.8231788 ], [-2.30826317]]), array([[0.12367965], [0.86430081], [2.93267143]]), array([[0.62077885], [1.26342586], [2.66514551]])) tvecs = (array([[ 2.31422848], [18.58075295], [61.13502679]]), array([[-4.16399451], [ 0.46782801], [55.49457131]]), array([[ 7.42302111], [-7.18078668], [62.37929283]]), array([[-24.16418994], [ -8.798595 ], [ 49.18685365]]), array([[ 2.82310537], [15.141083 ], [43.55466905]]), array([[17.42168171], [13.1790839 ], [43.74098382]]), array([[-3.46916923], [ 8.37366471], [28.06389763]]), array([[ 4.96076143], [-9.35237871], [66.37138906]]), array([[ 5.49067029], [ 3.41465461], [20.36913679]]), array([[-1.26075831], [ 4.16268067], [24.09055228]]), array([[ 6.10767187], [ 9.82234796], [33.85335488]]), array([[ 1.30978531], [ 3.1476454 ], [15.1738025 ]]), array([[ 6.97026186], [-1.06420473], [13.62421369]]), array([[ 5.7795443 ], [ 6.31574762], [58.04228663]]), array([[ 5.75050526], [ 3.99872441], [19.42359176]]), array([[10.94872743], [10.11298055], [58.93036659]]), array([[10.25449097], [ 8.31309871], [60.94987097]]), array([[-15.37630562], [ -9.69711889], [ 42.8500144 ]]), array([[ 3.95718153], [ 4.63123237], [13.93785951]]), array([[10.16318979], [ 5.46580038], [18.11169343]]), array([[ 4.19519187], [-4.73082655], [32.23472649]]), array([[-13.42320391], [ 0.42063172], [ 55.8171702 ]]), array([[-6.60475197], [ 9.16851895], [35.37318121]]), array([[ 4.26293134], [-4.653807 ], [46.45000252]]), array([[-3.68845549], [11.39562137], [37.51759368]]))

**Use:** This output was used to sanity check if the camera calibration results were correct. We learned that the re-projection error was quite high, which helped us in choice task 2

**Tool**: ChatGPT

**Prompt**: If I've calibrated my camera using opencv and now able to project points on a grid of a chess board what is the theory behind finding the real world distance between a point and the camera? Let's say the coordinates on the grid are (1, 1, 2)

**Use**: ChatGPT helped in explaining the theory from the lecture, being able to explain the maths more clearly so we could better understand what was going on.  
  
**Tool**: ChatGPT  
**Prompt**: My rotation vector is a 3x1 vector, did I do something wrong?

**Use:** We expected R to be 3x3, yet it gives 3x1 vectors. GPT explained why this was and the idea of the steps we could take to fix it.

1. We only found out just before the deadline that the assignment had been changed from 255 to 180. [↑](#footnote-ref-1)