# [Fall 2022] ROB-GY 6203 Robot Perception Homework 1

Rushi Bhavesh Shah

Submission Deadline (No late submission): NYC Time 11:00 AM, October 05, 2022
Submission URL (must use your NYU account): https://forms.gle/S5G4XqsbVEEKrVVDA

1. Please submit the **.pdf** generated by this LaTex file. This .pdf file will be the main document for us to grade your homework. If you wrote any code, please zip all the **code** together and **submit a single .zip file**. Name the code scripts clearly or/and make explicit reference in your written answers. Do NOT submit very large data files along with your code!

2. You don't have to use AprilTag for this homework. You can use OpenCV's Aruco tag if you are more familiar with them.

3. You don't have to physically print out a tag. Put them on some screen like your phone or iPad would work most of the time. Make sure the background of the tag is white. In my experience a tag on a black background is harder to detect.

4. Please typeset your report in LaTex/Overleaf. Learn how to use LaTex/Overleaf before HW deadline, it is easy because we have created this template for you! **Do NOT submit a hand-written report!** If you do, it will be rejected from grading.

5. Do not forget to update the variables "yourName" and "yourNetID".

## Contents

# Task 1 Sherlock's Message (2pt)

Detective Sherlock left a message for his assistant Dr. Watson while tracking his arch-enemy Professor Moriarty. Could you help Dr. Watson decode this message? The original image itself can be found in the data folder of the overleaf project (https://www.overleaf.com/read/vqxqpvbftyjf), named `for_watson.png`

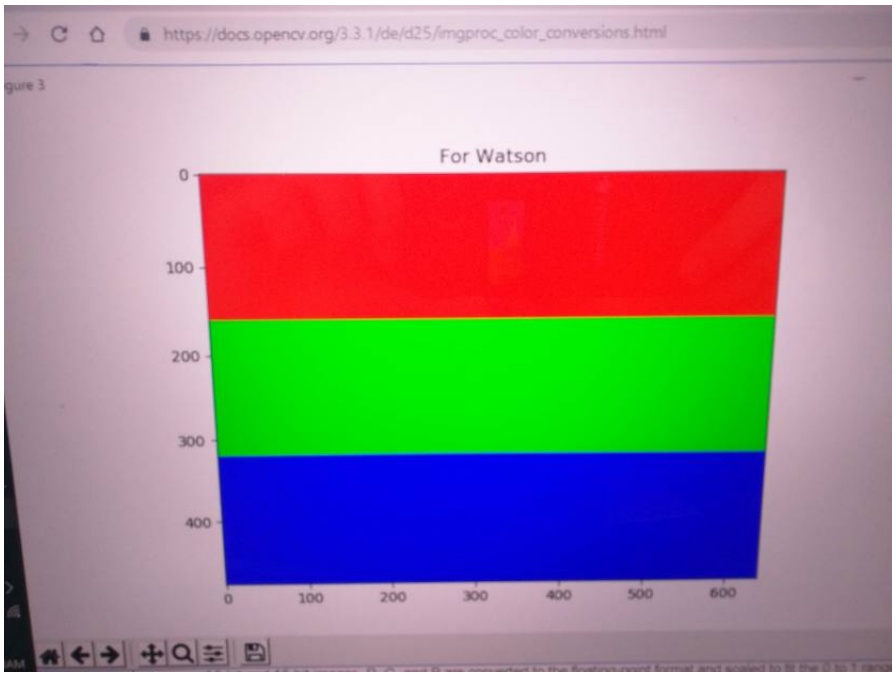
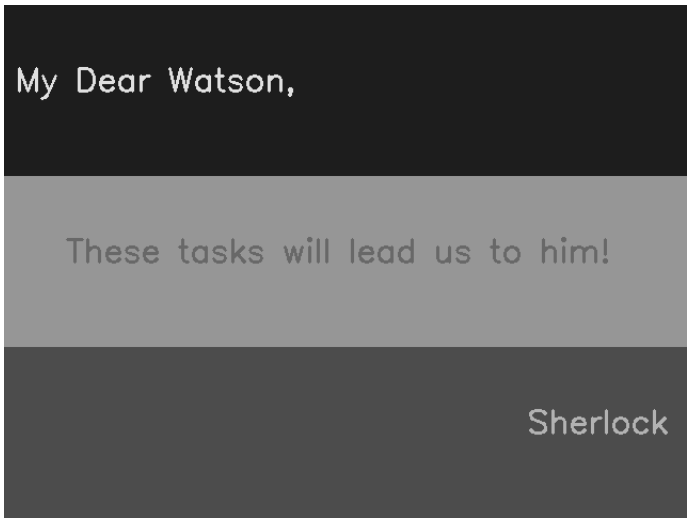



Figure 1: The Secret Message Left by Detective Sherlock

## a) (1pt)

Please submit the image(s) after decoding. The image(s) should have the secret message on it(them). Screenshots or images saved by OpenCV is fine.

**Answers:**



## b) (1pt)

Please describe what you did with the image with words, and tell us where to find the code you wrote for this question.

**Answers:**

After reading the image, I converted it to a Gray Scale Image from RGB image. Later on, I used Numpy's unique function to get the unique points or pixels from the image since those were the pixels where the message was encoded. Later on, I applied 'thresholding' to all those pixels and converted them to white color. Since I wanted to do it for all threes sections of the image (i.e. R, G and B) I did 'bit-wise xor' operation after applying 'thresholding'.

You can find the code in '/HW1/Q1/Q1.py'

# Task 2 Vanishing Points (4pt)

Take an image with your camera so that in the image you can find 3 orthogonal directions, i.e., 3 vanishing points.

## a) (2pt)

**Identify**, by hand or by code, those three vanishing points. **Calculate or estimate** their coordinates and **visualize** them by providing pictures.

**Answers:**


I found the vanishing lines and their intersection points by following the methods mentioned below:
(i) Captured an image where line features can be observed in all three directions.
(ii) Performed a Gaussian blur to remove noise, and assist the 'canny edge detector' to detect the edges in the image.
(iii) A list of line present in the image is obtained by using the 'HoughLinesP' function and out of these, three pairs of lines are selected, each pair being orthogonal to the other two.
(iv) The coordinates of end points of lines are obtained by using HoughLinesP' function and are represented as ax + by + c = 0..
(v) The intersection points of each pair of lines are obtained by doing a cross product of the lines, forming a pair.

In the figure above, there are three pairs of mutually orthogonal lines which are parallel in real world but tend to

intersect in the camera frame. The coordinates for intersection of these lines are :
(i) for lines 9 and 17 coordinates are (3961, 1479)
(ii) for lines 246 and 304 coordinates are (-2794, 1193)
(iii) for lines 6 and 151 coordinates are (1608, 17995)

## b) (2pt)

Use the information you gathered in a) to **calibrate** your camera. **Document** your process and provide us with the focal length f, and principal point position x0 and y0

**Answers:**
To calibrate the camera using the intersection points, I followed the below mentioned steps:
(i) After obtaining the vanishing points from the previous part, I converted all the coordinates in homogeneous coordinates.
(ii) All the homogeneous coordinates are stored in a (3x4) matrix called which comes from the equation $\mathbf{Ax} = \mathbf{b}$.
(iii) Further, SVD is performed on A.
(iv) A positive definite matrix is formed using the last row of $\mathbf{v}_h matrix.$
$(v) Further, Cholesky Decomposition is performed on the matrix to obtain the inverse of transpose of the camera calibration matrix.$
$(vi) Then, the output from the previous step undergoes inverse and transpose operations to provide K matrix, in which each element is d$

**I got following Camera Calibration matrix:** $\begin{bmatrix} 3.1014e+03 & 0.0000 & 1.2371e+03 \\ 0.0000 & 3.0914e+03 & 2.0432e+03 \\ 0.0000 & 0.00000 & 1.00000e+03 \end{bmatrix}$

**Hence the focal length is 4386.06204 and the principle point coordinates** $(\mathbf{x}_0, y_0) are$
$(1.2371e+03, 2.0432e+03)$.

# Task 3 Camera Calibration (4pt)

## a) (2pt)

Use the pyAprilTag package provided in the class, or other free packages (e.g., OpenCV's camera calibration toolkit) that you may be aware of, to **calibrate** your camera and provide the full K matrix, with the top two distortion parameters k1 and k2.

**Answers:**
First a 5x6 board of Aruco Markers is generated with length 2 cm and distance between two as 0.8 8 cm. Further all the markers are identified using **aruco.detectMarkers** function. A list of corners and marker-ids is stored. And further **aruco.calibrateCameraAruco** function is used to find the Camera calibration matrix and the distortion matrix.

The Camera Calibration matrix obtained is : $\begin{bmatrix} 3.1038e+03 & 0.0000 & 1.5686e+03 \\ 0.0000 & 3.0936e+03 & 2.1079e+03 \\ 0.0000 & 0.00000 & 1.00000e+03 \end{bmatrix}$

$and the values K_1 = 0.14506076 and k_2 = -0.59302787.$

## b) (2pt)

**Compare** this calibration result with the one you obtain above from using vanishing points and discuss the pros and cons of the two methods.

**Answers:**
On comparing both the camera calibration matrices, we can see that the values of fx , fy , x0 , and y0 are almost similar. However, it is better to use Zhang's method for camera calibration since it provides multiple testing conditions to the algorithm to calibrate the camera, and the world parameters are well-defined to compare the values obtained by the camera in each iteration. On the other hand, which using the vanishing lines method, we need to make sure that the image at at least two pairs of orthogonal lines, which may not be sufficient to calibrate the camera. Even knowing third pair may improve the result but still may not be accurate since the world parameters are not known.

# Task 4 Tag-based Augmented Reality (5pt)

Use the pyAprilTag package to detect an AprilTag in an image (or use OpenCV for an Aruco Tag), for which you should take a photo of a tag. Use the K matrix you obtained above, to draw a 3D cube of the same size of the tag on the image, as if this virtual cube really is on top of the tag. **Document** the methods you use, and **show** your AR results from at least two different perspectives.

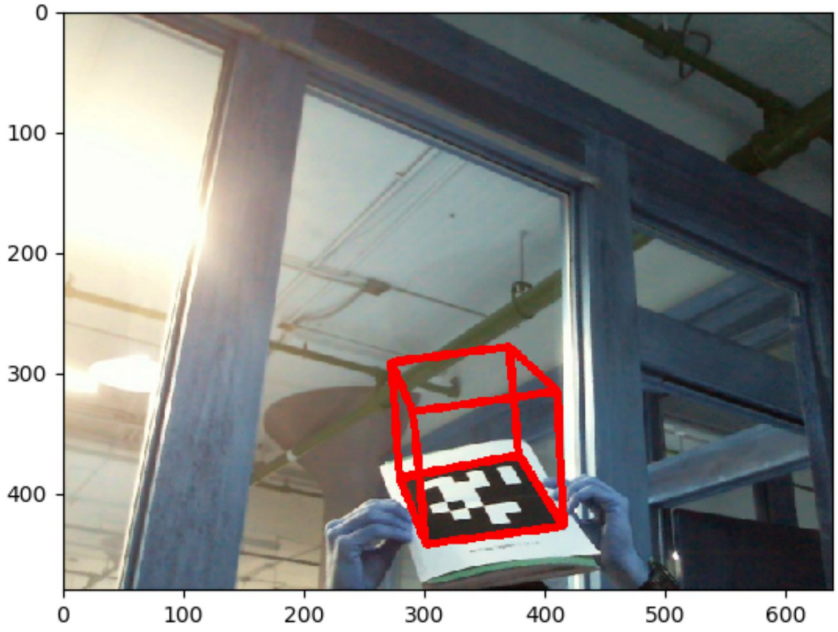



Figure 2: Caption

**Tips**: There are many ways to do this, but you may find OpenCV's `projectPoints`, `drawContours`, `addWeighted` and `line` methods useful. You don't have to use all these methods.

**Answers:**
(i) Aruco markers are detected in the images using the method defined in the previous task. Also the Calibration matrix is obtained from there.
(ii) Further, dimensions of the aruco Marker are obtained in the world frame by manual measurement.
(iii) Further, coordinate axis is defined for the marker.
(iv) The transformation of the marker is obtained in the camera frame.
(v) Points are projected around the marker using "projectPoints" function, using which the contour needs to be plotted.
(vi) Contours are drawn around the marker using "drawContours" function.
(vii) Further "line" function is used to join the points and form a cube around the marker.

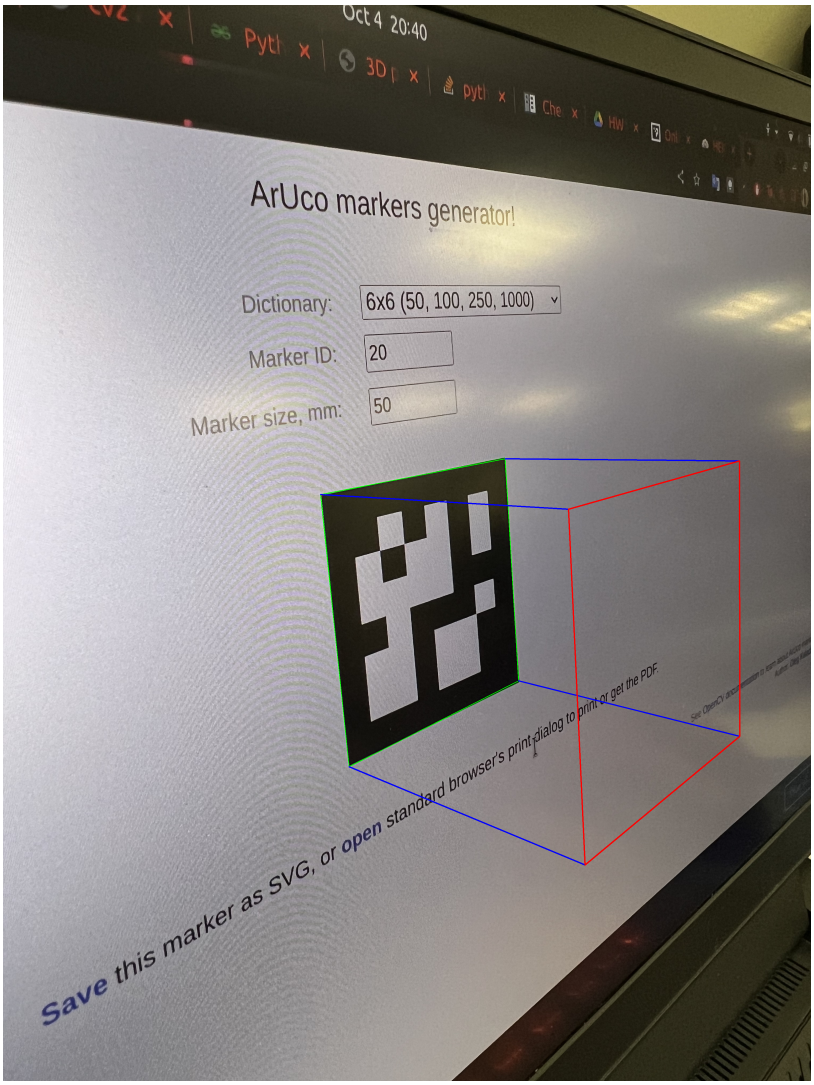



Figure 3: Contoured Image