ICBV HW #1

Dor Litvak Eytan Ivan Reuven

Question 1 – Perspective and orthographic projections

Section A

Camera A parameters: $f = 1$ and $O_1 = (0, 0, 0)$
The camera origin and axis coordinate system are the same as the polygon plane, so We do not need to add rotation or translation. Using the perspective projection homogeneous coordinates representation we learn in the lecture:

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} -\frac{f}{z} \cdot x \\ -\frac{f}{z} \cdot y \end{pmatrix} = \begin{bmatrix} -f \cdot x \\ -f \cdot y \\ z \end{bmatrix} = \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

For each point $p_k = (x_k, y_k, z_k) \in P_w$ the polygon, we will find the homogeneous representation of the point $p_k = \begin{bmatrix} x_k \\ y_k \\ z_k \\ 1 \end{bmatrix}$. And use the above matrix multiplication to find the correspond points in the cameras' image planes.

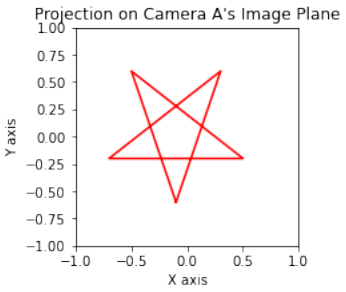

Figure 1: Polygon's perspective projection on Camera A plane.

The image points P_i are: (0.10,0.60),(0.30,0.60),(0.70,0.20),(0.50,0.20),(0.50,0.60)

Section B

Camera B parameters: $f = 1$ and $O_2 = (-5, 0, 4)$
In order to get from the camera coordinate system to the word' coordinates system we need to rotate $-90°$ in the Y axis (X axis of the camera coordinate system) and then $-90°$ in the Z axis.

3X3 Rotation matrix:

Rotation in the Y axis is define using the following matrix: $\begin{bmatrix} cos(\theta_1) & 0 & sin(\theta_1) \\ 0 & 1 & 0 \\ -sin(\theta_1) & 0 & cos(\theta_1) \end{bmatrix}$

Set $\theta_1 = -90°$:

$$Rot_Y = \begin{bmatrix} cos(-90) & 0 & sin(-90) \\ 0 & 1 & 0 \\ -sin(-90) & 0 & cos(-90) \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Rotation in the Z axis is define using the following matrix: $\begin{bmatrix} cos(\theta_2) & -sin(\theta_2) & 0 \\ sin(\theta_2) & cos(\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Set $\theta_2 = -90°$, the rotation matrix is:

$$Rot_Z = \begin{bmatrix} cos(-90) & -sin(-90) & 0 \\ sin(-90) & cos(-90) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The total rotation matrix is combination of the two matrices, the first rotation is one the right and the second on the left.

$$Rotation = Rot_Z \cdot Rot_Y = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

3X1 Translation matrix:

Since we are now taking in different coordinate system (the camera coordinate system) we also need to change the coordinate of the camera's location. Let's test what is the difference between the global coordinate system to the camera's coordinate system:

$X_{Camera} = Y_{Global}$

$Y_{Camera} = Z_{Global}$

$Z_{Camera} = X_{Global}$

This means we need to change $O_{2Global} = (-5, 0, 4)$ to be $O_{2Camera} = (0, 4, -5)$.

We also need to realize that we are looking the translation vector from the camera location to the origin location (0,0,0). This is the opposite vector of $O_{2Camera}$.

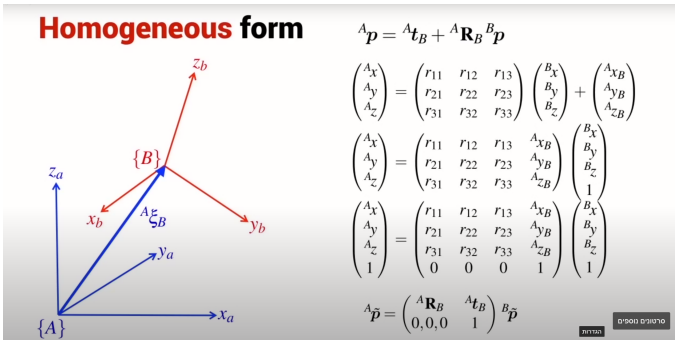

Figure 2: As we wish to represent vector from B coordinate system in A coordinate system we find the Rotation between the A and B systems and them add it the vector from A system to B system, in our case A is the camera and B is the Global world.

$$Translation = \begin{bmatrix} 0 \\ -4 \\ 5 \end{bmatrix}$$

$$\textit{Total Euclidean Transformation (Homogeneous form):}\ \mathbf{F} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -4 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\textit{The K matrix is the same as for camera A (f is known):}\ K = \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Using the perspective projection homogeneous coordinates representation we learn in the lecture:

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} -f \cdot \frac{y}{x+5} \\ f \cdot \frac{-z+4}{x+5} \end{pmatrix} = \begin{bmatrix} -fy \\ f(-z+4) \\ x+5 \end{bmatrix} = \begin{bmatrix} 0 & -f & 0 & 0 \\ 0 & 0 & -f & 4f \\ 1 & 0 & 0 & 5 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -4 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = K \cdot F \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

For each point $p_k = (x_k, y_k, z_k) \in P_w$ the polygon, we will find the homogeneous representation of the point $p_k = \begin{bmatrix} x_k \\ y_k \\ z_k \\ 1 \end{bmatrix}$. Assign in the question we got from the matrix multiplication and get the camera plan projection point $p_i = \begin{pmatrix} -f \cdot \frac{y_i}{x_i+5} \\ f \cdot \frac{-z_i+4}{x_i+5} \end{pmatrix}$

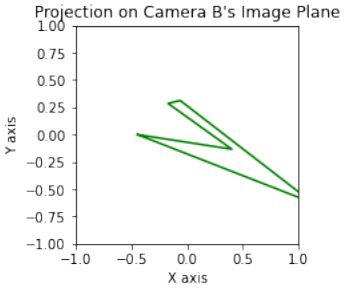

Figure 3: Polygon's perspective projection on Camera B plane.

Section C

For the orthographic projection on the B camera, we will use the same Euclidean Transformation matrix (F matrix - extrinsic parameters) and the K matrix correspond with the orthographic projection which is the identity matrix.

Total Euclidean Transformation matrix (from section B): $F = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -4 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

The K matrix for orthographic projection: $K = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Using the perspective projection homogeneous coordinates representation we learn in the lecture:

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} y \\ z-4 \end{pmatrix} = \begin{bmatrix} y \\ z-4 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -4 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = K \cdot F \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

For each point $p_k = (x_k, y_k, z_k) \in P_w$ the polygon, we will find the homogeneous representation of the point $p_k = \begin{bmatrix} x_k \\ y_k \\ z_k \\ 1 \end{bmatrix}$. Assign in the question we got from the matrix multiplication and get the camera plan projection point $p_i = \begin{pmatrix} y_i \\ z_i - 4 \end{pmatrix}$

Question 2 – Camera calibration

Section A - Chessboard corners

In this section we took 20 pictures of chess board using our cellphone camera (images size (1200, 1600, 3)) and used OpenCV methods to find the chess board corners and to draw them.

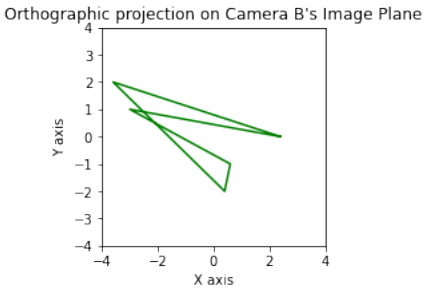

Figure 4: Polygon's orthographic projection on Camera B plane.

For each chess board picture we took:
We convert it to grey-scale for better color contrast.
Used *findChessboardCorners* OpenCV function which takes image and pattern size and returns the corners coordinates and ret value - which is true if the number of corners found match with the pattern size (we used **(9,6)**).
We also tests an option to use the *cornerSubPix* OpenCV function that find the sub-pixel accurate location of corners using the image gradient but the results were almost identical.
Using the corners returned from the *findChessboardCorners* function we used the OpenCV method *drawChessboardCorners* to draw the corners over the RGB image.
The results of two images are shown here.

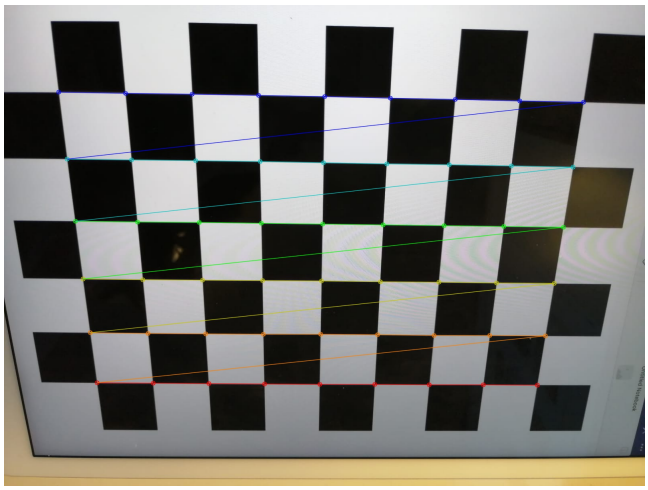

Figure 5: Chess board corners draw over in image No.1.

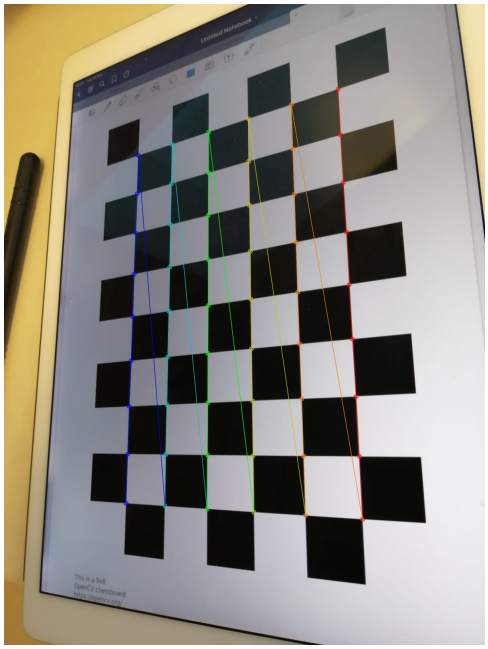

Figure 6: Chess board corners draw over in image No.2.

Section B - Create matching 3D points

As shown in the practical session, we created a set of 3D points matching with our board dimension and square size **(9,6)**, the world points vector is **(54, 3)** size:
[[0.0.0.][1.0.0.][2.0.0.][3.0.0.][4.0.0.][5.0.0.][6.0.0.][7.0.0.][8.0.0.]
[0.1.0.][1.1.0.][2.1.0.][3.1.0.][4.1.0.][5.1.0.][6.1.0.][7.1.0.][8.1.0.]
[0.2.0.][1.2.0.][2.2.0.][3.2.0.][4.2.0.][5.2.0.][6.2.0.][7.2.0.][8.2.0.]
[0.3.0.][1.3.0.][2.3.0.][3.3.0.][4.3.0.][5.3.0.][6.3.0.][7.3.0.][8.3.0.]
[0.4.0.][1.4.0.][2.4.0.][3.4.0.][4.4.0.][5.4.0.][6.4.0.][7.4.0.][8.4.0.]
[0.5.0.][1.5.0.][2.5.0.][3.5.0.][4.5.0.][5.5.0.][6.5.0.][7.5.0.][8.5.0.]]

Section C - Perform the calibration

We preformed the camera calibration using OpenCV *calibrateCamera* method.
The function get as an input: object points, image points, image size, camera matrix and distortion coefficients.
As an input we gave the function the following values: object points = world points, image points = image corners, image size = image shape, and camera matrix and distortion coefficients = None since we do not know their values, we can use approximating methods to get an approximation of them, but the results tern good as well as with using None.
The function returns 4 values as an output:

1. cameraMatrix - 3x3 calibration matrix of the camera, contains the camera intrinsic parameters, the K matrix in the equation $M = K \cdot F$. The intrinsic parameters include the focal length, the optical center, also known as the principal point, and the skew coefficient. The camera intrinsic matrix, K, we got from the *calibrateCamera* method is:

$$\text{cameraMatrix} = \mathbf{K} = \begin{bmatrix} 1.36532907e+03 & 0.00000000e+00 & 6.98753332e+02 \\ 0.00000000e+00 & 1.36090556e+03 & 6.71768122e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

2. distCoeffs - distortion coefficients, The camera matrix does not account for lens distortion because an ideal pinhole camera does not have a lens. To accurately represent a real camera, the camera model includes the radial and tangential lens distortion.
   Radial Distortion - Radial distortion occurs when light rays bend more near the edges of a lens than they do at its optical center. The smaller the lens, the greater the distortion. The presence of the radial distortion manifests in form of the "barrel" or "fish-eye" effect.
   For the radial factor we use the following formula: $r = x^2 + y^2$
   $x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$ $y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$
   Tangential Distortion - Tangential distortion coefficients of the lens. Tangential distortion occurs when the lens and the image plane are not perfectly parallel.
   It can be represented via the formulas: $r = x^2 + y^2$
   $x_{distorted} = x + [2p_1xy + p_2(r^2 + 2x^2)]$ $y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$

$$\text{distCoeffs} = \begin{bmatrix} k_1 \\ k_2 \\ p_1 \\ p_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} -0.12055615 \\ 0.23675179 \\ -0.00462749 \\ -0.00510707 \\ -0.32732872 \end{bmatrix}$$

3. rvec - Output vector of rotation vectors (Rodrigues) estimated for each object points and image points. The rotation vectors for pairs of world points and corners points, in order to bring the world point to be in the same coordinate system as the camera axis coordinate system using circular movement around the origin.
   That is, each i-th rotation vector together with the corresponding i-th translation vector brings the extrinsic camera matrix (**F**) uses to change the object coordinate space (world points) to the camera coordinate space (image corners).
   Rotation vector for the first image: $\begin{bmatrix} 0.43137359 \\ 0.04464785 \\ -0.02535632 \end{bmatrix}$

4. tvec - Output vector of translation vectors estimated for each object points and image points. The translation vectors for pairs of world points and corners points. The translation vectors for pairs of world points and corners points, in order to bring the world point to be in the same coordinate system as the camera coordinate system using a vector from the camera coordinate system to the origin of the world coordinate system.
   Translation vector for the first image: $\begin{bmatrix} -3.36752405 \\ -2.33730697 \\ 7.86649577 \end{bmatrix}$

Section D - Camera Matrix

1. The cameraMatrix output parameter of the OpenCV *calibrateCamera* refer to the **K** matrix in the formula. The K matrix in the formula is the calibration matrix of the camera, contains the camera intrinsic parameters, same as in the OpenCV method.

2. To build for each chess board image the extrinsic transformation matrix **F**. We are using the rvec and tvec we got from the prev C section. We first changed the rotation vector to be a rotation matrix using OpenCV *Rodrigues* function.

For each image: $\text{rvec}_i = \begin{bmatrix} R_{00} & R_{01} & R_{02} \\ R_{10} & R_{11} & R_{12} \\ R_{20} & R_{21} & R_{22} \end{bmatrix}$ $\text{tvec}_i = \begin{bmatrix} T_0 \\ T_1 \\ T_2 \end{bmatrix}$

Using both rvec_i, tvec_i we can know build the extrinsic transformation matrix **F**. We are using homogeneous representation so we are adding 4th line with zeros and one in the end.

$$F = \begin{bmatrix} R_{00} & R_{01} & R_{02} & T_0 \\ R_{10} & R_{11} & R_{12} & T_1 \\ R_{20} & R_{21} & R_{22} & T_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Using the camera intrinsic parameters matrix, cameraMatrix, **K** and the camera extrinsic parameters matrices **F** we found in section D2. WE now have all the Camera Calibration Parameters and we can calculate using the following formula the full camera matrix. Because we are using homogeneous representation we need to add another zeros column to the K matrix.
   For each images the M matrix is:

$$\mathbf{M} = \mathbf{K} \cdot \mathbf{F} = \begin{bmatrix} 1.36532907e+03 & 0.00000000e+00 & 6.98753332e+02 & 0 \\ 0.00000000e+00 & 1.36090556e+03 & 6.71768122e+02 & 0 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 & 0 \end{bmatrix} \cdot \begin{bmatrix} R_{00} & R_{01} & R_{02} & T_0 \\ R_{10} & R_{11} & R_{12} & T_1 \\ R_{20} & R_{21} & R_{22} & T_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Section E

In this section we project the world points we defined in Section B on top of one of images No.10 from the calibration process. The original image No.10 of chess board is presented in the following figure.

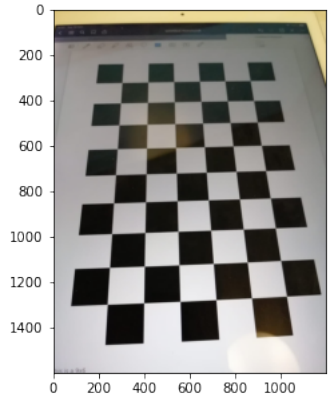

Figure 7: Image number 10.

The original chessboard corners found in section A and plotted using *drawChessboardCorners* OpenCV method are presented in the next figure.

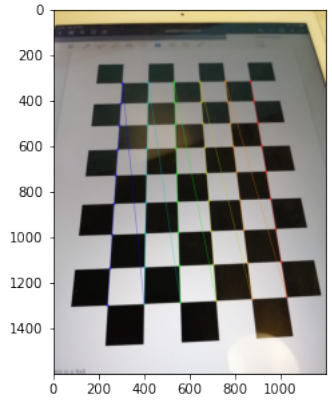

Figure 8: Chess board corner results for image number 10.

Plotting of the projection of the world points defined in section B on top of image No.10 using the camera matrix we got from the calibration process $\mathbf{K}$, and using the calculation we preform in Section D3 $\mathbf{M} = \mathbf{K} \cdot \mathbf{F}$ is presented in the following figure.

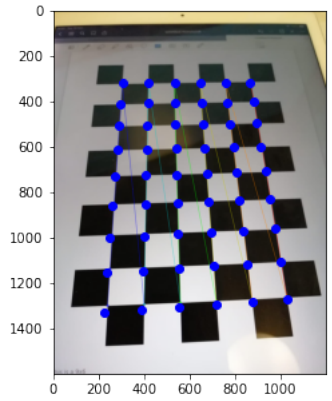

Figure 9: Image No.10 world points projection result using the M matrix.

Another Plotting we present is the world points projection on the image plane using the OpenCV method *projectPoints*, also over image No.10 for comparison.

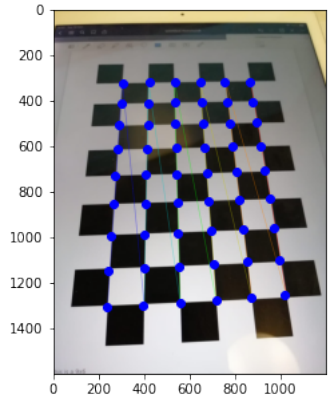

Figure 10: Image No.10 world points projection result using the OpenCV method.

Is there a difference? Yes. As you can see the M matrix projection is not so accurate as the OpenCV method projection (The down-left corner). Why? The OpenCV method considerate in one more important value our M matrix calculation doesn't. It is the distortion coefficients. The camera matrix does not account for lens distortion because an ideal pinhole camera does not have a lens. But of course our phone camera does have a lens. The OpenCV method take into account the Radial Distortion and the Tangential Distortion so it is much more accurate when we are taking about images taking from a lens.

Section F

This time we choose image No.9 because of its angle, it would be more convenient to see 3D objects presented on it as a cube. The image is presented in the following figure.

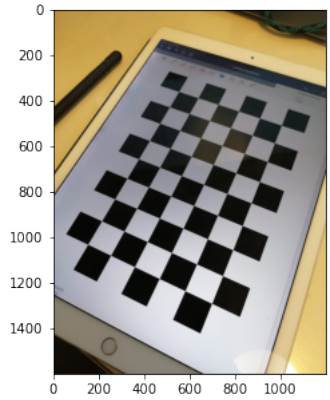

Figure 11: Image No.9 for cube drawing.

We first found image plan cube axis and draw them over the image on the left upper black square. Red is the X axis, Green is the Y axis a,d Blue is the Z axis.

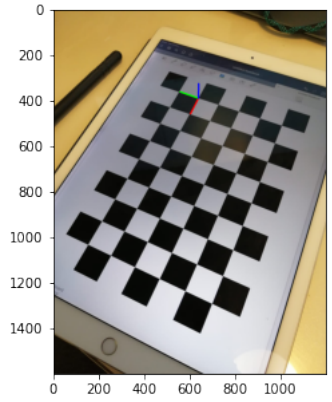

Figure 12: Image No.9 cube axis.

Now it is much more convenient to draw the full cube on the left upper black square of the image. The cube presented in the next figure.

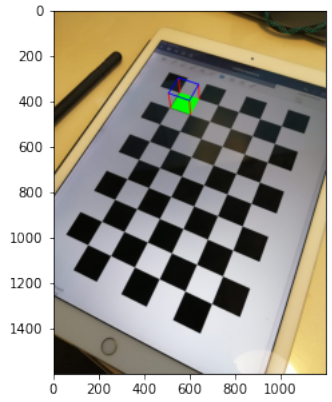

Figure 13: Image No.9 cube plot.

We choose to rotated the cube 30° around the Y axis. The result shown in the following figure.

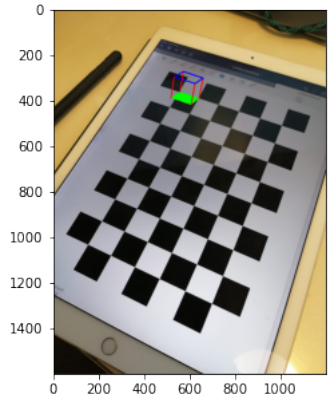

Figure 14: Image No.9 cube 30° in Y axis.

Section G

Yes, there is a problem with the student's approach. Since he took multiple images from multiple sources on the internet, and the images probably didn't took from the same camera, they doesn't share the same camera calibration parameter. Specifically, the camera intrinsic parameters and distortion coefficients. The student wouldn't be able to match camera parameters and distortion coefficients using the camera calibration OpenCV method since there aren't such camera that would match to all the image taking from different camera's. (He will get a result but not so good one). And the results certainly would not represent a real camera.

Question 3 – Reading Material

Section A

The dilemma that evolution had to solve during the development of the eye is the approach in which the light(photons) will enter into the eye. The pinhole approach suggests that the light will enter into the eye via only one small hole("almost infinity", as Dawkins says). The advantage of this approach is that the image is sharp("sharpish", as Dawkins says) but at the cost of a dark image. The wide approach suggests that the light will not be constrained to one pinhole, and let more than one ray of light that extends from a single point to enter the retina. The advantage of this approach is a brighter image, but at the cost of a fuzzy image. The fuzzy image was created because multiple light rays that extend from a single point project to multiple places in the retina, and the result is projection of "infinite objects" at the retina, as Dawkins says. The solution that combines the two approaches advantages is to open a bigger hole than a pinhole(like the wide approach), but project all the rays that extend from the same origin point to a single location in the retina. The lens makes this happen, by bending the rays.

Section B

In our opinion, computers should emulate human vision. We will present the arguments and examples to support our opinion:

Section 1.2.1 (Visual Illusions, Ambiguities, and Inconsistent) Nalwa suggests that Computer Vision(CV) shouldn't emulate the human vision system and abilities precisely because the human vision is not unequivocal.

We disagree with his claim. Firstly, we believe that each visual interpretation should be acceptable. For example, Figure 1.5 shows the vase/faces image. If someone will see the vase, does it mean that his vision is damaged? And what about someone that would see the faces? Is his vision damaged? We believe that all answers are "no."

The purpose of computer vision is to infer the state of the physical world from the inherently noisy and ambiguous images of the world.
What is infer means? We believe that it is to be as closest possible to the human inference. Therefore, if both vase and faces inference of the scene would be acceptable if we are talking about humans, the same should hold about computers. And as the human vision can give multiple interpretations, we want the computer's vision to infer multiple interpretations. The computer's multi-interpretation vision is useful in emulating the infer the state of the world better.

Another thing that Nalwa says is that we know about only a little bit of what happens in the region of the brain dedicated to vision[page 25].
We think it would probably be a benefit to infer both fields together, and the uncertainty of the human brain can be discovered using computer vision development.
We believe this is exactly the reason why Computer Vision should target to copy precisely the human vision. The two fields can support each other: anything neuroscientists will prove on human brain vision can be applied in computers. Any progress in computer vision can be a hypothesis about brain seeing ability and use neuroscientists.
This collaboration can be beneficial in both fields. The same as neural networks idea took from human vision now helps computers infer much more on the physical world. Nural networks now help analyze seeing brain signals and figure out more about the brain processing stages functions.

Given the purpose of computer vision is to infer the state of the physical world, we would want to make machines see as we do, and even better.

Section C

The 4 main differences between CCD(Charged Couple Device) and CMOS(Complementary Metal Oxide Semiconductor) are:
Rolling Shutter - the rolling shutter feature in CMOS, which means that the light will insert into the camera row-by-row, instead of all together in the CCD. That feature makes CMOS better than CCD for low-light shots, at the cost of some weird patterns in some of the pictures. This unwanted cost can be mitigated by pixel memory or really fast read out.
Parallel vs Sequential Reading - in CCD, after the process of capturing light ends, the camera reads the pixels in sequential order, row-by-row, by shifting the rows towards the capacitor. This reading is very slow and inefficient, and requires a lot of electricity. But, in CMOS, the reading of the pixels happened in parallel. Every pixel has its own capacitor and amplifier, and the result of the amplifier sent to a bus in parallel. This method is faster and more efficient in electricity requirements because there is no need to shift rows.

Hardware Design - obviously, there are several design differences between CCD and CMOS. While both built around silicon semiconductor, in CCD above the silicon layer, there is a layer of stop channels, above that an oxide layer of silicon and at the top there is a charged rows of aluminium layer. Each pixel has stop channels in both of its sides, and 3 aluminium rows - top, middle and bottom in vertically to the stop channels. Unlike CCD, in CMOS the pixels don't have stop channels, and each pixel has its own amplifier and capacitor, unlike the one capacitor and one amplifier in CCD.
Noise Sensitivity - due to the hardware design of the CMOS, especially the fact that every pixel has its own amplifier, makes every noise of every pixel to be amplified. This scenario doesn't happen in CCD, due to the fact that CCD have only one amplifier. As a result, the pictures in CCD are more noise-clean than the pictures in CMOS.

Section D

We will demonstrate two different representations of a simple cup of coffee.
Representations:
We can represent a cup of coffee by the number of coffee beans and the amount of hot water(like in Elite Melt Coffee). The parameters of the number of coffee beans and the amount of hot water are explicit, and we can probably deduce the level of strength by these two parameters. The size of the cup, on the other hand, isn't a factor here and as a result pushed to the background.
We can represent a cup of coffee by the Level of strength and the size of the cup(from a set of small, medium, large, like in a Nespresso machine). The parameters of the Level of strength and the size of the cup are explicit. The number of coffee beans and the amount of hot water, on the other hand, isn't a factor here and as a result pushed to the background.