PART A) Running Calibration images using the luxinous OpenCV code provided. Running the following code.

```python
import numpy as np
import cv2 as cv
import glob


# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob('*.jpg')
mtx=''
for fname in images:
    img = cv.imread(fname)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, (7,6), None)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)
        # Draw and display the corners
        cv.drawChessboardCorners(img, (7,6), corners2, ret)
        #cv.imshow('img', img)
        cv.waitKey(500)
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
print(mtx)
cv.destroyAllWindows()
```

Gives us the following Calibrated Camera Matrix.

```
C:\Users\Osama\depthai>py parta.py
[[536.39560293   0.         334.03199671]
 [  0.         536.43966656 239.43394616]
 [  0.           0.           1.        ]]
```

Part B) Using a picture of the Charuco board. We find the real-world dimensions. We run a python script to give us the real-world dimension. Then we validate the dimensions by cross-checking the python script dimensions with the actual world measured dimensions.

```
C:\Users\Osama\depthai>py partB.py
[[536.39560293   0.         334.03199671]
 [  0.         536.43966656 239.43394616]
 [  0.           0.           1.        ]]
Intrinsinc matrix:  [[536.39560293   0.         334.03199671   0.        ]
 [  0.         536.43966656 239.43394616   0.        ]
 [  0.           0.           1.           1.        ]
 [  0.           0.           0.           1.        ]]
Extrinsinc matrix,  [[ 9.98541160e-01  5.37521637e-02 -4.27596393e-03  3.97631343e+00]
 [-5.37902915e-02  9.98523190e-01  8.12498554e-03  6.21899178e-01]
 [-4.70696973e-03 -7.88312715e-03 -9.99957849e-01 -1.49077740e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
Final matrix:  [[ 5.34040809e+02  2.61992075e+01 -3.36311525e+02 -2.84679648e+03]
 [-2.99822544e+01  5.33759959e+02 -2.35065289e+02 -3.23581577e+03]
 [-4.70696973e-03 -7.88312715e-03 -9.99957849e-01 -1.39077740e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
[[-23.88332738]
 [-16.23329077]
 [-47.6693961 ]
 [  1.        ]]
length along x axis is :  -23.88332737695301
length along y axis is : -16.233290770049727
length along z axis is : -47.66939610463006
```

The script ran was as follows:

partB.py > ...

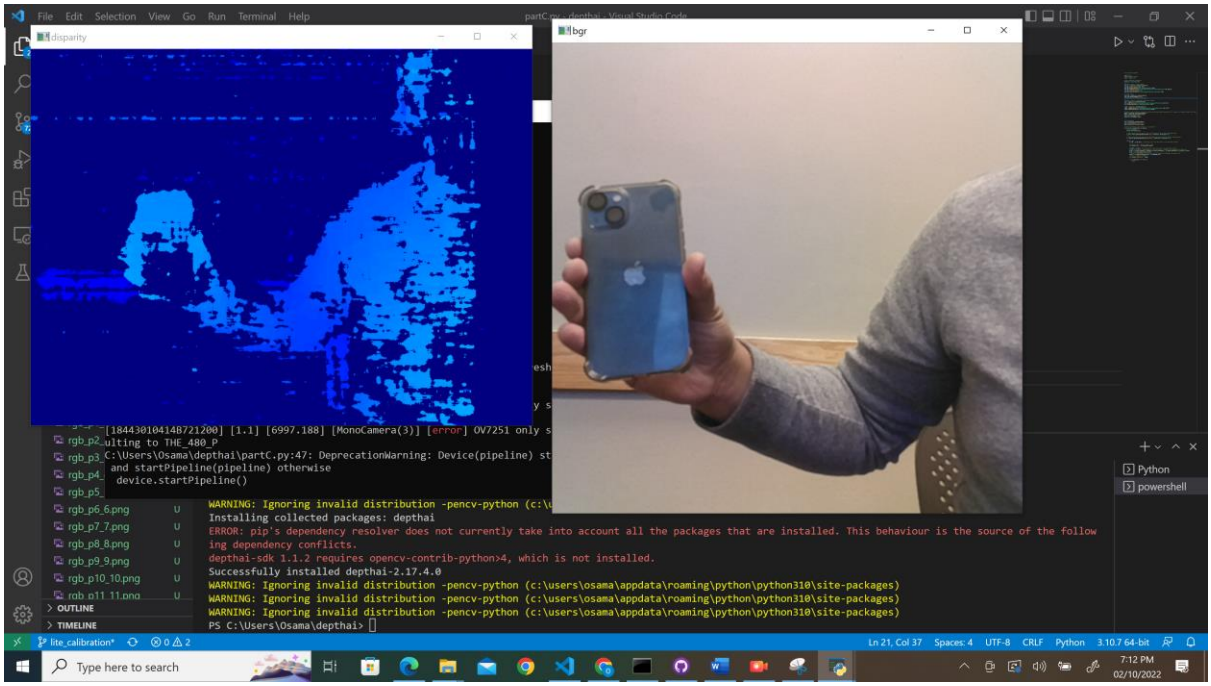
```python
import numpy as np
import cv2 as cv
from scipy.spatial.transform import Rotation
from math import cos, sin, radians
import calib
rvecs = calib.rvecs
tvecs = calib.tvecs

r_obj = Rotation.from_rotvec(np.array(rvecs[0]).reshape(1,3))
rot_matrix = r_obj.as_matrix()

nprvecs = np.array(rvecs)
nptvecs = np.array(tvecs)

## Formation of rotation matrix from rotation vector
# calucation of cos and sin of angle from rotation vector
# since it gives radians we are converting them to degrees
xc, xs = cos(radians(nprvecs[0][0][0])), sin(radians(nprvecs[0][0][0]))
yc, ys = cos(radians(nprvecs[0][1][0])), sin(radians(nprvecs[0][1][0]))
zc, zs = cos(radians(nprvecs[0][2][0])), sin(radians(nprvecs[0][2][0]))

# formation of translation matrix
tx = nptvecs[0][0][0]
ty = nptvecs[0][1][0]
tz = nptvecs[0][2][0]
translation_mtx = np.array([
    [1,0,0,tx],
    [0,1,0,ty],
    [0,0,1,tz],
    [0,0,0,1]
])

#Forming rotation matrix around x, y z axis
rotation_x_matx = np.array([
    [1,0,0,0],
    [0,xc,-xs,0],
    [0,xs,-xc,0],
    [0,0,0,1]
```

partB.py > ...

```
    [0,0,0,1]
])

#Forming rotation matrix around x, y z axis
rotation_x_matx = np.array([
    [1,0,0,0],
    [0,xc,-xs,0],
    [0,xs,-xc,0],
    [0,0,0,1]
])

rotation_y_matx = np.array([
    [yc,0,ys,0],
    [0,1,0,0],
    [-ys,0,yc,0],
    [0,0,0,1]
])

rotation_z_matx = np.array([
    [zc,-zs,0,0],
    [zs,zc,0,0],
    [0,0,1,0],
    [0,0,0,1]
])

extrensic_matx = np.dot(rotation_z_matx, np.dot(rotation_y_matx, np.dot(rotation_x_matx, translation_mtx)))

#Converting intrinsic matrix from 3X3 to 4X4
intrinsic_matx = np.append( np.append(calib.mtx, [[0],[0],[1]], axis=1), [np.array([0,0,0,1])], axis=0)

print(calib.mtx)

#Final camera matrix
camera_matrix = np.dot(intrinsic_matx, extrensic_matx)

print('Intrinsinc matrix: ', intrinsic_matx)

print('Extrinsinc matrix, ', extrensic_matx)
```

```
extrensic_matx = np.dot(rotation_z_matx, np.dot(rotation_y_matx, np.dot(rotation_x_matx, translation_mtx)))

#Converting intrinsic matrix from 3X3 to 4X4
intrinsic_matx = np.append( np.append(calib.mtx, [[0],[0],[1]], axis=1), [np.array([0,0,0,1])], axis=0)

print(calib.mtx)

#Final camera matrix
camera_matrix = np.dot(intrinsic_matx, extrensic_matx)

print('Intrinsinc matrix: ', intrinsic_matx)

print('Extrinsinc matrix, ', extrensic_matx)

print("Final matrix: ", camera_matrix)


cv.destroyAllWindows()


inverse_mat = np.linalg.inv(camera_matrix)

#the dimensions of the image is x = 2, y = 20, z = 15
projection_points = np.array([[5],[21],[34],[1]])

real_world_dim = inverse_mat.dot(projection_points)

print(real_world_dim)

print("length along x axis is : ", real_world_dim[0][0])
print("length along y axis is :", real_world_dim[1][0])
print("length along z axis is :", real_world_dim[2][0])
```

Part C Q3) I ran a Python script to show the depth and the RGB stream.



partC.py > ...

```python
#!/usr/bin/env python3

import cv2
import depthai as dai
import numpy as np

# Start defining a pipeline
pipeline = dai.Pipeline()

#Define a source - color camera
cam_rgb = pipeline.createColorCamera()
cam_rgb.setPreviewSize(600, 600)
cam_rgb.setBoardSocket(dai.CameraBoardSocket.RGB)
cam_rgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
cam_rgb.setInterleaved(False)
cam_rgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.RGB)

# Create output
xout_rgb = pipeline.createXLinkOut()
xout_rgb.setStreamName("rgb")
cam_rgb.preview.link(xout_rgb.input)

# Define a source - two mono (grayscale) cameras
left = pipeline.createMonoCamera()
left.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
left.setBoardSocket(dai.CameraBoardSocket.LEFT)

right = pipeline.createMonoCamera()
right.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
right.setBoardSocket(dai.CameraBoardSocket.RIGHT)

# Create a node that will produce the depth map (using disparity output as it's easier to visualize depth this way)
depth = pipeline.createStereoDepth()
depth.setConfidenceThreshold(200)
left.out.link(depth.left)
right.out.link(depth.right)
```

```python

# Create output
xout = pipeline.createXLinkOut()
xout.setStreamName("disparity")
depth.disparity.link(xout.input)

# Pipeline defined, now the device is connected to
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queue will be used to get the rgb frames from the output defined above
    q_rgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)

    # Output queue will be used to get the disparity frames from the outputs defined above
    q = device.getOutputQueue(name="disparity", maxSize=4, blocking=False)

    while True:
        in_rgb = q_rgb.get()  # blocking call, will wait until a new data has arrived

        # Retrieve 'bgr' (opencv format) frame
        cv2.imshow("bgr", in_rgb.getCvFrame())

        in_depth = q.get()  # blocking call, will wait until a new data has arrived
        # data is originally represented as a flat 1D array, it needs to be converted into HxW form
        frame = in_depth.getData().reshape((in_depth.getHeight(), in_depth.getWidth())).astype(np.uint8)
        frame = np.ascontiguousarray(frame)
        # frame is transformed, the color map will be applied to highlight the depth info
        frame = cv2.applyColorMap(frame, cv2.COLORMAP_JET)
        # frame is ready to be shown
        cv2.imshow("disparity", frame)

        if cv2.waitKey(1) == ord('q'):
            break
```

PART C Q4) Ran the Calibration script. Ran the given lite calibration script using the command python calibrate.py -s 2.5 -brd OAK-D-LITE -db . Output was the Camera Matrix.

Got the following Matrix:

```
[[503.52258196   0.         326.25873716]
 [  0.         502.32811587 182.53332087]
 [  0.           0.           1.        ]]
```

P.S all the code and videos are uploaded on GITHUB AS WELL