Communicating with Oak-D-Lite

January 23, 2023

[2]:
```
import depthai as dai
pipeline = dai.Pipeline()
```

[ ]:
```
#a mono camera is one of the left or right ones (greyscale depth perception␣
 ↪ones). We have just created a node for the mono camera

mono = pipeline.createMonoCamera()

#specify that it is the left camera. Creates XLinkIn internally, connecting the␣
 ↪host (the computer) TO THE device (the OAK D).

mono.setBoardSocket(dai.CameraBoardSocket.LEFT)

#Creates XLinkOut, and names it specifically for the given camera (can have␣
 ↪XLinkOut's for multiple cameras).
xout = pipeline.createXLinkOut()
xout.setStreamName("left")
#The below links the mono camera to the actual output. Takes the output of the␣
 ↪mono camera and putting it as the input to the XLinkOut. We are attaching␣
 ↪the camera (left camera) to xLinkOut (left)
mono.out.link(xout.input)

#transfers all the code from the host to the actual device

with dai.Device(pipeline) as device:

    # queries the XLinkOut from the device to the host
    queue = device.getOutputQueue(name = "left")
    #gets frame from device
    frame = queue.get()

    #getCvFrame() converts frame into the OpenCV format
    imOut = frame.getCvFrame()
    #displays frame
    cv2.imshow("Image", imOut)
```

[ ]:
```
#rectifiedLeft, rectifiedRight and disparity are XLinkOut nodes

import cv2
import depthai as dai
import numpy as np

def getFrame(queue): #gets the last frame from the output queue, and converts␣
 ↪to OpenCV format
    # Get frame from queue
    frame = queue.get()
    # Convert frame to OpenCV format and return
    return frame.getCvFrame()

def getMonoCamera(pipeline, isLeft):
    # Configure mono camera
    mono = pipeline.createMonoCamera()

    # Set camera resolution
    mono.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)

    if isLeft:
        # Get Left camera
        mono.setBoardSocket(dai.CameraBoardSocket.LEFT)
    else:
        # Get Right camera
        mono.setBoardSocket(dai.CameraBoardSocket.RIGHT)
    return mono

# NEW CODE

def getStereoPair(pipeline, monoLeft, monoRight):
    # Configure stereo pair for depth estimation
    stereo = pipeline.createStereoDepth() #creating the stereo depth node
    # Checks occluded (when object comes too close to the device) pixels and␣
 ↪marks them invalid
    stereo.setLeftRightCheck(True)

    # Configure left and right cameras to work as a stereo pair
    monoLeft.out.link(stereo.left)
    monoRight.out.link(stereo.right)

    return stereo

def mouseCallback(event, x, y, flags, param):
    global mouseX, mouseY
    if event == cv2.EVENT_LBUTTONDOWN:
        mouseX = x
```

```
        mouseY = y

if __name__ == '__main__':
    mouseX = 0
    mouseY = 640

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

    # Set up left and right cameras
    monoLeft = getMonoCamera(pipeline, isLeft = True)
    monoRight = getMonoCamera(pipeline, isLeft = False)

    # Combine left and right cameras to form a stereo pair
    stereo = getStereoPair(pipeline, monoLeft, monoRight) # taking the output␣
 ↪from the left and right mono cameras and putting them as input to the stereo␣
 ↪camera

    # Define and name output depth map,
    # xoutDepth = pipeline.createXLinkOut()
    # xoutDepth.setStreamName("depth")

    # Set XLinkOut for disparity, rectifiedLeft, rectifiedRight

    xoutDisp = pipeline.createXLinkOut()
    xoutDisp.setStreamName("disparity")

    xoutRectifiedLeft = pipeline.createXLinkOut()
    xoutRectifiedLeft.setStreamName("rectifiedLeft")

    xoutRectifiedRight = pipeline.createXLinkOut()
    xoutRectifiedRight.setStreamName("rectifiedRight")

    # Pipeline is defined, now we can connect to the device

    with dai.Device(pipeline) as device:

        # Output queues will be used to get the rgb frames and nn data from the␣
 ↪outputs defined above. Gets the frames back.

        disparityQueue = device.getOutputQueue(name = 'disparity', maxSize = 1,␣
 ↪blocking = False)
        rectifiedLeftQueue = device.getOutputQueue(name = 'rectifiedLeft',␣
 ↪maxSize = 1, blocking = False)
        rectifiedRightQueue = device.getOutputQueue(name = 'rectifiedRight',␣
 ↪maxSize = 1, blocking = False)
```

```
        # Calculate a multiplier for colormapping disparity map
        disparityMultiplier = 255 / stereo.getMaxDisparity()

        cv2.namedWindow("Stereo Pair")
        cv2.setMouseCallback("Stereo Pair", mouseCallback)

        # Variable use to toggle between side by side view and one frame view.
        sideBySide = False

        while True:

            # Get disparity map
            disparity = getFrame(disparityQueue)

            # Colormap disparity for display
            disparity = (disparity * disparityMultiplier).astype(np.uint8)
            disparity = cv2.applyColorMap(disparity, cv2.COLORMAP_JET)

            # Get left and right rectified frame
            leftFrame = getFrame(rectifiedLeftQueue);
            rightFrame = getFrame(rectifiedLeftQueue)

        if sideBySide:
            # Show side by side view
            imOut = np.hstack((leftFrame, rightFrame))
        else:
            #Show overlapping frames
            imOut = np.uint8(leftFrame/2 + rightFrame/2)

        #converting image to colour
        imOut = cv2.cvtColor(imOut, cv2.COLOR_GRAY2RGB)

        imOut = cv2.line(imOut, (mouseX, mouseY), (1280, mouseY), (0, 0, 255),␣
↪2)
        imOut = cv2.circle(imOut, (mouseX, mouseY), 2, (255, 255, 128), 2)
        cv2.imshow("Stereo Pair", imOut)
        cv2.imshow("Disparity", imOut)

        # Check for keyboard input
        key = cv2.waitKey(1)
        if key == ord('q'):
            # Quit when q is pressed
            break
        elif key == ord('t'):
            # Toggle display when t is pressed
            sideBySide = not sideBySide
```

0.1 Notes:

- Depth is inversely proportional to disparity; the larger the disparity (the same image being apart due to left and right view) is, the closer that image would then be (lower depth)
- Shadowish areas are where both cameras cannot see (depth is incorrect as both cameras need to be able to see the point in order to estimate depth)
- Another reason there could be a shadow-like area (incorrect depth perception) is that usually there are two pixels at different points (seen by the different cameras), that are usually matched. If you cannot do that, depth perception will go wrong. E.g. A flat white wall will have identical pixels all around, therefore even though we may be able to see it, the two cameras won't be able to match the points because they all look the same. Even shiny surfaces face this problem.
- There is a minimum distance from the camera that the object/person must be farther than to get depth perception. If we are too close, the disparity will be very large, and Oak D is not programmed to look for such a large disparity (for efficiency purposes). This can be altered (make the search space larger), but will make the depth estimation slower. (Too close and the depth estimate will be poor as we are not searching in that disparity range)
- Scan lines will show that a point will lie on the same vertical axis (if we slice horizontally) because of image rectification.
- To find corresponding points on two different mono cameras, you would choose a point on one camera, and then move along the scan line on the other camera, but not look through the entire scan line (otherwise it would take too long), you just need a small number of points to check over.