Introduction to Robotics Lab

Laiba Ahmed and Muhammad Suleiman Qureshi and Syed Muhammad Mustafa

April 2023

Lab 08

Task 8.1

```
function determineIntrinsics()
    % Make Pipeline object to manage streaming
    pipe = realsense.pipeline();

    % Start streaming on an arbitrary camera with default settings
    profile = pipe.start();

    % Extract the color stream
    color_stream = profile.get_stream(realsense.stream.color).as('video_stream_profile');

    ir_stream = profile.get_stream(realsense.stream.depth).as('video_stream_profile');

    % Get and display the intrinsics
    color_intrinsics = color_stream.get_intrinsics()
    ir_instrinsics = ir_stream.get_intrinsics()


end
```

```
color_intrinsics =

  struct with fields:

     width: 1920
    height: 1080
       ppx: 931.9031
       ppy: 525.8997
        fx: 1.4071e+03
        fy: 1.4071e+03
     model: 0
    coeffs: [0 0 0 0 0]
```

```
ir_instrinsics =

  struct with fields:

     width: 640
    height: 480
       ppx: 316.3301
       ppy: 245.5135
        fx: 475.2710
        fy: 475.2711
     model: 2
    coeffs: [0.1297 0.1540 0.0047 0.0061 -0.0812]
```

Figure 1: Instrinsics of both cameras

Task 8.2

```
function determineExtrinsics()
    % Make Pipeline object to manage streaming
    pipe = realsense.pipeline();
```

```
    % Start streaming on an arbitrary camera with default settings
    profile = pipe.start();

    % Extract the color and depth streams
    color_stream = profile.get_stream(realsense.stream.color).as('video_stream_profile');
    depth_stream = profile.get_stream(realsense.stream.depth).as('video_stream_profile');

    Tdc = depth_stream.get_extrinsics_to(color_stream)
    t = [Tdc.translation]';
    R = [Tdc.rotation(1:3) ; Tdc.rotation(4:6); Tdc.rotation(7:9)];
    T = [R t; 0 0 0 1] %standard homogenous transformation
end
```

```
>> determineExtrinsics

Tdc =

  struct with fields:

       rotation: [1.0000 -0.0016 -4.0173e-04 0.0016 1.0000 -0.0031 4.0682e-04 0.0031 1.0000]
    translation: [0.0247 0.0013 0.0043]


T =

    1.0000   -0.0016   -0.0004    0.0247
    0.0016    1.0000   -0.0031    0.0013
    0.0004    0.0031    1.0000    0.0043
         0         0         0    1.0000
```

Figure 2: Extrinsic Parameters from Depth Camera to Color Camera

We did not use the transformation above because we did not use the depth camera in our calculations.

Task 8.3

Resolution: 640x480 (IR) 1920x1080 (Depth)
Frame Rates(FPS): 30,60,120,200 (IR) 10,30,60 (Depth)
Depth field of view: Horizontal FOV: 69±3 Vertical FOV: 54±2
Depth Start Point: Front of Lens(Z'): 0.9mm, Back of Module (Z"): 3.0mm

Task 8.4

f1 = (d/2)/tan(β/2) = (500mm/2)/tan((68-2)/2) = 385 mm

f2 = (d/2)/tan(α/2) = (500mm/2)/(tan((41.5-2)/2) = 696 mm

We take the larger number in order to get the entirety of the board.

Once we placed the camera about 696 mm, we could see the edge of the board (from the vertical part of the camera) and we could see some extra outside the board (from the horizontal part of the camera). This comes in line with our calculations because to capture the complete horizontal aspect of the board, we only needed our camera to be at 385 mm and now since it is at a height of more than 385 mm, we can see a bit beyond the board and can capture the vertical edge of the board.

Task 8.5

We created a transformation that mapped from the camera frame to the world frame origin. We got to it by rotating by $-\pi$ about the x-axis and translating by Zc, the distance from the camera to the desk.

$$T_W^C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & z_c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The code below displays going from pixel coordinates to real-world coordinates.

```
function real = pixel_coo_to_real(pixel)
    fx = 1407;fy = 1407;u0 = 932;
    v0 = 526;s = 0;K = [fx s u0 ; 0 fy v0 ; 0 0 1 ];
%     coo = inv(K)*[(pixel(1)-22)/ ((602-22) / 1920); (pixel(2)-76)/ ((417-76) / 1080); 1;] * 69
    coo = inv(K)*[(pixel(1)-17)/ ((610-17) / 1920); (pixel(2)-70)/ ((426-70) / 1080); 1;] * 696;
    coo =[coo;1];
    zc = 696    ;
    R_T_W = [1 0 0 0;
             0 -1 0 0;
             0 0 -1 zc;
             0 0 0 1];
    final = R_T_W*coo;
    real=[final(1) final(2)];
end
```

Task 8.6

Please find the video and codes attached to the submission. The final code to run to make everything work together will be the code titled final.m. The video displays our test run which is very close to success. Other than a small displacement which does not allow our end-effector to accurately grasp the cube, our objective is almost accomplished. We think that our small offset comes from the fact that the image we get is radially distorted (barrel distortion) as shown in the figure attached below (we go from one edge to another but the middle of the image is above the edge). Another issue we had to deal with was the fact that our image dimensions or resolution in our datasheet was given by 1920x1080 while our extracted image is only 640x480 and even in that, we still have to account for the black edges that appear.

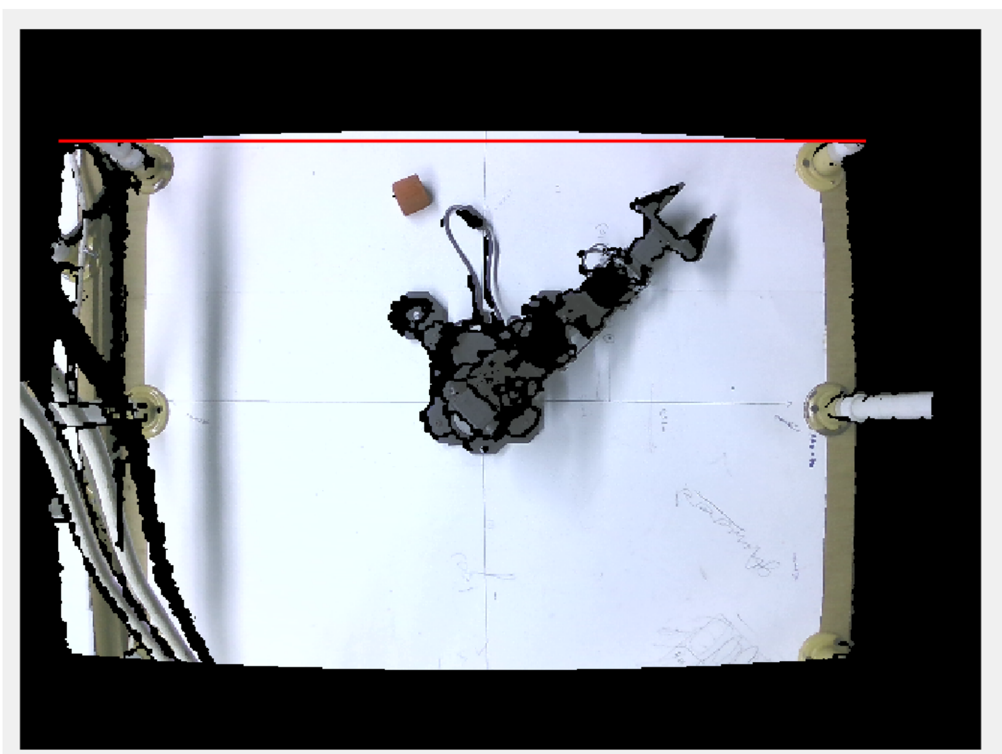

Figure 3: Displaying the positive radial distortion