Assignment 2 – Monitoring against COVID19

1. Introduction

In this assignment you will have several tasks, with the final aim being the monitoring of the number of persons in a laboratory. To do so, we will use an Intel RealSense RGBD sensor, namely a D435 model, placed above the entrance of the laboratory and pointing down, as shown in Figure 1. The goal is to count the number of persons inside the lab, with the sensor being used to count for the number of persons entering and exiting the laboratory. Given that the sensor is pointing down, it is harder to see the people faces, thus the risk of data protection problems due to identity tracking is much smaller – we do not want to distinguish between people, but only know the number of persons inside the room, i.e, the number of persons that have entered the room minus the number of persons that has left the room.

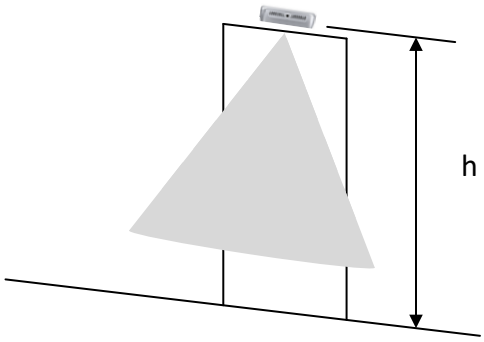

Figure 1 – Sketch of the sensor placement above the entrance.

This assignment is divided in two mandatory tasks and one optional task:

Task 1: implement computing depth maps from stereo camera images.
Task 2: compute the number of people inside a laboratory, by tracking persons entering and exiting the room.
Task 3 (optional)**:** Perform the calibration of the vision system.

This document is structured as follows: the reminder of this section provides for an description of the Intel RealSense D435; Section 2 describes the material provided for this assignment, namely the provided files and their contents; Section 3 provides for the first task of this assignment; Section 4 describes the second task of this assignment; Section 5 describes an optional task that can be implemented for extra credit; finally, Section 6 provides for some final notes which can be helpful when developing this assignment.

1.1 The Intel RealSense D435

With the release of the Intel RealSense devices, a new low cost, high accuracy, RGBD devices appeared in the market, allowing for seamless use of these types of sensors, mainly for robotics, production, and monitoring/inspection applications. Figure 1 shows an external view

of the Intel RealSense D435 device which, measuring only 90 mm x 25 mm x 25 mm, includes an active stereo vision system, composed by an infrared projector and two infrared cameras, and an RGB camera, as shown in Figure 3. Furthermore, the camera is an USB-powered device with an onboard Vision Processor ASIC module capable of computing 1280x720 pixels depth images at 90 fps from about 0.1m to 10 m, all while consuming about 2 W.

Figure 2 – Intel RealSense D435 external view.

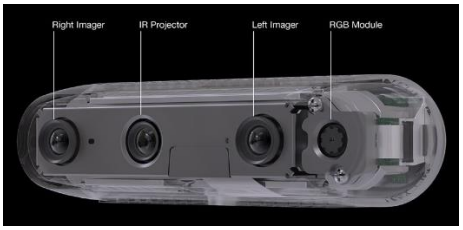

Figure 3 – Intel RealSenseD435 inner view.

When connected to Windows or Linux, the camera is detected as an UVC device, and thus immediately available for typical camera use. However, in order to fully take advantage of the camera capabilities, Intel provides for the Intel RealSense SDK, which is multi-platform and provides bindings/wrappers for multiple languages, including Python. When using this SDK, you have access to each individual stream (images taken from the individual RGB and/or infrared cameras), to the depth stream (depth image computed onboard the sensor from the two infrared cameras), among other functionalities such as controlling the infrared projector. More details on the use of the SDK will be given below.

2. Setup and provided material

In this section we describe the requirements needed to be installed in order to work with the provided data, and detail that provided data.

Most of the data provided for this project is given as a *bag* file. Although we could have provided the students with the videos of each camera, by providing the data as bag files, you get all the data that was acquired from the three cameras with the corresponding timestamps, as if you were repeating exactly the same experiments, i.e., as if you were acquiring the data directly from the camera. If we were to provide the individual videos of each camera, it would be more likely to come across synchronization problems.

Intel provides currently provides for the Intel RealSense SDK 2.0, which supports many different programming languages, such as C++, Javascript and Python, among others. Since we are using Python 3.7.X, the SDK can be easily installed by running the following command on the Windows terminal:

```
>> pip install pyrealsense2
```

You are advised to use Python 3.7.X, as instructed in the beginning of the semester, in order to use the install command above, otherwise you need to install the library from source. Regarding the documentation for the Intel RealSense SDK, it is available here, but you should only go through it if you are interested in further details, given that bellow you will find the information needed to start working with the sensor data.

Having installed the SDK, you can now interact with the supplied data files. If you which to have a glance on the data being made available, open the Intel RealSense Viewer that was just

installed. Then, in the viewer, click the *Add Source* option on the top left, choose *Load Recorded Sequence*, choose the bag file, and then click the play button.

The following list describes all files being made available, both bag files and non-bag files (more information will be given later):

CV_D435_20201104_160738_RGB_calibration.bag[1]**:** bag file with containing only the RGB stream, while showing a known chessboard pattern in different poses;

CV_D435_20201104_161043_Full_calibration.bag[1]**:** bag with all the sensor streams, while showing a calibration pattern and with the infrared projector turned off.

CV_D435_20201104_162148.bag[1]**:** full bag file with standard acquisition of all streams, with the sensor infrared projector turned on;

CV_D435_IR_(left|right)_01(.png|.raw|_metadata.csv): which correspond to the left and right infrared cameras snapshots, with the infrared projector on, in PNG and RAW format. The csv files contain information about the stream, namely the resolution and the camera intrinsic parameters and the distortion model being used (in this case, the Brown-Conrady distortion model).

CV_D435_IR_(left|right)_calib_Infrared(.png|.raw|_metadata.csv): similar as the one above, but taken with the infrared projector turned off and with the chessboard calibration pattern placed on the floor level.

CV_D435_RGB_calib_Color.png: snapshot taken with the RGB camera in the same situation as the infrared cameras calibration capture above.

If you are using Microsoft Windows, the Intel RealSense SDK is usually installed in `C:\Program Files (x86)\Intel RealSense SDK 2.0`. There you can find a `samples` folder with many examples, but which are mainly in C++. On github you find more examples written in Python where, among others, you have the read_bag_example.py example, which shows on one can retrieve data from a bag file with Intel RealSense data. Download this example, change, on line 43, in the function `enable_stream` call, the resolution to 848x480 (2nd and 3rd function arguments, respectively) and the framerate to 15 (5th function argument), to match the configuration streams with which the bag was obtained (the provide *read_bag_example.py* file already includes these changes).

The example you have just tried, shows on to obtain a depth frame formatted as an OpenCV (Numpy) image which you can use for your project. The example is well explained through the included comments, but we will go through the most important functions to better understand the interaction with the Intel RealSense cameras and streams.

```
    # Create pipeline
    pipeline = rs.pipeline()

    # Create a config object
    config = rs.config()
    # Tell config that we will use a recorded device from file to be used
 by the pipeline through playback.
```

[1] The bag files are given as links, due to their size (~400Mb, ~800Mb and ~3Gb, respectively). All other files are included in the assignment compressed file.

```
    rs.config.enable_device_from_file(config, args.input)
    # Configure the pipeline to stream the depth stream
    config.enable_stream(rs.stream.depth, 848, 480, rs.format.z16, 15)

    # Start streaming from file
    pipeline.start(config)
```

Code 1 – Library initialization, stream configuration and capture start.

In Code 1 you see the typical configuration and start of the stream. The only difference when compared to the real camera, is the use of the config file with the function *enable_device_from_file*, which would not be used if we were communicating directly with a real camera. In the RealSense sensors each camera provides for a stream, and an additional *depth* stream is also made available from the onboard, hardware-based, depth computation using the infrared stereo vision system. The *enable_stream* can be used to configure each stream and, in this case, is being used to configure the *depth* stream to use a frame with 848 pixels width, 480 pixels height, `rs.format.z16` (16-bit unsigned int per pixel), and a framerate of 15 fps. Note that only when you call *pipeline.start(config)*, does the camera actually starts streaming with the desired configuration.

```
# Create colorizer object
    colorizer = rs.colorizer()

    # Streaming loop
    while True:
        # Get frameset of depth
        frames = pipeline.wait_for_frames()

        # Get depth frame
        depth_frame = frames.get_depth_frame()

        # Colorize depth frame to jet colormap
        depth_color_frame = colorizer.colorize(depth_frame)

        # Convert depth_frame to numpy array to render image in opencv
        depth_color_image = np.asanyarray(depth_color_frame.get_data())

        # Render image in opencv window
        cv2.imshow("Depth Stream", depth_color_image)
        key = cv2.waitKey(1)
        # if pressed escape exit program
        if key == 27:
            cv2.destroyAllWindows()
            break
```

Code 2 – Acquiring and showing a depth frame.

In Code 2 you find the code related with capturing each depth frame and showing it to the user. One first starts by obtaining a colorizer object, which is used later to create a color image from depth files, with the color proportional to each pixels depth (helpful for visualization purposes only). The Intel RealSense cameras provides for synchronization between streams. By

calling the function *wait_for_frames* the code blocks until a new set of frames is available from the various cameras. After that, one can retrieve the newly acquired frames, in this case the depth frame. The *colorize* function is used here to generate a color image with the color proportional to the depth for each pixel, with 8 bit per color, per pixel. The function *get_data*, together with the *Numpy asanyarray* function, is then used to obtain, with little overhead, an OpenCV/Numpy matrix that can be used with OpenCV/Numpy functions. If no type is specified, it is determined automatically from the data.

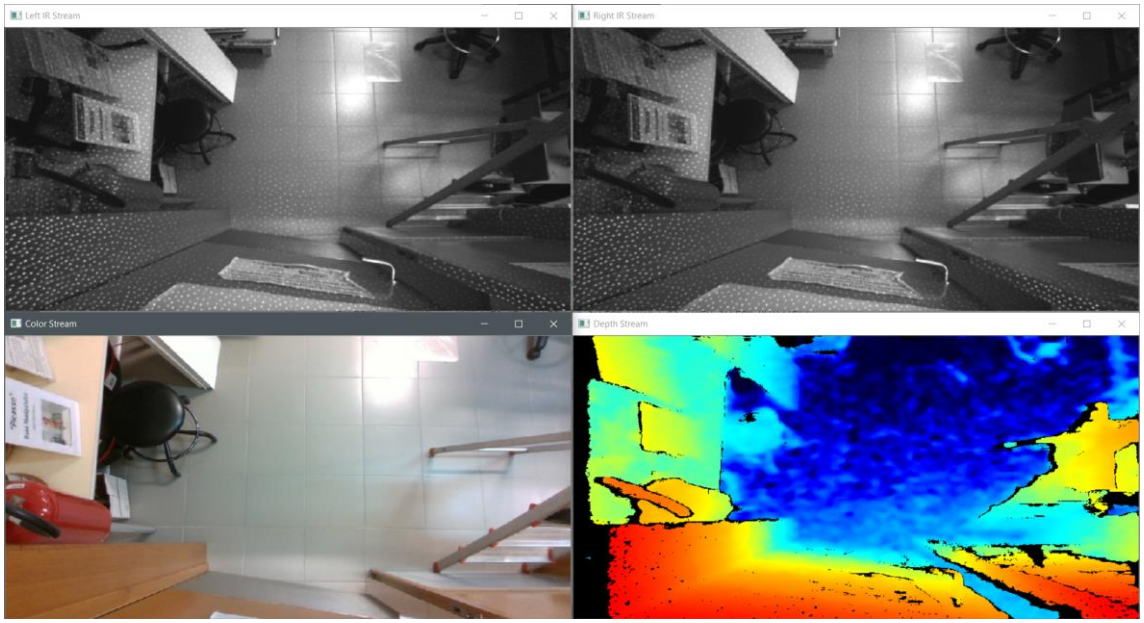

Figure 4 – Overview of the Intel RealSense D435 camera streams from the provided bag file.

Another script is provided, called `read_bag_full_example.py`, which, besides showing the depth frame, also shows the individual infrared and RGB frames, as can be seen in Figure 4.

3. Assignment task 1 – Stereo vision

Although the Intel RealSense sensors already computed the depth map, in this task we want to evaluate computing it using the standard algorithms discussed in the Computer Vision classes.

The Intel RealSense cameras are all factory calibrated [1], and the transformation between each camera onboard the sensor can be computed through the `get_extrinsics_to` function. The provided `read_bag_full_example.py` file also includes the code to compute obtain both the intrinsic parameters of each camera, and the extrinsic parameters of each camera, considering the left camera as the sensor origin. However, because we are using the bag file, the returned values are not correct. As such, and as denoted on the script file, we still simply assume that all cameras are parallel, and that the distance between the two stereo cameras is 5 cm.

Regarding the intrinsic parameters, which are obtained in the example script, and can also be seen in the *metadata* files, we are assuming no distortion for all cameras. For instance, for the infrared cameras, the camera intrinsic parameters are given by (values shown are rounded to the 3rd decimal place):

$$K = \begin{bmatrix} 425.789 & 0 & 426.796 \\ 0 & 425.789 & 234.032 \\ 0 & 0 & 1 \end{bmatrix}$$

Given the information above, **implement a script which computes the depth image from the two infrared streams and compare your result with the depth stream provided by the sensor**. To convert from the sensor depth image pixel value to meters, consider the depth scale given in the example script, in $^{m}/_{pixel\ value}$. Your script should run with the bag file provided and show the computed depth frame in each iteration.

To compare your results with the sensor computed depth image, compute, for each iteration and per pixel, the difference between your depth image and the sensor depth image, computing the average absolute difference and its standard deviation. If the sensor depth image has a 0 value, then it means it was not computed, in which case you ignore that pixel in the corresponding frame (it will not count for the measurements being done). To avoid memory problems when computing the average difference and its standard variation, use the Welford's online algorithm [2] for each depth pixel over time, given by:

$$\bar{x}_n = \frac{(n-1)\bar{x}_{n-1} + x_n}{n} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

$$\sigma_n^2 = \frac{M_{2,n}}{n},$$

where

$$M_{2,n} = M_{2,n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n),$$

and

n – Current number of total elements/frames, corresponding also to the acquisition discrete time instant.
x_n – Element computed at *time instant* n.
$\bar{x}_n$ – Average of all first n elements.
σ_n^2 – Variance of all first n elements (square of the standard deviation).

Include in your results the average time taken to compute each depth frame and the average FPS. To measure the time in your script you can use the `process_time` function.

4. Assignment task 2 – People counter

In this task, your goal is to count the number of people inside the laboratory. To do so, you need count people entering the room and exiting the room, considering the difference to be number we wish to obtain. You can assume that there are no people in the laboratory in the beginning of the experiment.

To solve this task, you can use any of the sensor cameras (depth, infrared and/or color). If you use the depth image, you can either use the depth map directly provided by the sensor, or the result of the depth computed in the first task. The output of your algorithm should be a plain Comma Separated Value (CSV) text file (using the "csv" extension) with the time at which a

person entered or left the laboratory (HH:MM:SS), the event type (“in” or “out) and the number of people that stayed inside (number of the entrances minus the number of exits). The first file line must show your group number and corresponding student names and numbers, and the second line the algorithm start time and initial number of people (typically, 0). Table 1 shows an example of such file.

```
# Group X, <num1>, <name1>, <num2>, <name2>
10:22:32,none,0
10:23:32,in,1
10:25:25,in,2
10:27:45,out,1
10:37:48,in,2
10:39:15,out,1
10:57:42,out,0
```

Table 1 – Results output file on access monitoring.

5. Extra credit

Although the camera is factory calibrated, you can compute your own intrinsic and extrinsic parameters. The “calibration” files with the chessboard pattern are provided for that matter, which you can use to compute your own calibrations parameters. This task is optional but can provide with up to 2 extra values. If not done, the maximum score is limited to 19 values. The goal is to calibrate only the cameras that you use.

6. Final notes

The depth computed by the Intel RealSense using the onboard hardware, albeit based on the same algorithm as we learned in classes includes several post-processors to improve the estimated depth. As such, in Task 1, do not expect to obtain the same values as the sensor computed depth image, although these should be similar.

Different approaches can be used when detecting persons entering and leaving the laboratory, but using the depth is probably the approach that yields best results, and which is less dependent on environment changes. Although simple solutions might provide for fast results, solutions that make use of the taught algorithms, such as tracking algorithms, will be valued.

Although you are asked to compute time related information regarding the execution of your algorithm, the time factor is of little importance when compared to the solution algorithm per se. Furthermore, your algorithm can be tested with new experiment videos, but you can assume that the camera pose will remain the same as on the provided data.

As stated in the evaluation calendars, the work is to be **submitted** on Moodle **until 16th of December**, with the defenses taking place on the 18th of December.

References

[1] L. Keselman, J. Iselin Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, ‘Intel RealSense Stereoscopic Depth Cameras’, Jul. 2017.

[2] B. P. Welford, ‘Note on a Method for Calculating Corrected Sums of Squares and Products’, *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962, doi: 10.1080/00401706.1962.10490022.