Canadian Museum for Human Rights

Project Report

Kunal Rajpal, Ronnie D'Souza, Zooey Schock

1. Introduction (background)

The aim of this project was to create a library for a touch screen-like system using the Intel Realsense d435 depth camera in order to allow for regular surfaces (non screen/touch-screen surfaces) to be used as touch screens. This library would allow for touch detection on flat or curved surfaces such as walls, posters, tables, and other similar situations.

This project uses the DIRECT[1,2] touch tracker as a base. DIRECT is written in C++ and uses the Microsoft Kinect in concert with OpenCV and openFrameworks to perform touch detection on non touch surfaces. Our goal was to use Python and OpenCV to develop a portable library for the CMHR for use in their exhibits.

2. Progress made

i. Research
ii. Edge detection
iii. Background separation

3. Difficulties encountered

Software Unsuitability

We had initially planned on using Google's MediaPipe Hands library in combination with the Intel RealSense Camera. While we were able to get it set up fairly easily, we soon found that

[1] Robert Xiao et al, "DIRECT: Making Touch Tracking on Ordinary Surfaces Practical with Hybrid Depth-Infrared Sensing," *ISS '16: Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Space*, (2016): 85–94.
[2] DIRECT github repository: https://github.com/nneonneo/direct-handtracking

this was unsuitable for precise depth detection as is needed for touch detection. We also found it to be quite slow and therefore decided that it would be unsuitable for our task and worked to restructure our plan.

Kunal (Wrapper)

1. ARM and Intel Realsense

RealSense SDK was designed to run on Intel x86 architecture and hence does not have native support for the new ARM-based Apple Silicon architecture on the M1 and M2 Macbook machines. We explored several possible solutions to this problem, including:

- Virtual Machines: (VMWare/VirtualBox): would likely have performance issues
- Rosetta 2: A compatibility layer that translates x86 code to ARM code allowing it to run on the architecture of the latter. This was a potential solution but would also come with significant performance issues.
- Dual-boot: Dual-boot was another potential solution.

2. Performance using Python

We initially wanted to make wrappers to embed some C/C++ code in the library. C/C++ may have been better in terms of performance and control over low-level hardware but since our focus was on functionality and portability, python seemed to be a better choice for a library, especially one that was mainly intended to be a proof of concept. This proved to be tough due to limited documentation and support for the Intel RealSense library.

Ronnie (Background Separation)

1. openCV and numpy image formatting

Since the default Realsense objects are not compatible with openCV, it took a fair bit of trial and error to find the proper representation for the depth data. The examples in the template

were not applicable for our purposes and eventually I was able to settle on using uint8 numpy arrays. There were several opencv methods to show images, each with its own requirements. Figuring out which method to use and what data it needed was an initial obstacle. In the end imshow() and uint8 numpy representation proved the simplest.

2. Performance optimisation

Pre-scanning the background live seemed to reduce performance, compared to scanning the background in a separate program, saving it, and then reading it to be used live. There were many operations that ate up performance, like multi frame background processing, which had to be substituted with a single background frame instead of the rolling background that was used in the DIRECT paper.

3. Conversion from C/C++ to Python

The available DIRECT code was commented, but not enough to explain what variables and methods did. This made it difficult to understand what the code did as the purpose of variables that were unclearly named was hard to ascertain. Ultimately much of the code was unused in the background separation as it was not as optimised in python. This was further exacerbated by the fact that the code used Kinect API methods which didn't usually have Intel Realsense equivalents, even if the methods were understood.

Zooey (Edge Detection, Diff Image)

1. Documentation

One of the most consistent issues in development was a lack of easily digestible documentation. openCV's documentation is well above average, and there are many helpful techniques illustrated in addition (these will be included in section 6).

The pyrealsense2 library had extremely incomplete documentation, with many functions having no description, and many data types completely unexplained. Additionally, openFrameworks has documentation that is occasionally inconsistent, and directs users to its forums in lieu of providing illustrative examples or more developed explanations. The openFrameworks Kinect wrapper was also without documentation (there is documentation for V1, but DIRECT uses V2)

Finally, although the DIRECT whitepaper was very helpful and written in a straightforward manner, the specifics were not covered, and the C++ code for the project was written with extremely sparse comments and many vague variable names making it laborious to work through. Correspondence with Dr. Xiao was helpful and very much appreciated, limitations being of course that he is a busy individual, and covering every issue that arose (especially via email) was not feasible.

2. Difference in hardware

The Kinect V2 is a time-of-flight depth camera while the Realsense uses stereo depth. Most of the DIRECT algorithms were still relevant despite the difference in technology, however accessing the different images necessitated some changes.

The Realsense also produced a lot of noise, and the depth information provided by it was highly volatile. This causes problems when building the edge map.

3. Complexity of algorithms

While algorithms such as Canny Edge Detection were relatively straightforward (likely due to its wide use), others such as region growing and anisotropic diffusion were far more complex. Even with the moderate background in mathematics our program entails, this algorithm took several tries to implement. In the end, the implementation in DIRECT served

well. All in all, computer vision is a completely new field for our group, and it contains many hidden complexities.

4. Differences between languages and libraries

Parsing DIRECT's C++ code and adapting it to the more familiar Python posed many problems. Using python meant that we do not have easy access to pointers, so these shortcomings had to be made up using parameters. Additionally, as mentioned before, the lack of consistent documentation of, and access to openFrameworks forced us to fill gaps left by that library using numpy.

Numpy is a fantastic tool and covered our needs well, but the differences resulted in a lot of necessary restructuring of functions and algorithms. One very prevalent example of this is how numpy stores images in ndarrays. In DIRECT, the authors were able to use bit manipulation exclusively when working with pixel data in the edge, blob and diff maps. Using numpy, these images were split into 3 separate channels (and it was not possible to use an alpha channel) which required many workarounds.

5. Cascading impacts of changes elsewhere in the library

Because of some of the performance limitations, the background updater function was changed to not include a rolling window. For this reason there is no ability to record a per-pixel mean or standard deviation. This prevents us from classifying new depth data based on its deviation from the background pixel mean, which is how the diff image is formed (the diff image being essential to the edge map building functionality).

4. Future recommendations

1. Additional time

Due to the complexity of the task, and the amount of research involved, more time would be required to produce a completely functional library.

2. Access to different/additional equipment

The moderate instability of the depth data provided by the Realsense produced roadblocks for our project. As the early attempts at using the Mediapipe library demonstrated, unstable depth data can result in unreliable performance. Of course this was also partially due to Mediapipe being designed more for gesture recognition and X/Y positioning.

While researching different approaches to depth detection, other depth cameras would often be featured in videos, git repositories, or papers, often shown producing very stable depth images. Two of these that could prove more useful for this particular application are the Microsoft Azure Kinect[3], and the Chronoptics Kea[4,5].

In addition to this, as mentioned by Kunal in his Difficulties Encountered section, several of the libraries are only available on specific platforms/architectures.

3. Official involvement of Dr. Xiao

As one of the authors of the DIRECT paper, and as a professor and researcher specializing in human-computer interaction, Dr. Xiao's assistance would be invaluable in this project.

---

[3] Azure Kinect DK Depth Camera: https://learn.microsoft.com/en-us/azure/kinect-dk/depth-camera
[4] Chronoptics Kea: https://www.chronoptics.com/products/kea
[5] Comparison of Kea and Realsense d435:
https://medium.com/chronoptics-time-of-flight/comparing-depth-cameras-itof-versus-active-stereo-e163811f3ac8

5. Discussion

The main difficulty encountered during this project has been documentation. The code base we worked from is very sparsely commented, likely as it was intended to be a proof of concept as a product of research itself. Significant time was spent researching the eccentricities of the libraries used, and our eventual use of different ones necessitated more research on top of this. In writing the edge detection code, I attempted to leave a surplus of comments to convey what I had learned about the original code which is very dense.

Replacing openFrameworks with a combination of numpy and an increased reliance on OpenCV was a large undertaking as well. It seems that openFramworks was primarily used for its access to a plugin that allows communication with the Kinect V2.

As mentioned, our other primary source of difficulty was performance. As the results of the research show, the methods demonstrated in DIRECT produce some of the most accurate and reliable touch tracking demonstrated to date. Much of our work has demonstrated the necessity of the choices made by Xiao et al in their project. Although Python is capable of producing very portable and readable code, it comes with significant performance drawbacks when compared to C/C++.

The anisotropic diffusion portion significantly slows the edge detection code, but it is also a large part of what makes DIRECT as accurate as it is. Having a very precise and, more importantly complete, edge map allows the arm segment flooding algorithms to function properly. This section of the DIRECT project uses raw pointers to access the pixel information stored in the images produced by the Kinect, as well as the images that the algorithms produce. As it is currently written, our python code uses passed parameters to accomplish this.

The pixel information was stored in 32 bit ARGB pixels in DIRECT, with the alpha channel used as an 8-bit flag to indicate there is relevant information in the other channels of the pixel. Our pixel data is stored in numpy ndarrays, which are RGB (or BGR) in an array of

size w x h x 3 (for the 3 R, G, and B channels). As a consequence of the difference, an image-sized array of flags was chosen to work in parallel. This entails increased memory usage. Additionally, DIRECT used bitwise or and and operations to augment the pixel data during image construction. Although we obviously had access to these operators, we still needed to iterate through 3 channels, and change the corresponding flag array entry, as opposed to being able to do the same in a single operation as in DIRECT.

A deque was chosen to implement the queue of fill-candidate pixels during the 8-way neighbor comparison, as lists are far too slow. It may be the case that a deque is also not sufficient for the speed this task demands.

In addition to the above, DIRECT records the depth of the background in a “rolling window”, taken at 15 fps. The background pixels are updated, and have per-pixel mean and standard deviation recorded as well to aid with the construction of the difference image.

In summary, we feel that the research done during this project was necessary to understand the algorithms used in DIRECT. Unfortunately, this was very time consuming, and (perhaps in addition to too much idealism) did not make it immediately clear to us how much of a disadvantage python would be at in terms of performance.

6. References

1. Xiao, R., S. Hudson, C. Harrison, C. Harrison.“DIRECT: Making Touch Tracking on Ordinary Surfaces Practical with Hybrid Depth-Infrared Sensing.” *ISS ’16: Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Space*. (2016): 85–94. https://doi.org/10.1145/2992154
2. Maeda, Junji, et al. “Segmentation of Natural Images Using Anisotropic Diffusion and Linking of Boundary Edges.” *Pattern Recognition.* 31.12. (1997): 1993-9.
3. Canny, John. “A Computational Approach to Edge Detection.” *IEEE Transactions on Pattern Analysis and Machine Intelligence.* PAM-8.6. (1986).

7. Appendix

Documentation used:

1. OpenCV documentation - DIRECT used OpenCV for most of their image processing functions, and these and many more have been useful in development of our project:
    a. cv::Mat - This is the general container for image data used in the c++ implementation of OpenCV. DIRECT uses this container along with a similar one in openFrameworks for reasons discussed below.
        i. https://docs.opencv.org/4.x/d6/d6d/tutorial_mat_the_basic_image_container.html
    b. Anisotropic Diffusion - A source used to help clarify the academic paper on the same subject listed above:
        i. https://docs.opencv.org/3.4/d4/d70/tutorial_anisotropic_image_segmentation_by_a_gst.html
    c. Canny Edge Detection - Description of the steps involved in the algorithm. This was helpful in understanding what parameters are suitable and may help with output quality:
        i. https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
        ii. https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html

    d. Image smoothing/filtering techniques - Initially different image smoothing algorithms were tried, eventually settling on Gaussian Blur. However, different algorithms could produce better/worse results based on lighting conditions and other factors:
        i. https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html
        ii. https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html
    e. General tutorial section of OpenCV
2. openFrameworks documentation - DIRECT seems to use openFrameworks as a way to connect the Kinect to openCV (it seems the two latter cannot communicate directly, or at least they chose not to do this). Much of the openFrameworks documentation is incomplete, and the suggestion in said documentation to consult the forums often leads to convoluted descriptions, hyper-specific use cases, or deprecated functions:
    a. OFImage - This was the main container used for image data in openFrameworks:
        i. https://openframeworks.cc/documentation/graphics/ofImage/
    b. ofxKinectV2 - Although this is not officially part of openFrameworks, it is a plugin developed for the library. This appears to be the main reason for DIRECT using openFrameworks:
        i. https://github.com/ofTheo/ofxKinectV2
3. Pyrealsense2 Documentation - This was the only extensive documentation of the pyrealsense2 library. It is lacks examples, descriptions of many functions and data structures (even things like descriptions of what return types mean making handling them difficult):
        i. https://intelrealsense.github.io/librealsense/python_docs/
    b. Some of the examples and documentation of the c++ pyrealsense library were helpful in understanding the equivalent in python (although many of these were simply absent in the latter):
        i. https://github.com/IntelRealSense/librealsense