An Offline Occupancy Mapping Implementation for the Microsoft Hololens 2

Kaan Akduman, Noelle Nelson, Tyler Petrie

Electrical, Computer, and Systems Engineering 499

Project 1 Report

Abstract— **This report describes our efforts to build a 3D mapping of a room from infrared (IR) sensor data acquired by the Microsoft Hololens 2. 42 sensor scans were acquired and successfully processed offline to create a maximum-likelihood graph depicting the occupancy of Olin 410 across three different horizontal planes at varying heights.**

I. Introduction

This project explores the implementation of occupancy grid mapping algorithms using data collected from off-the-shelf hardware. The ability to map spaces is critical for the operation of robots and autonomous devices, but mapping of spaces can be extended to other research realms.

This project explored mapping algorithms for use with Augmented Reality (AR) applications. Mapping is important for AR applications because the system needs to be able to orient itself in space in order to place holograms within reality. The AR device that was used for this project was the Microsoft Hololens 2. The sensors on the Hololens 2, including an RGB camera, infrared (IR) depth sensors, and gyroscopes, can be used to create a mapping of a space.

The goal of this project was simply to create a MATLAB program that can take raw sensor data from the Hololens 2 and build an occupancy grid map. One application of this technology is to help with rehabilitation for those with spinal cord injuries (SCI). The Hololens 2 can be used to create a mapping of their home in order to determine where potential modifications could be made in order to make day to day tasks easier. Due to the translational applications of this approach, we decided to combine existing data collection and processing software with mapping algorithms discussed in class to make an implementation that can be widely accessible.

II. Methods

The processing of our IR sensor data was done with a combined approach in MATLAB and Python. This was partially due to the skills of our group members but was also done to improve the ease of processing sensor data.

A. Sensor Data Acquisition and Pre-Processing

For this project, data was collected using the StreamRecorder application for the Microsoft Hololens 2 [1]. This application was created to provide researchers with direct access to the color camera and infrared (IR) depth cameras to use for computer vision and robotics systems. The Hololens 2 also provides the user with hand, head, and eye tracking data, as well as acceleration and orientation measured with on-board gyroscopes.

Due to the simplicity of the room layout, we decided to capture data in Olin 410. We marked out a 10ft by 10ft grid on the floor and defined an upper left triangle using the line y = x, which can be seen in Figure 1a. This defined our x and y-plane as we faced the entrance, which we refer to in this report as our world frame. To align the world frame with the Hololens, the user stood at the location (0,0) when the StreamRecorder application was launched. The user then walked 10 feet in the positive y-direction, before turning 90 degrees and walking 10 feet in the positive x-direction. Finally, the user turned 45 degrees to walk back to the world origin point. This was repeated while gathering IR data in order to achieve a complete scan.

To interpret the results of our scans, we had to transform the data from depth images to camera space, and then from camera space to grid space.

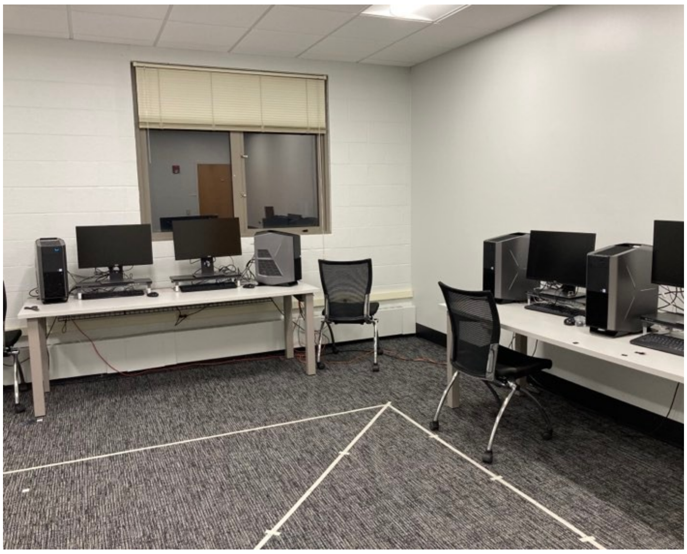

Figure 1a: Photograph showing Olin 410, where data was captured. The tape on the floor marks our x and y axes, as well as the line y=x.

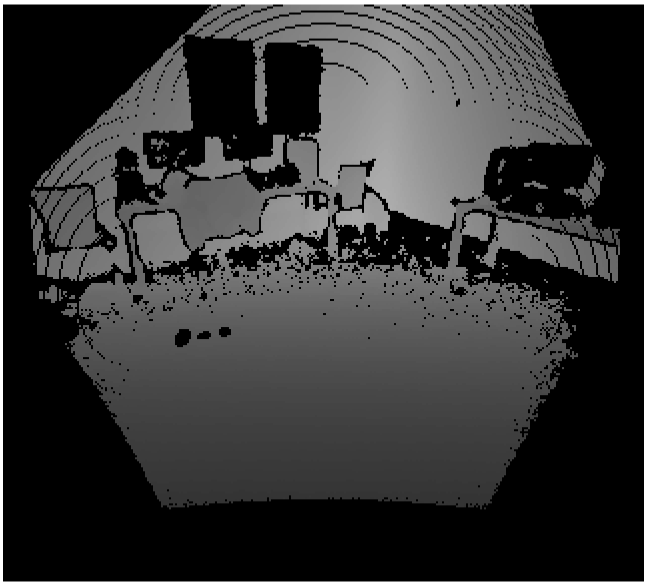

Figure 1b: A depth image captured by the StreamRecorder application prior to processing. Lighter pixels correspond to objects that are farther from the sensor, while darker pixels represent closer objects. The black pixels around the edge are data that is not captured due to the arcing nature of the IR scan. The black rectangles on the wall are a window, which is not well detected by IR imaging

This was achieved through the combination of multiplying multiple transforms in order to move between coordinate frames and using a truncated signed distance function (TSDF).

The coordinate transformations used in this process can be seen in Figure 2. It should be noted that these frames are not the same as traditional computer vision principles would dictate. In order to build our map, we had to construct a point cloud in the world frame. This required moving from a depth image, to camera space, to grid space, and then finally transforming between grid space frames. Transformations in grid space can be completed as follows:

$$T_{depth\ camera/world} = T_{RGB\ camera\ at\ start/world} * T_{curr}$$

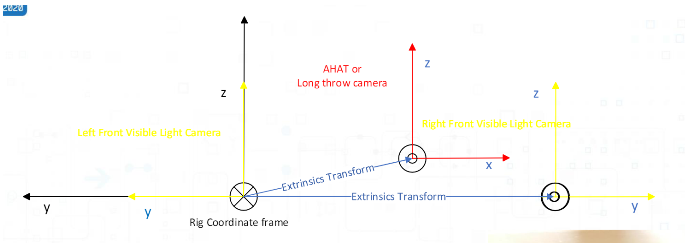

Figure 2: Coordinate frames for the Microsoft Hololens 2. The Rig Coordinate Frame is designated as the central transformation for all sensors and is placed at the left RGB camera. The depth sensor is located in the middle of the HMD, and it can be located by multiplying the Extrinsics Transform by the Rig Coordinate Frame Transform. The coordinate frames are defined with the z-axis pointing away from the user, towards what they are looking at. [3]

Conversion from the depth images to camera space, and again to grid space, was done using the aforementioned Python scripts. To extract the location of each point in camera space, each depth sensor image was multiplied by a binary lookup table, which provided the x, y, and z location of each point. The location of each point was then transformed to grid space through the TSDF process. This TSDF process aligned the 42 frames with the world frame and combined each frame to create a volume with a voxel size of 0.04m. The TSDF process is a raycasting method that increments through each frame captured by the Hololens 2 by providing weights for each grid that depends on the distance and angle of the surface. This TSDF creates almost perfect alignment with our world coordinate system, however, a rotation about the x-axis is performed in MATLAB since the camera defines the z-plane as directed away from the user, instead of up. Finally, since the initial RGB camera frame is determined by the location of the Hololens 2 relative to the room upon launching the app, the data points had to be shifted down 1.5 meters to account for the user's height. Figure 4 shows the completed point cloud, after final processing in MATLAB.

To accomplish these tasks, data was downloaded from the Hololens 2 for offline processing via a WiFi connection through the Windows Device Portal. This data consisted of a set of 42 depth images (containing the distance at each pixel in uint16), a lookup table to convert these images from 2D pixel space to 3D camera space, and two transformation matrices to align the depth sensor with the world frame. The TSDF process also assigns each point a color value based on the captured RGB data, but this was removed from our visualizations for simplicity.

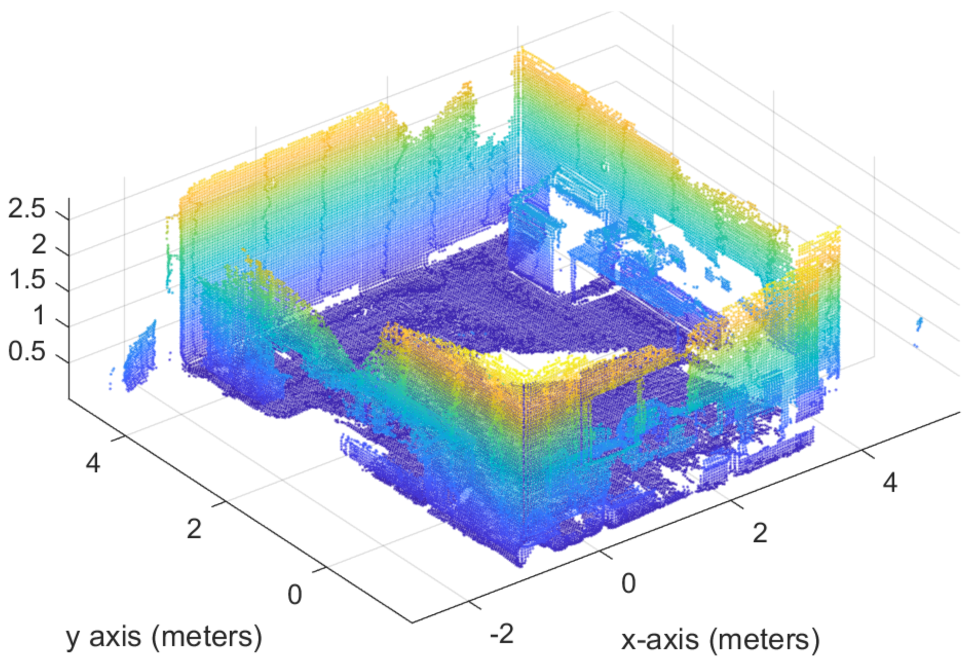

Fig. 3 Complete point cloud showing IR sensor data in MATLAB. Traditionally, the color would depict distance from the sensor where warmer colors indicate objects that are farther away. Due to the rotation matrices associated with transforming the data to and from the depth sensor frame, The (0,0,0) point is aligned with the world frame origin.

An example of a depth image, and a comparison to the room, is shown in Figure 1b. Initial preprocessing was done to create point clouds of each scan using Python scripts provided by the same Microsoft Team who developed the StreamRecorder application [2].

Once processed, the grid space was segmented into several planes to create a "floor" mapping to determine the number of points in each grid cell.

B. *Occupancy Grid Map Construction*

We took our .mat file of the combined grid spaces where we had a point cloud including the x, y, and z coordinates of all hits from all of the grid spaces. We converted this file into an excel file so that we could more easily import it into our Python script. The Python script, which you can view at https://github.com/akdukaan/ecse499, begins by using the pandas read_excel method on the file which allows us to more easily read and manipulate the data. We then declared a variable called cell_count and initialized it to 10. This value represents the number of bin lengths per meter. A value of 10 means that our bins are split up into 1/10 meters, or 10cm x 10cm bins. Using a number too small resulted in large bin sizes which made it difficult to see finer details. Using a number too large resulted in small bin sizes with very few hits per bin, and this caused many bins to have an excessively high belief. We chose the number 10 because we found that it balanced the tradeoffs well. The script goes through each hit and adds 2 to both the x and y coordinates. This allows us to more easily plot the data because all of the x, y, and z coordinates are positive values. Then, the code creates three different matrices to store the number of times that a hit was in that bin. We chose to create three of these matrices because it allowed for one to perceive depth without overcrowding the plot. Hits with low z values fell into the bottom plane, hits with medium z values fell into the middle plane, and hits with high z values fell into the top plane. After constructing these matrices, we are able to plot the data to our 3D graph with the number of times that a bin has been hit correlating to the brightness of the point representing the bin. Figure 4 represents a superposition of data captured from two IR scans, with the points from each plotted in different colors (pink and green). As more scans are added, areas with high overlap (that, therefore, have a higher likelihood of being occupied) will count more hits than areas with low overlap between scans (and alternatively may indicate a false detection by the sensor).

An occupancy grid map represents the environment as a grid. It estimates the probability that a location is occupied by an obstacle by the belief equation. One way of implementing the belief equation is through the inverse sensor model or the

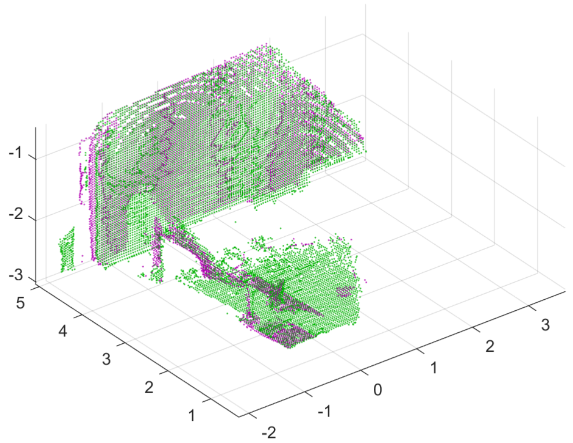

Figure 4: A plot showing point clouds from two subsequent frames plotted on top of each other, one in pink and the other in green. This figure illustrates the superposition of points as the full point cloud is constructed. These points will be counted to create our mapping of Olin 410.

log-odds representation. A simpler way is just to count the number of hits where a hit is defined as the number of cases where a beam ended in that bin and a miss is defined as the number of times that a beam passed through that bin. We can calculate the belief with the following equation:

$$Bel(\, m^{[xy]}) \; = \; \frac{hits(x,y)}{hits(x,y) + misses(x,y)}$$

We can then use this belief to determine the color of each bin.

Our implementation of the occupancy grid map is slightly different from the one described above. Our sensor does not keep track of which bins a beam passes through. Since we also don't have the position from which each reading was taken, we also cannot calculate this data. Instead, we had to assume that any reading that was not a hit in a bin was a miss. This resulted in all points having a very low belief, and thus having a very dark point in our occupancy grid. To account for this, we had to magnify the belief at each point, and we did this by multiplying all beliefs by an alpha such that the largest belief was equal to 1. One downside to this method is that our occupancy grid colors the area beyond the wall as black showing that we believe that it is unoccupied space. However, this is not a fair belief as we have no hit or miss information for this grid area. Since we have no information about it, we would assume that there is an equal probability that the space beyond the wall could be occupied or not occupied, and should therefore be colored gray.

C. Most-Likely Mapping Determination

To create a most-likely mapping determination from an occupancy grid map, one would normally clip the occupancy grid map at a threshold of 0.5.

In our implementation of the most-likely mapping determination, which you can view the source of at the same GitHub repository linked earlier, this threshold seemed too high and only resulted in a few of our bins being plotted white. This is likely due to the improper calculation of our occupancy grid map beliefs. To create the graph below, we used a threshold of 0.2 instead.

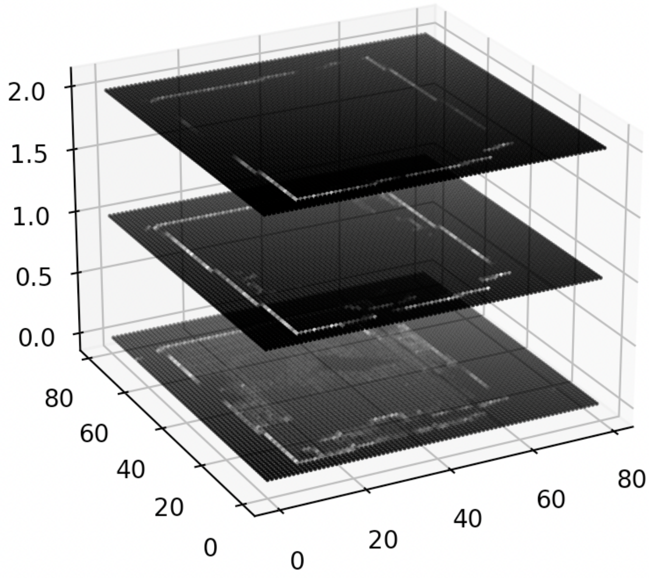

Figure 5: A 3D occupancy grid map constructed using our Python script

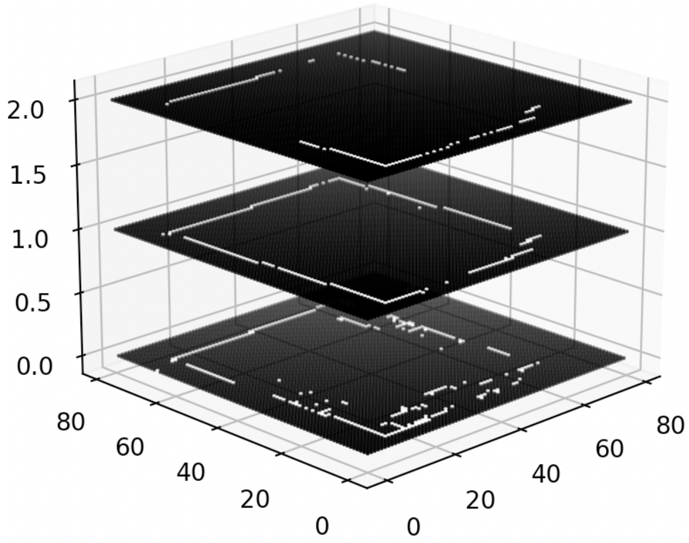

Figure 6: A 3D most-likely mapping determination constructed using our Python script.

III. Discussion & Limitations

While this project had some success creating a 3D occupancy map there are many significant limitations with our current solution.

A. Discussion of Results

In this project, we were able to successfully record, download, and process IR scanning data from the Microsoft Hololens 2 and build a likelihood mapping. This mapping was performed with real data collected on campus, and largely utilized existing hardware and software. Depth images were processed and converted to point clouds, which were then used to calculate the occupancy grid mapping based on the number of hits and misses in each sensor reading. This grid map was then used to create a maximum-likelihood mapping.

B. Offline Processing

The first limitation is that our implementation is offline only, meaning that it can't be done in real-time. While this works fine in order to collect the data and create the map, it may not be ideal in real-world applications. Since the map cannot be made using an application running onboard the Hololens 2, the data must be downloaded from the Hololens to a separate computer. This not only takes more time but also could be inconvenient if the scan data was not good. For example, if data was collected from a specific location, the user would not be able to check to see if the map was good until processing the data. Furthermore, since the download process can take a significant amount of time the user may need to go to a different location to do this, depending on the application. This means if the data was captured incorrectly or some feature of the space was missed then the user would have to revisit the location in order to recreate the map.

C. LIDAR vs IR Scans

Our implementation was also limited by the algorithms we could use given the sensor data we collected. Many of the SLAM algorithms we explored used point clouds generated from LIDAR data. However, we only had data from IR scans. Originally, we tried to convert our IR scans into 3D point cloud data, in order to adapt the LIDAR algorithms for the data from the Hololens. There were two primary challenges when trying to transform our data into something similar to a LIDAR point cloud. The first spatial transformation from the camera space of the IR scan to the three-dimensional world space proved to be difficult, this was especially because LIDAR data exists in a spherical shape, while the IR scan data exists in a grid space making the algorithm incompatible with our data. Additionally, we also had to transform each of our data frames from the camera space to the world space. While this is theoretically possible, since we had odometry data from the Hololens 2, it was difficult to properly line up and transform each data frame to make one cohesive image. We struggled with misaligned data frames. Furthermore, LIDAR data scans tend to have higher quality images than IR scans. This increased the incompatibility of our data with the algorithms that we were trying to use, as well as simply not being able to create as high-resolution images as could be done with LIDAR.

D. Counting vs Occupancy Grid Map

Another shortcoming in our implementation was in calculating the probability that each cube in our grid was occupied. The original plan was to calculate the probability of whether or not each space on the grid map was occupied for each individual scan, and then combine these probabilities to find the most likely map. However, we ended up being more successful using a simple counting algorithm. Rather than calculating the probability of each point for each scan we simply added up the number of times each point in the gridspace was hit for all of the scans. This means that our gridmap ended up being a representation more of the frequency that a space was occupied rather than the probability that it was occupied. While this works, for a crude estimate of a map, it is not as robust as calculating the probability for each scan. It is more susceptible to error in that a space may appear more likely to be occupied simply because it was visited more often than other spaces.

IV. Conclusions & Future Work

This project demonstrates a successful implementation of offline occupancy mapping using commercially-available hardware and real world sensors. While there are some limitations with the approach and a counting method may lead to a less exact occupancy mapping, this is acceptable for applications where there is low risk of hitting obstacles in the map.

Further work on this project can explore taking an increased number of slices along the z-axis of the world frame, to get higher resolution by increasing the number of horizontal planes. Additional work would explore improving the transformation of individual frames in order to better count the probability of a square being occupied. Finally, location tracking and feature extraction can be used to monitor for loops in the user's trajectory, and expand the project from mapping to a localization and mapping problem.

There are many applications for this project, some of which connect to our own research. Algorithmic mapping and path planning can be used to help people with newly acquired SCI assess maneuverability in their homes. A map of their home space, combined with a path planning algorithm, can detect locations where collisions may occur and where modifications may need to be made to assure safe operation of a wheelchair around the home. Alternatively, mapping of the home space can help people with SCI and their care team determine if certain assistive technologies can fit in the rooms where they may need to be used. Due to the shape of some assistive technology, especially patient-assisted lifts, it is critical to get 3D mapping data, as opposed to a 2D floor scan. This will allow for a more accurate depiction of the home space and the interaction between assistive technologies and their environment.

References

[1] D. Ungureanu, Dorin, F. Bogo, S. Galliani, P. Sama, X. Duan, C. Meekhof, J. Stühmer, T. J. Cashman, B. Tekin, J. L. Schönberger, B. Tekin, P. Olszta and M. Pollefeys. "HoloLens2 Research Mode as a Tool for Computer Vision Research". *arVix:2008.11239.* 2020.

[2] D. Ungureanu, Dorin, F. Bogo, S. Galliani, P. Sama, X. Duan, C. Meekhof, J. Stühmer, T. J. Cashman, B. Tekin, J. L. Schönberger, B. Tekin, P. Olszta and M. Pollefeys. "HoloLens2ForCV: StreamRecorderConverter". *GitHub Repository*.

[3] D. Ungureanu and P. Sama. "HoloLens 2 Research Mode: API Overview" *ECCV 2020 Online.* August 2020.