Room Mapping Using Motion Vectors

Julian Ewaied[1] Namir Ballan[2]

ABSTRACT

In our project we deal with how to build a 3D point cloud of a closed room using only a drone and a camera. We learned about two important algorithms for doing so: orb, which is relatively slow, and motion-vectors, which is computed using some special hardware which makes it very fast and light in calculation. Motion vectors describe the translation of a pixel from one frame to the next, and we claimed to use this sort of information along with other formulas to calculate the (x, y, z) of the point represented by this pixel as we will explain in this report.

In this project, we experimented a new path for the drone to take in order to scan the room, which is vertical rotations.

THE COMPUTATION

The inverse camera matrix is a matrix that recovers the points with normalized z coordinate.

$$\begin{pmatrix}\hat{x}\\ \hat{y}\\ 1\end{pmatrix} = \begin{pmatrix}\frac{1}{f_x} & 0 & \frac{-c_x}{f_x}\\ 0 & \frac{1}{f_y} & \frac{-c_y}{f_y}\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}x_p\\ y_p\\ 1\end{pmatrix} = \begin{pmatrix}\frac{x_p - c_x}{f_x}\\ \frac{y_p - c_y}{f_y}\\ 1\end{pmatrix}$$

Let d be the depth of the point, then

$$\begin{pmatrix}x\\ y\\ z\end{pmatrix} = d\begin{pmatrix}\hat{x}\\ \hat{y}\\ 1\end{pmatrix}$$

Consider the representation of the same point in two different frames as shown in figure 1.

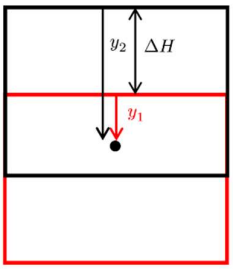

Figure1 : red frame is frame k-1, and black frame is frame k. we look at the y coordinate of the same point in two different frames, and calculate the depth using the distance

---

Submitted for Real Time Systems Lab, University of Haifa, July-August 2023, Prof. D. Feldman.

[1] Holder of i.d. 215543497.

[2] Holder of i.d. 326165156.

Notice that:

$$\frac{\Delta H}{y_2} = \frac{y_2 - y_1}{y_2} = \frac{\hat{y}_2 - \hat{y}_1}{\hat{y}_2} = \frac{y_{p_2} - y_{p_1}}{f_y \hat{y}_2}$$

And therefore, we can conclude that:

$$d = \frac{\hat{y}_2}{y_2} = \frac{f_y \Delta H}{y_{p_2} - y_{p_1}} = f_y \frac{\Delta H}{\Delta y_p}$$

Therefore, we can conclude that:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = dC^{-1} \begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} = f_y \frac{\Delta H}{\Delta y_p} \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix}$$

INTIIAL ALGORITHM

We begin by defining what we can have from the drone. First of all, we have the motion vectors of each frame, meaning, we know where does every macro-block's center (of size 16×16) moves in the next frame. We also have the height of each point frame, with accuracy of 10 cm. In addition, we are aware of the path, which is: rise, rotate 60°, fall, rotate 60°, rise again and so on until we scan the whole room. This means that we will convert each vertical movement into a cloud of points, and then rotate them according to their angle. Here is a pseudo-code of the algorithm we implemented in C++.

Initial Mapping Algorithm

For each angle file,

1. For each frame in the file,
    a. For each center c_p in the frame
        i. If $\Delta y_p = 0$, ignore it.
        ii. Else, add $dC^{-1}c_p$ to the temporary cloud *tmp*.
2. rotate *tmp* to the corresponding angle of the file around the axis $\hat{y}$.
3. Concatenate *tmp* to the total cloud P.

IMPROVEMENTS

FPS Rate

One thing we noticed once we started working with the videos is that a high fps rate gives some very bad results. After debugging and looking directly at the data we realized that with a high fps, every movement is extremely decomposed into very small movements such that the motion vectors are mostly 0's. therefore we move to 10 fps which worked properly.

Height Sensor Limitations

Our height sensor accuracy is limited to 10cm which made the results very discrete and not-obvious, besides making a lot of $\Delta H = 0$, meaning it cancels out points just because the height sensor didn't detect the change of height even though the camera did, so we looked at the

height function which seemed to be very linear if we ignore the sides of the function, thus we decided to first continuize the height function as follows.

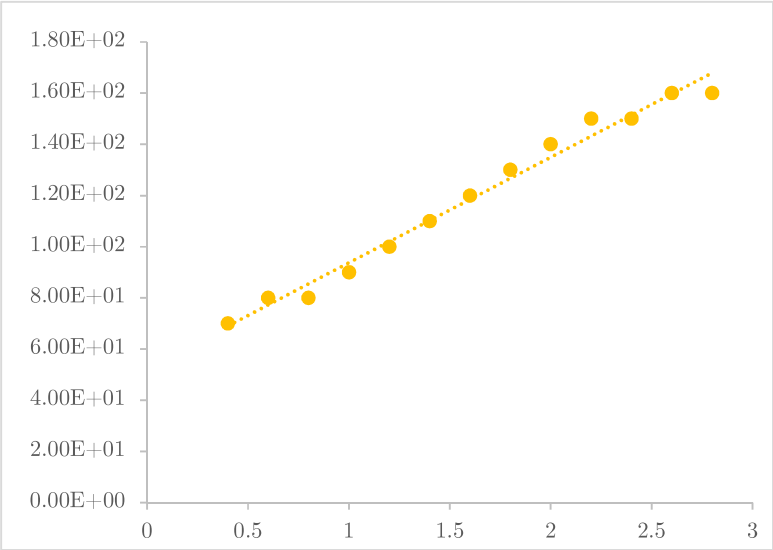

Figure 2: height function is almost linear

Continuization Algorithm

1. Set $0 \to i$.
2. Set $0 \to j$.
3. While $i < heights.size$
   a. While $heights[i] = heights[j]$, do $i + 1 \to i$
   b. Let $\frac{heigh\ \ [i]-heights[j]}{i-j} \to dif$
   c. For $j \leq k \leq i$,
      i. $heights[k] + dif \cdot (k - j) \to heights[k]$
   d. set $j = i$

and this way we have overcame the heights accuracy obstacle.

Presenting the Results

Depth Maps

We built depth maps which show how far a point is from the camera with the video playing in the background to recognize what points are problematic.

OBJ Files

We built a "mesh model" in which each point is represented by a single triangle in the mesh model. This maybe wasn't the optimal choice, but it did work and help us use an interactive environment to represent the points without implementing a whole interactive mesh-viewer. We used Microsoft 3D builder, and Maya to show the mesh models, which was very easy and convenient to use and study the problems in our code and project in general.

Matching y-Coordinate

Notice that the y-coordinates we get from the computation above is shifted, so we had to add the height of the frame to all y coordinates of the same frame.

PRIMITIVE FILTERING POLICIES

Consider the results given by the above-described algorithm with all the above-mentioned improvements. We get the following results:

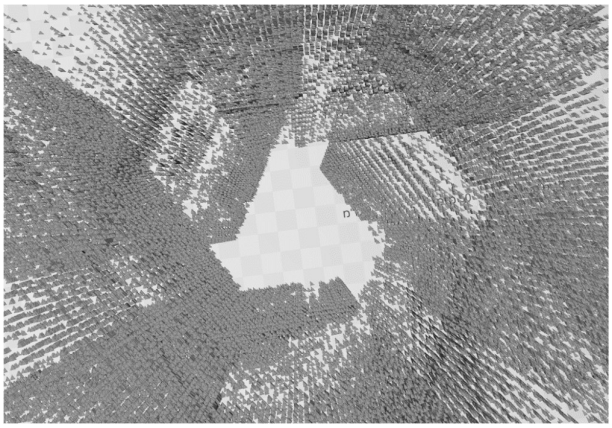

Figure 3: points we get from running the intial algorithm with the improvements

At first sight, this is incorrect, but actually, it is a correct map, just covered with a lot of outliers, thus our next part was to find policies to filter out outliers.

1. Δx_p Filtering

We tried multiple Δx_p thresholds to filter out outliers, but they all failed.

2. SAD Filtering

SAD (Sum of Absolute Differences) is defined as the squared norm of the motion vector. We applied a binary search to find the "optimal" interval of SAD values to keep, which we found as [100,125], and gave the following results.

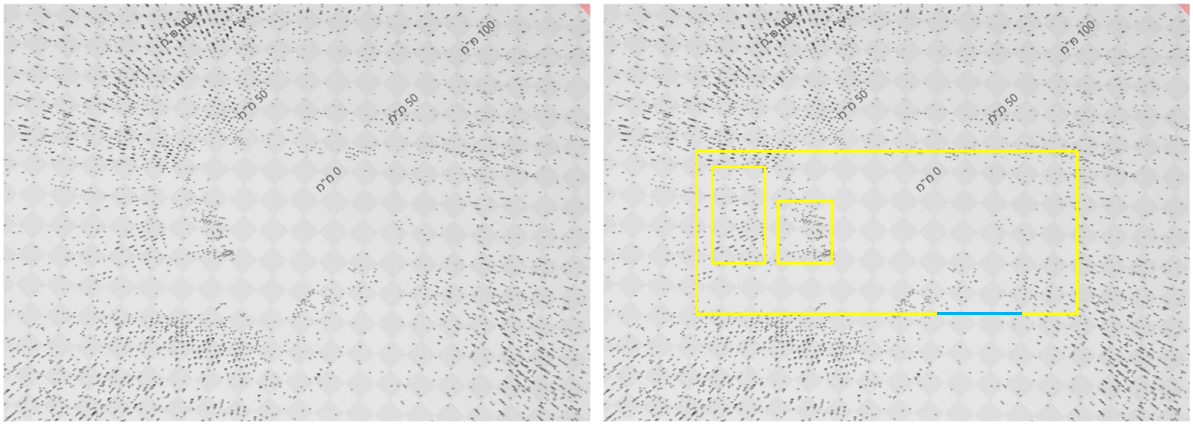

Figure 4: the results (left image), and how they should be interpreted. The yellow rectangles are a chair and a table that appeared in one of the videos.

AVERAGE COLOR ALGORITHM

This algorithm aims to filtering the floor and white walls data using the block-average algorithm described below.

Average Color Algorithm

For each macro-block y in the image:

1. sample 16 arbitrary (and not random) points, for instance, the diagonal of the macro-block.
2. Average their colors values: if $c_1, \dots, c_{16} \in [255]^3$, calculate $\bar{c}_y = \frac{1}{16}\Sigma_{i=1}^{16} c_i$.
3. If $\frac{1}{8}\Sigma_{x\in\Gamma(y)} \left|\left|\bar{c}_x - \bar{c}_y\right|\right|_2 \leq f$, where $\Gamma(y)$ is all blocks adjacent (horizontally, diagonally, and vertically) to y, and f is some global constant, get rid of the point resulted by the center of y.

This is a useful filtering algorithm, since if two blocks have almost the same color average, meaning they're useless, the difference between their color-averages is very small, and therefore we get rid of them.

However, although the room maintains its shape, it loses lots of points, which makes it hard for us to see the shape of the room, and even harder for a computer to recognize. However, we can see that at the exit it is almost a clear area, which defines a tradeoff between how clear the room borders are and how clean the exit is.

Therefore, we could already see that the tradeoff we have is very extreme that we don't need to use the color average algorithm. Our explanation to this extreme tradeoff is that macro-blocks of the same color represent together an object, which is of some size. By applying our algorithm, we are only taking the "borders" of these objects, which makes it shallower and harder to recognize with a top-down view. Here are the results using this algorithm.

Figure 5: results of color-average algorithm

MEDIAN COMPRESSING ALGORITHM

In this algorithm, we replace every radial slice with their depth median and thus minimize outliers' effect on the total shape. It can be implemented by converting the points to polar representation, then scanning them into buckets of angles, and finally using Tarjan's SELECT algorithm on each slice in $\mathcal{O}(n)$.

Unfortunately, we didn't have enough time to implement this algorithm, but another group managed to get some very good results with it (using 20% instead of 50% - median), and we assume this algorithm would also work on our dataset, giving us good results (hypothetically).

FINAL ALGORITHM

Initial Mapping Algorithm

For each angle file,

1. Continuize the height function using the algorithm above.
2. For each frame in the file,
    a. For each center c_p in the frame
        i. If $\Delta y_p = 0$, ignore it.
        ii. If $SAD \notin [100,125]$, ignore it.
        iii. Else, add $dC^{-1}c_p$ to the temporary cloud tmp.
3. rotate tmp to the corresponding angle of the file around the axis $\hat{y}$.
4. Concatenate tmp to the total cloud P.

The running time of this algorithm is approximately 0.2 seconds on an average computer, thus approximately 2 seconds on an average drone micro-processor (particularly, raspberry-pi).

FINAL RESULTS

The final results are satisfying. They are obviously not optimal and need more complex filtering policy, but we assume that by applying the median compressing algorithm we can achieve even better results. For the current time the best results we have are the results from the SAD filtering, which are the following.

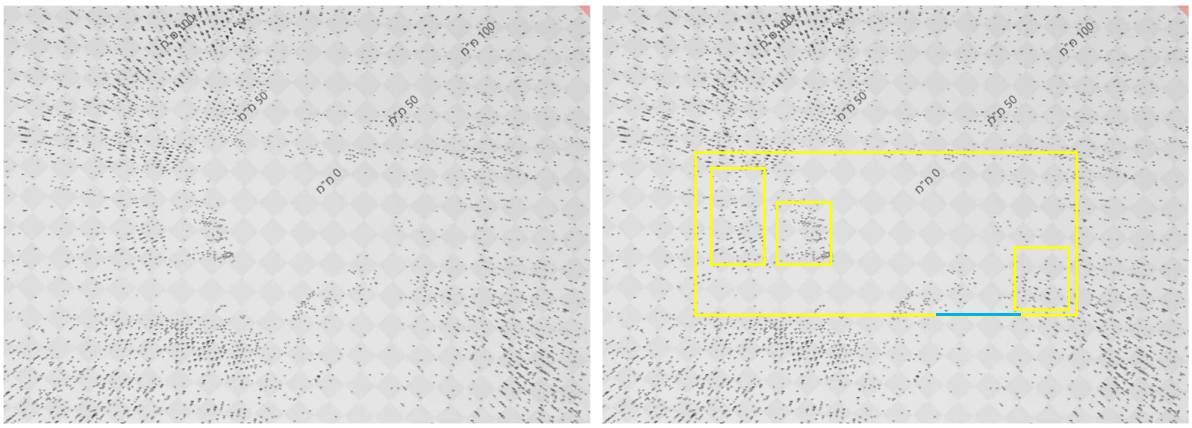

Figure 6: final results of our project for mapping the room.

We realized the "noise" on the entrance was not noise at all. By looking at the videos we can notice a chair next to the door, and a table on the other side, which made that much points next to the entrance, therefore we can conclude that the inner points cloud is correct and works in a very good resolution, concluding that our project is a success.

Figure 7: objects located next to the entrance such as the chair, the shelves and the leader make the points which appeared to be noise, as shown in the presented frames from the videos before and after the entrance video ($\pm 60°$)

PROJECT SUMMARY

The project included multiple requirements of algorithmic, mathematical and geometric thinking, along with technical skills which we had to develop independently. We wrote our project in C++, while our instructor's language is python. We overcame the obstacle of how complex it is to build a C++ repository, link it with the needed libraries and download them, and eventually handling files just to start implementing algorithms. However, we managed to achieve some very good results[3], and paved the path for futuristic researchers and students to use C++ easily by following our instructions and tutorials we packed in our reports.

Additionally, we made sure to keep our code elegant, documented, and simple for a foreign reader to understand.

FUTURISTIC QUESTIONS

Can we use the data from some room to generalize the filtering policy to other rooms? And if so, what is the best model for learning this filtering policy? This sort of question cannot be solved using classical machine learning algorithms. Even with non-linear SVM, it would be at least very complex if not impossible to find the kernel that can make the features of the points $(\Delta x, \Delta y, x, y, c)$ linearly separable, thus we will need some more complex model for this classification problem, for instance deep neural network whose final layer consists of the two neurons representing the answers "outlier" or "not outlier". Such a question requires more knowledge than we currently have, but can be left to futuristic research.

[3] See the attachments to this file – data.obj, demo.mp4, code.zip for further information on the results and the code: the mesh-model of the points, a demo of depth map, 3D cloud presentation and how the code works, and the code itself compressed into zip file.