UNIVERSITY OF HAIFA

אוניברסיטת חיפה جامعة حيفا

RT Systems Lab

Weekly Report number (2)

Contact Us

Julian Ewaied 215543497 Namir Ballan 326165156

8/4/2023

Contents

This Week's Tasks

- Running on a full room (360 degrees)
- Testing lower fps rates.
- Testing different height approximation functions.
- Testing different maximal heights
- Moving to Real Time setup.

Tasks Done

Running on full room

We started by writing the path in python, which we done using the existing wrappers Bar offered us. We got 6 h264 files, 6 csv height files, and 6 npy files. To ease our mission, we updated the python code to automatically convert whole .npy files directory into a new directory of the files with the same names. Next up, we wrote the main function as it follows:

```
void continuize(vector<double>& heights)
{
    int i = 1, j = 0;
    while (i < heights.size())
    {
        while (i < heights.size() && heights[i] == heights[j]) i++;
        if (i == heights.size()) break;
        double d = heights[i] - heights[j];
        double diff = d / (i - j);
        for (int k = j + 1; k < i;k++)
            heights[k] = heights[k - 1] + diff;
        j = i;
    }
}

vector<Eigen::Vector3d> extractPoints(string path, string heights_path,int angle)
{
    auto motionVectors = importMV(path);
    CSVFile height_file(heights_path, NUM_FRM);
    height_file.openFile();
    auto heights = height_file.readColumn();
    auto centers = getCenters();
    Analyzer analyzer(fx, fy, cx, cy);
    vector<Eigen::Vector3d> points;
    // continuize the heights function.
    continuize(heights);
    differences(heights);
    for (int i = 0;i < motionVectors.size();i++)
    {
        vector<Eigen::Vector3d> tmp = analyzer.mapPoints(centers,
motionVectors[i], heights[i]);
        points.insert(points.end(), tmp.begin(), tmp.end());
    }
    Analyzer::rotatePoints(points,angle);
    return points;
}
```

```
void showTD(vector<Eigen::Vector3d> points)
{
    string window_name = "Room Map";
    PointDisplayer displayer(window_name);
    displayer.topDownView(points);
}
int BuildTDView(vector<string> mvFiles, vector<string> heightFiles)
{

    if (mvFiles.size() != heightFiles.size()) throw "Invalid sizes in
BuildTDView";
    vector<Eigen::Vector3d> points;
    for (int i = 0;i < mvFiles.size();i++)
    {
        auto tmp = extractPoints(mvFiles[i], heightFiles[i], 60*i);
        std::cout << "Processing Angle : " << 60 * i << std::endl;
        points.insert(points.end(), tmp.begin(), tmp.end());
    }
    showTD(points);
    return 0;
}
```

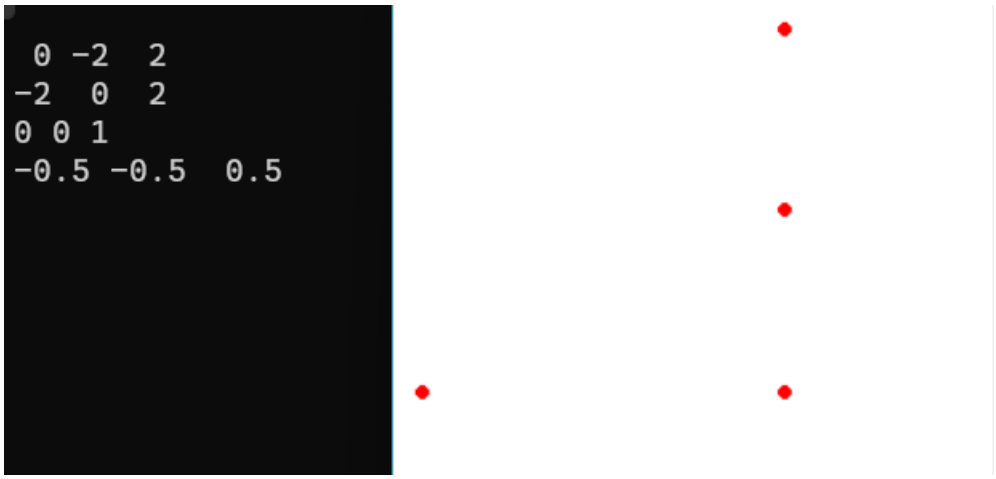
We got the following results:

```
Processing Angle : 0
Processing Angle : 60
Processing Angle : 120
Processing Angle : 180
Processing Angle : 240
Processing Angle : 300
```

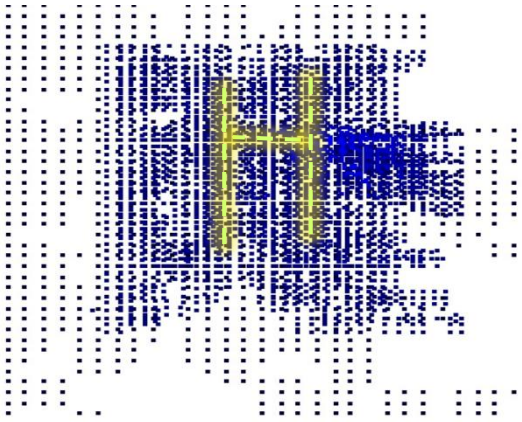
The results seemed incorrect, so we decided to recheck our code. We ran on a sanity check:

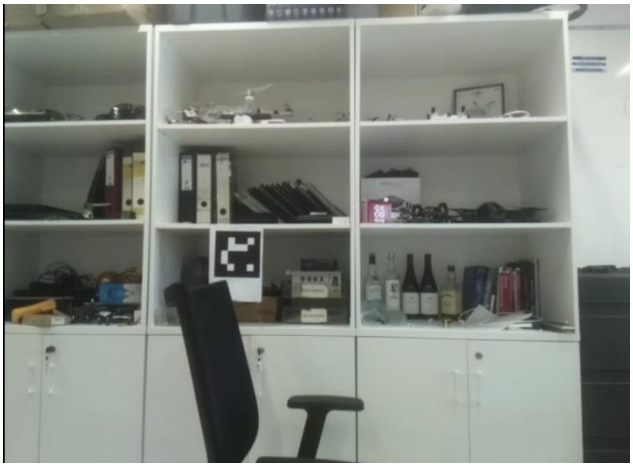
```
   vector<Eigen::Vector2d> motionVectors {

        Eigen::Vector2d(0,1),
        Eigen::Vector2d(0,1),
        Eigen::Vector2d(0,2),
        Eigen::Vector2d(0,4)
    };
    double height = 2;
    vector<Eigen::Vector2d> centers{
        Eigen::Vector2d(1,0), Eigen::Vector2d(0,1),
        Eigen::Vector2d(1,1), Eigen::Vector2d(0,0)};
```

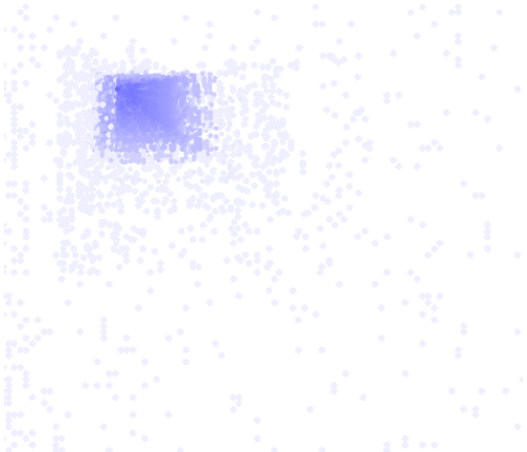
We ran the code and got the expected points:

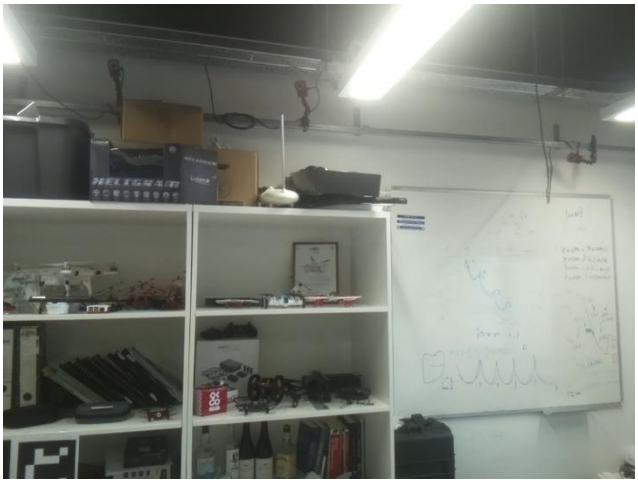


We realized the data is well processed, but we don't know the top-down view shape, meaning that it's not a useful way to check if we're doing the right calculations, so we implemented depth map, and got the following results (yellow dashed lines are lines we added to indicate the emphasized outline).









Note: in the last example, we shaded out points with depths outside the range of $\pm\sigma$ (standard deviation of the depths) from the mean μ.

In the last example, we could notice that the algorithm could recognize the shelf, but we are not sure where these points are, so our next step is to place the points on the source frames.

We must mention that we used 5 fps rate, and moved up to 90 cm (height range of 70-160 cm), which allowed better data values distribution (less 0 motion vectors), and did give better results.

Testing Different Height Approximation Functions

Up until now, we have overcome the sensor resolution limitation using linear approximations, meaning,

$$\frac{\partial H}{\partial y} = c \in \mathbb{R}$$

For each height interval. This is obviously not realistic, so we want to try other functions.

Let $H: \mathbb{R}^+ \to \mathbb{R}$ be a function that given t a specific moment in seconds, outputs the height of the drone. Let ΔH_i be the corresponding height difference to Δt_i, and let $\{t_i\}_{i=1}^k$ denote the t moments with a distinct height measurement than its predecessor, corresponding to heights $\{h_i\}_{i=1}^k$ we require:

1) H is a monotonic function per file.
2) $\sum_{t_i<t<t_{i+1}} \Delta H = h_{i+1} - h_i$.

A good function to start with is quadratic approximation, which matches "constant" acceleration of the drone. We will be using the parabola approximation:

$$H\ y\ = \frac{\alpha_i}{2} t^2 + h_0, \qquad y_i < y < y_{i+1}$$

Which matches a constant acceleration α_i, and initial location h_0. This means that:

$$\Delta H\ y\ = \alpha_i t, \qquad y_i < y < y_{i+1}$$

Thus, giving us:

$$\sum_{y_i<y<y_{i+1}} \Delta H = \alpha_i \sum_{t_i<t<t_{i+1}} t = h_{i+1} - h_i$$

$$\alpha_i = \frac{h_{i+1} - h_i}{\sum_{t_i<t<t_{i+1}} t}$$

A good approximation of t_i is $t_i = \frac{i}{5}\ sec$, since our fps rate is 5fps.

Next Week's Tasks

- ☞ Matching between two representations of the same point in two different frames
- ☞ Making the code more elegant
- ☞ Moving to Real Time Setup.