

RT Systems Lab

**Weekly Report number (2)**



Contact Us

Julian Ewaied 215543497 Namir Ballan 326165156

Contents

[This Week's Tasks 3](#_Toc141385865)

[Tasks Done 3](#_Toc141385866)

[Understanding Camera Matrix 3](#_Toc141385867)

[Installing OpenCV and Eigen3 libraries for C++ 5](#_Toc141385868)

[Recovering Original Coordinates 6](#_Toc141385869)

[Considering Improvements 8](#_Toc141385870)

[Next Week's Tasks 8](#_Toc141385871)

# 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.

## 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;

int 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());

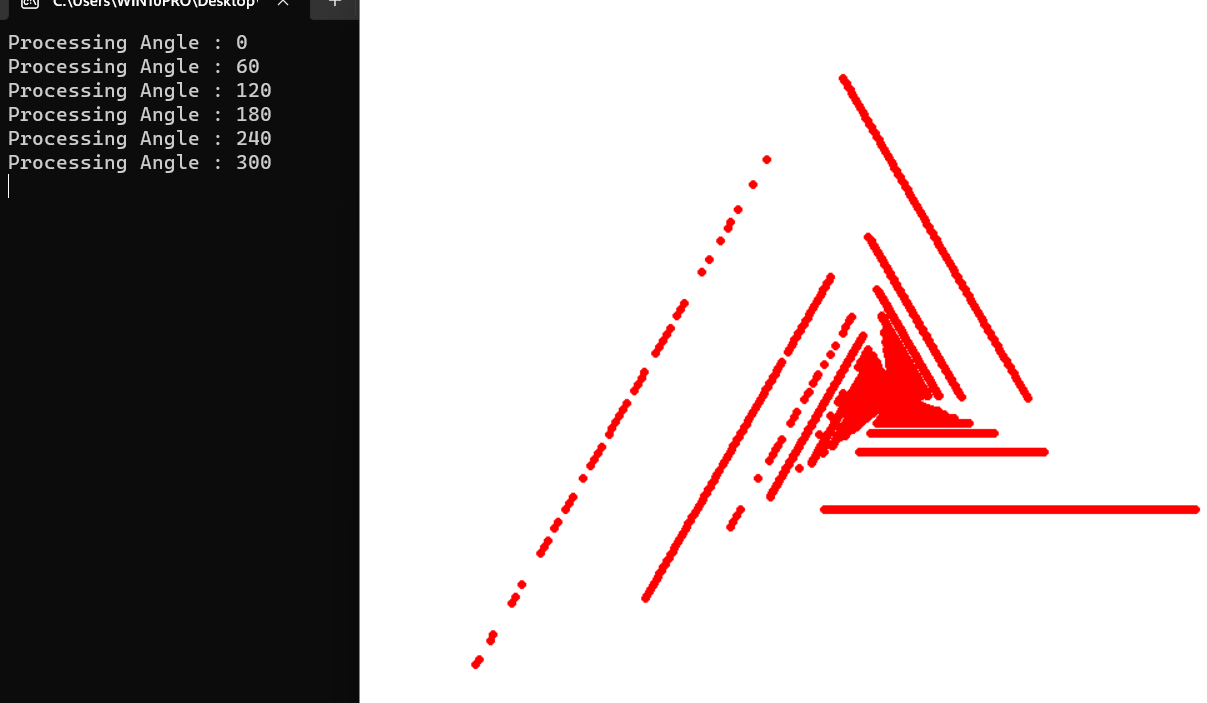
}

showTD(points);

return 0;

}

We got the following results:



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;

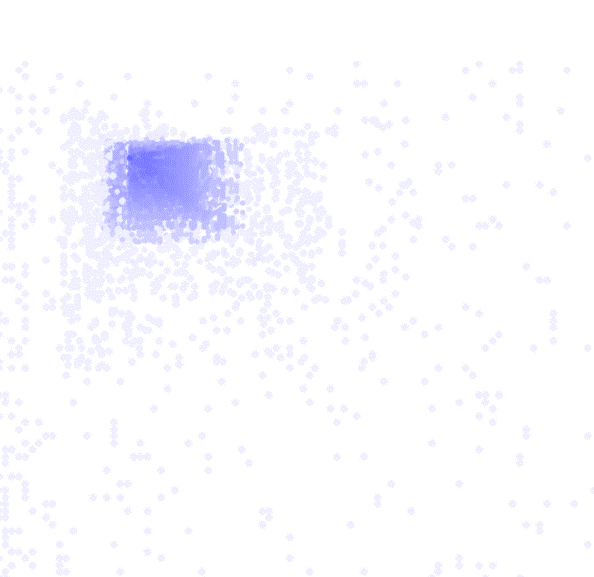
vector<Eigen::Vector2d> centers{

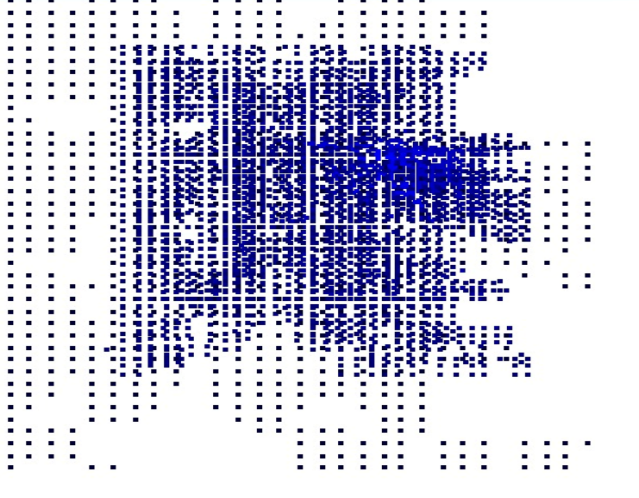
Eigen::Vector2d(1,0), Eigen::Vector2d(0,1),

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 (standard deviation of the depths) from the mean .

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,

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

Let be a function that given of a specific point in the picture, outputs the height of the drone. Let be the corresponding height difference to , and let denote the coordinates with a distinct height measurement than its predecessor, corresponding to heights we require:

1. is a monotonic function per file.
2. .

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

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

Thus, giving us: