UNIVERSITY OF HAIFA

אוניברסיטת חיפה جامعة حيفا

RT Systems Lab

Weekly Report number (3)

Contact Us

Julian Ewaied 215543497 Namir Ballan 326165156

8/9/2023

Contents

This Week's Tasks

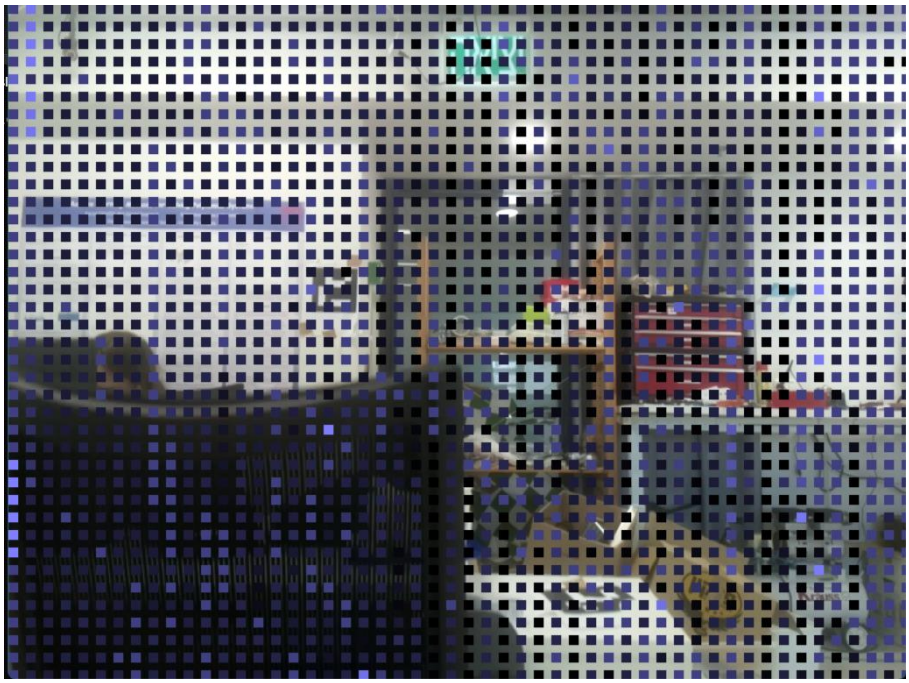
- Using safe-environment presentation of points cloud (MAYA or MS 3D Painter) with obj files.
- Building more interactive depth map.
- Further debugging of the code.
- Matching between two representations of the same point in two different frames
- Making the code more elegant
- Making the code more efficient

Further Debugging

We debugged the code further, but didn't find issues neither in the computation itself, nor in the reading, so we assume 2 issues might be present: presentation, and combination of points clouds (rotation and matching points in different frames). Another issue that might have been in the way is the data resolution, and outliers, of which we will need to think of other solutions

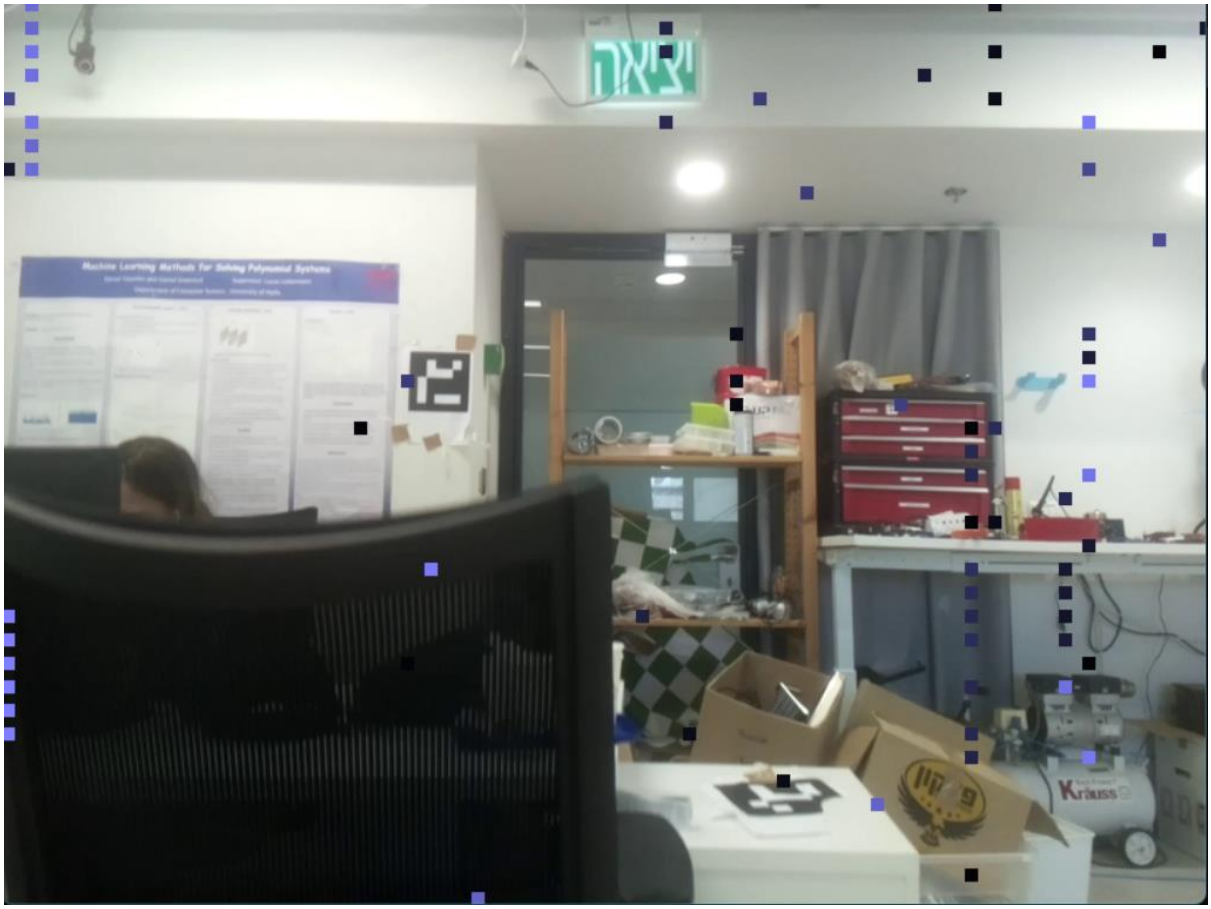
Building more interactive depth-map

We used OpenCV to read the frames all at once, then drew rectangles over the center with the color proportional to their depth (in this case we used Δy instead of $\frac{f_x \Delta H}{\Delta y}$, since $f_x \Delta H$ is constant per frame), and to make an easier working environment, we made the frames go forward when we click on W, and down when we click S. We got the following results:



We could already see good results, but so many outliers, so we decided to filter out points using SAD values, or dx values. Starting by dx we found out it doesn't work properly, so we moved to SAD,

where we require $SAD \in [a, b]$ such that $a > 0$. We tried some values of a, b and this is the result for $[a, b] = [100, \infty]$.



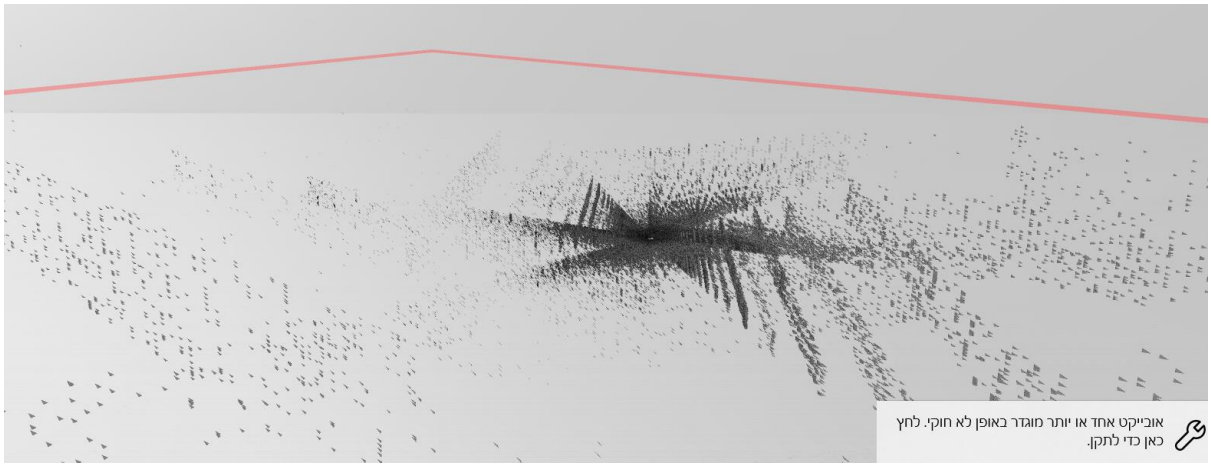
We can already see the filtering is good. We will be studying how to automatically choose the filtering criteria next week.

Presenting 3D results using 3D Builder

We went over the points, and wrote in the format: v x y z the point shifted in 1 coordinate each time. Then we wrote the faces built on the 3 copies of the vertex. Here's the code we used:

```
    for (auto point : points)
    {
        string s = "f ";
        for (int k = 0;k < 3;k++) {
            int A = 20;
            out << "v " << point(0) + ((k%3)/2)*A << " " << point(2) +
(((k+1)%3)/2)*A << " " << point(1) / 10 + A*((k+2)%3)/2 << std::endl;
            s = s + std::to_string(line++) + " ";
        }
        faces.push_back(s+"\n");
        //if (line >=10) break;
    }
    for (auto f : faces)
    {
        out << f;
    }
```

We got the following results:



We can still see problematic results, yet more suspiciously problematic rotations. We attach the obj file to our report if you would like to check it out.

Making the code more elegant

We reordered the code in which all the interface has become with the class PointDisplayer. We moved all computation functions to Analysis.cpp, and all projection functions to PointDisplayer.cpp. We only kept 2 significant functions: countFile, and writeOBJ, which are used for further use cases other than just displaying data.

Here's a summary of the API we built:

```
class Analyzer
{
protected:

        double fx, fy;
        int cx, cy;
        bool matrixBuilt;
        Eigen::Matrix3d CameraMatrix;


        const Eigen::Matrix3d& getCameraMatrix() const;

        static Eigen::Matrix3<double> getRotationMatrix(double theta, const Eigen::Vector3d& axis);

        // mapping into z-normalized map
        vector<Eigen::Vector3d> mapNormalizedPoints(const vector<Eigen::Vector2d>& points);

public:

        Analyzer(double fx = 1, double fy = 1, int cx = 0, int cy = 0) : fx(fx), fy(fy), cx(cx), cy(cy),
CameraMatrix(Eigen::Matrix3d::Zero()),matrixBuilt(false) {};

        // returns C^-1 where C is the camera matrix.
        const Eigen::Matrix3d& buildCameraMatrix();

        // returns a list of MV for each frame.
        static vector<frames> importMV(const string& path);

        // returns points of a single video
        vector<Eigen::Vector3d> extractPoints(string path, string heights_path, int angle);

        static void rotatePoints(vector<Eigen::Vector3d>& points, double angle, const Eigen::Vector3d& axis =
Eigen::Vector3d(0, 1, 0));

        vector<double> getDepths(const frames& mv, double dH);
```

```
        // given a frame and its mv, with height difference, return the full map, without rotation
        // assumes center[i] matches mv[i].
        vector<Eigen::Vector3d>  mapPoints(const vector<Eigen::Vector2d>& centers, const
vector<Eigen::Vector2d>& mv, double dH);

        // returns the center of the frame!
        static vector<Eigen::Vector2d> getCenters();

        // fixing sensor inaccuracy
        static void continuize(vector<double>& heights);

        static void differences(vector<double>& vec);
};


class PointDisplayer {
        const int CIRCLE_RADIUS = 2;
        const int HEIGHT = 700;
        const int WIDTH = 700;
        string window_name;

        void fitPoints(vector<Point2i>& points) const;
        void displayPoint(const Point2i& point, cv::Mat img) const;

public:
        PointDisplayer(string& window_name);

        void display(const vector<Eigen::Vector2d>& points) const;
        void display(const vector<cv::Point2i>& points) const;

        void topDownView(const vector<Eigen::Vector3d>& points) const;


        // builds top-down view of all the files and plots it
        static int BuildTDView(vector<string> mvFiles, vector<string> heightFiles);

        // builds depth map of one file and plots it
        static void BuildDepthMap(const string& path, const string& videoPath,
vector<vector<double>> sads);

};
```

Making the code more efficient

We start off by doing a single profiling, and attaching the profiler report. In the following weeks we will be working on 2 threads: optimizing running time, and optimizing accuracy.

We can already recognize that the heaviest part of the code is kernel time(reading from files - about 90% of the running time), meaning, the running time if we were to use this code on my machine, without the reading/writing code, would be 10 times faster, while it takes about 2 seconds on a decent OS, here's the profiler results

Function Name	Total CPU [unit, %]	Self CPU [unit, %]
RTS Lab (PID: 23136)	345 (100.00%)	0 (0.00%)
[System Code] ntdll.dll!0x00007ffa8ac8485b	345 (100.00%)	0 (0.00%)
__scrt_common_main_seh	345 (100.00%)	0 (0.00%)
Run	343 (99.42%)	0 (0.00%)
PointDisplayer::BuildTDView	343 (99.42%)	0 (0.00%)
Analyzer::extractPoints	331 (95.94%)	0 (0.00%)
Analyzer::importMV	313 (90.72%)	0 (0.00%)
CSVFile::readFile	313 (90.72%)	0 (0.00%)
CSVFile::readNext2	313 (90.72%)	0 (0.00%)
[System Code] msvcp140.dll!0x00007ffa79f298b0	81 (23.48%)	81 (23.48%)
std::basic_stringbuf<char,std::char_traits<char>,std::allocator<char> >::_Init	75 (21.74%)	1 (0.29%)
[System Code] msvcp140.dll!0x00007ffa79f23bf1	75 (21.74%)	75 (21.74%)

However, we still have 10% to run faster, besides all the memory loss we have on the way, and that would be added to our tasks from next week and after.

Next Week's Tasks

- ☞ Checking the rotation is done well
- ☞ Matching between two representations of the same point in two different frames
- ☞ Making the code more efficient
- ☞ Automating the filtering policy

Attachments to this file:

- data.obj, the file of the 3D model of the points

- Depth Map.mp4 a video of depth map implementation