Puzzle games (like Rubik's cube) solver

Vitalii Zakharov
University of Tartu
vitaliiz@ut.ee

1. INTRODUCTION

This project is based on multiple projects: the first one is Sudoku Solver by Artjom Lind and his fellows and my own approach of the same task; the second one is my own approach of Rubik's cube detection and extraction of data for cube reconstruction and solving; and the last one but not the least one is Bootstrapping planar AR and tracking without markers [1]. Since in the previous work it was managed to extract cube faces from the video stream this time the goal was set to do the same but with different approach that will help me to expand abilities of the program, for instance, make the program to be able to extract, track, and analyze any figure, not only a cube. To complete my goals I analyzed different source and PTAM (Parallel Tracking and Mapping for Small AR Workspaces) [2] was somehow an answer to the current problem. Since this algorithm was too complicated to write it by myself and running C++ Linux code under .NET platform was not an easy task too there was found another approach [1] which behaves the same as PTAM but in much simpler way. In this project running this algorithm was a goal and it was successfully gained. Since this project is a part of the master thesis, full project goals are covered in this paper.

2. RELATED WORK

There was not found any projects that have the final goal as mine, in this chapter "Bootstrapping planar AR and tracking without markers", which is PTAM like approach, is covered as a part of the final project.

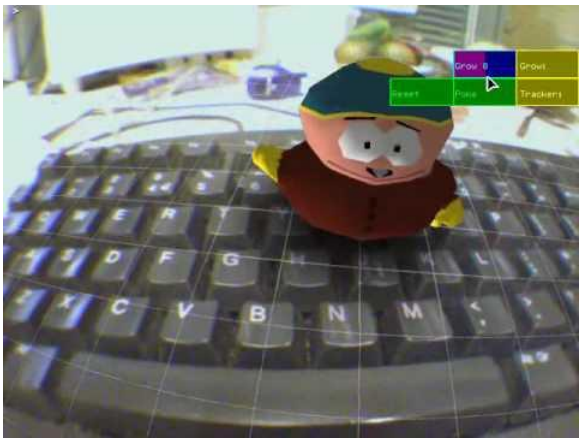

Fig. 1. PTAM example.

2.1 Bootstrapping

This step is the first one and the most simple. It just stores initial data, for instance, extracted key points of the first image, the first image in grayscale and sets the pipeline to move on. All the magic happens on next step.

2.2 Bootstrap tracking

This part of the algorithm is considered to be the most important. It should analyze images from video stream by comparing it to the first image from bootstrap step. As a result, we will get 3d point cloud, basically its simplified version. Since we now know the 3d structure of the object we can project objects on our detected plane. Since this algorithm is made for augmented reality solutions its goal is projection of different models onto the flat surface with correct perspective.

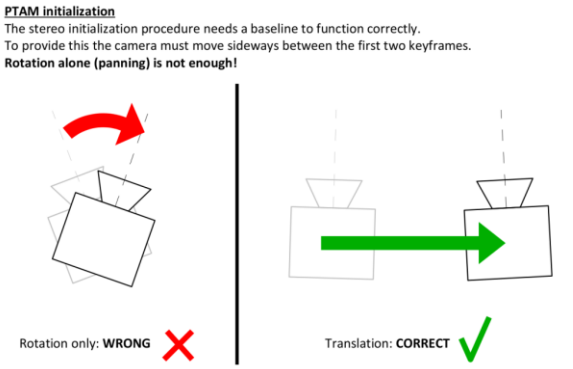

Fig. 2. PTAM like algorithm bootstrapping (initialization) rule.

This step consists of 4 major sub steps:

2.2.1 Optical flow and homography filtering

In order to know where the key points of the previous image will appear on the next image algorithm uses Optical flow algorithm.

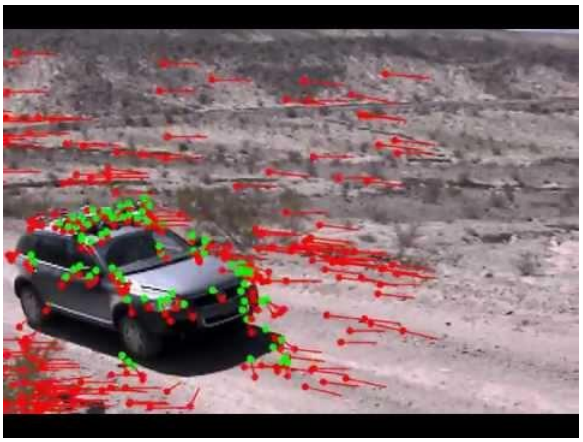

Fig. 3. Optical flow example (lines show optical flow of key points from the initial image to the current one).

After that it checks if 80 percent of points have survived, if not it stops bootstrapping and says that tracking failed. If this condition is satisfied and we it is tracking more than 4 points algorithm applies homography, it takes the bootstrap key points and tracking ones. Using mask, it checks how many points survived homography, so called inliers, and filters the lists of bootstrapped and tracking key points again so that these lists contain only the most precise key points. Then it analyzes camera motion using key points from the bootstrap step and currently tracking ones, it checks sufficient motion with OpenCV's function estimateRigidTransform. If all the conditions are satisfied it moves to the next step.

2.2.2 Extracting essential matrix from fundamental

During this step it finds fundamental matrix using bootstrapped and tracked key points. In computer vision, the fundamental matrix is a 3 by 3 matrix that relates corresponding points in stereo images [3]. Then it filters key points again. The next step is computing Essential matrix. In computer vision, the essential matrix is a 3 by 3 matrix, with some additional properties, which relates corresponding points in stereo images assuming that the cameras satisfy the pinhole camera model [4]. The algorithm needs to get intrinsic parameters of the camera on its setup since it does not include calibration part. Intrinsic parameters are physical camera parameters, like focal length and so on. To compute essential matrix, it multiplies transposed intrinsic matrix by fundamental matrix and multiplies this by intrinsic matrix, not transposed.

$$E = K'^{T} * F * K$$

Fig. 4. Essential matrix formula K and K' being the intrinsic calibration matrices of the two images involved).

This matrix is needed to extract camera rotation and translation in relative perspective from bootstrapped image to the current one. In order to do this, it decomposes essential matrix with SVD (Singular value decomposition) [5].

$$E = U * \Sigma * V^{T}$$

Fig. 5. An SVD of E
(where U and V are orthogonal3 by 3 matrices and Sigma is a 3 by 3 diagonal matrix).

Decomposition results into 2 possible rotation matrices and 2 possible translation vectors. If the determinant of first rotation matrix plus 1.0f is less than 1e-09 than algorithm changes signs of all numbers in essential matrix and decomposes it again due to "Showing that it is valid" chapter of

essential matrix wiki page [4]. Then it proceeds to the next step.

2.2.3 Triangulation of points.

During this step it assumes that the first image (camera) extrinsic parameters matrix and the second one is constructed from rotation matrices and translation vectors. The resulting matrix is 3 by 4. Since there are 2 rotation matrices and 2 translation vectors algorithm creates 4 possible extrinsic matrices for the second camera and tries to triangulate until the correct one is found or fails if none found. In order to triangulate, it normalizes bootstrapped and tracking key points coordinates. Then it proceeds triangulation and computes status by checking z component of points. It filters the key points by this status array. After that it computes reprojection error. If the error is not in the acceptable range, it runs triangulation again with different sets of rotation and translation components. When error is acceptable it goes to the next step where it finds a plane.

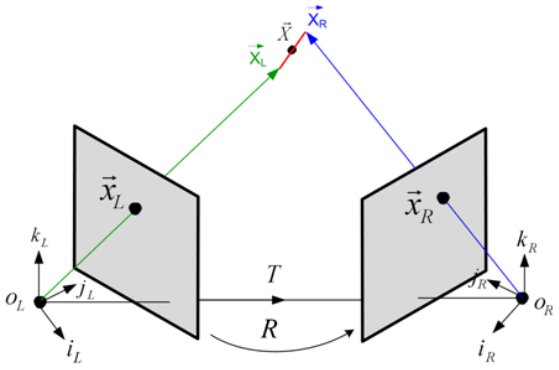

Fig. 6. Triangulation visualization.

2.2.4 Finding plane using 3d point cloud.

This step contains PCA (Principal component analysis) [6]. It is used to extract normal of plane and proceed key points filtering again. If more than 75 percent of points are on the same plane bootstrap tracking is considered to be finished.

2.3 Tracking

This step is the final for this algorithm. It simply calculates optical flow from the point when triangulation succeeded to the current image. Then it solves PnP (Perspective-n-Point) [7] in order to calculate Model-View matrix for OpenGL rendering. That is it, simple augmented reality.

2.4 Conclusion

The main difference of this algorithm from PTAM [2] is that it is tracking all captured plane points even if they disappear from camera view when you put camera back it still sees that points. On the contrary, described approach simply cuts down all unseen key points until the minimal amount exists, when it is not satisfied it breaks.

3. MY APPROACH

Since this seminar's project is a part of my master thesis I have not created my own approach. On the contrary, I have implemented my own version of the described algorithm in C# and .NET platform.

4. RESULTS

I did manage to completely finish implementation and testing of this algorithm. During the process I have received results that differ from the C++ version. I have discovered that some methods of OpenCV produces slightly different result for input matrices that have equal numbers with difference for about 0.0001. I took C++ input and tested all the EmguCV (.NET wrapper of OpenCV) methods with C++ input, it produced exactly the same, correct, result. From this perspective I observe a bit different results in numbers between these two implementations but the same correct logic. Algorithm has been choosen as a prefereble option for current problem in master thesis according to observations made during the implementation, it successfully found a plane, its normal and 3d points that form this plane.

5. FUTURE WORK

There are a lot of items on my list of future plans since its part of master thesis. In this chapter detailed description of working plan is covered.

5.1 Plane intersection

In the previous chapter plane was successfully found and analyzed. The next logical step is to find planes intersections that basically are corners of our puzzle game. Mathematically it is easy to find intersection of 3 planes so for this case we just need to take 3 planes from the list and their normals have to be all in different directions, along 3 different axis. When all corners of the puzzle game are found we can proceed further.

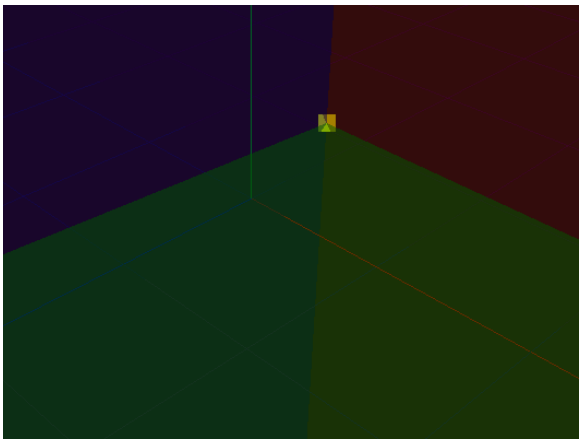

Fig. 7. Intersection of 3 planes.

5.2 Face color extraction

Run the recorded video from plane detection step and proceed warp perspective using projections of our found corners onto the current image frame. In the it produces images of faces that can be split in some specific for game way and colors can be extracted.

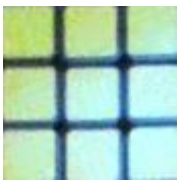

Fig. 8. Extracted face of a cube.

Fig. 9. Segmented face item (a piece of a face from which it is possible to extract color for puzzle reconstruction).

5.3 Puzzle game reconstruction

Since game corners and colors of the pieces are already known it is possible to render the model. It will be rendered with OpenGL.

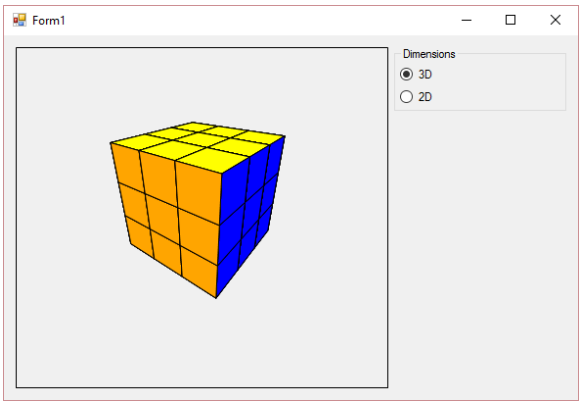

Fig. 10. Rendered cube model (final result of reconstruction from video stream).

5.4 Puzzle game tracking

Described above algorithm is designed to track plane in order to put augmented reality thing on it but in this case it will be used to get rotation and translation of the plane and with this data it is possible to manipulate rendered 3d model of the game on the screen,

5.5 Puzzle game solving

The final goal is to solve the puzzle game. Since there are a lot of different games in the world, thesis is assumed to be a framework with possibilities to write custom pluggable modules with logic for specific games in order to detect and solve them correctly.

6. CONCLUSION

To sum up, I have implemented great algorithm for augmented reality. Working with this project gave me opportunity to get more knowledge in

this field and I really found that interesting. Now, I want to use my knowledge to complete the project and probably something new in the field of augmented reality.

7. ACKNOWLEDGMENT

I would like to express my special thanks of gratitude to my supervisor Artjom Lind as well as the head of Distributed Systems research group professor Eero Vainikko who gave me good opportunity to do this wonderful project, which also helped me to learn about so many new things.

8. REFFERENCES

1. Bootstrapping planar AR and tracking without markers (http://www.morethantechnical.com/2015/03/16/bootstrapping-planar-ar-tracking-without-markers-wcode/)
2. PTAM (http://www.robots.ox.ac.uk/~gk/publications/KleinMurray2007ISMAR.pdf)
3. Fundamental matrix (computer vision) (https://en.wikipedia.org/wiki/Fundamental_matrix_(computer_vision))
4. Essential matrix (https://en.wikipedia.org/wiki/Essential_matrix)
5. Singular value decomposition (https://en.wikipedia.org/wiki/Singular_value_decomposition)
6. Principal component analysis (https://en.wikipedia.org/wiki/Principal_component_analysis)
7. Perspective-n-Point (https://en.wikipedia.org/wiki/Perspective-n-Point)