Puzzle games (like Rubik's cube) solver

Vitalii Zakharov
University of Tartu
vitaliiz@ut.ee

1. INTRODUCTION

This project is a continuation of the PTAM (Parallel Tracking and Mapping for Small AR Workspaces) like approach solution [1]. Basically, it turned to be not an appropriate solution for recognizing full cube under not ideal illumination. Since the process might fail during the processing of any face out of six, it has to start from scratch, thus it will not be finished at all. On the contrary, this algorithm gave a better understanding of Tracking and Mapping problem as well as Structure from Motion problem. At the current stage it was decided to use approaches from Sudoku Solver and First version of Rubik's cube face extraction to reconstruct cubes model. At the same time PTAM can be used to track cube motion but not rotation. It was also decided to use Kalman filter to optimize PTAM work and to use it for motion prediction because of its nature. This project also contains self-written Rubik's cube 3D model rendering with the newbie solution method programmed to solve the cube in real time. This means that program can reconstruct cube from the video, not ideally, it might take some time and a lot of rotations until the cube is recognized, and solve it while displaying all rotations on the 3D model and writing all movements as a text to the text box.

2. RELATED WORK

Since there wasn't found any projects that have the final goal as mine, in this chapter Kalman filter is covered as a part of final project.

2.1 General information

Kalman filter can be used in any dynamic system where some uncertain information exists. This filter is making educated guesses about the next step of the system based on mathematical model of the system. Even when noisy data interferes with the real measurements Kalman filter does a great job on guessing what is really going on in the system.

Due to the nature of Kalman filter it is an ideal tool for dynamin, changing systems. The reason for that is it does not require huge amounts of memory or computational power since it only needs to store previous step state in order to compute the prediction and current measurements to adjust current system state. All of that is done by a set of mathematical equations thus it makes Kalman filter a great tool for real time or embedded systems [3].

2.2 Kalman filter vision

Kalman filter requires mathematical description of the system with its parameters and it assumes that these parameters are Gaussian distributed and random. Every parameter has the most likely state which is a mean of the random distribution and range of uncertainty values, variance. One more important thing about parameters is that they must be correlated which means that one parameter can be used to find the other one. Basically, Kalman filters aims to get as much information from uncertain measurements as possible [2].
Kalman filter describes correlation of parameters with so called covariance matrix. This matrix is symmetric and its values are degrees of correlation between two parameters states where these parameters are taken depending on the indices of the matrix value. Covariance matrix is often labelled as $\mathbf{\Sigma}$ [4].

$$\hat{\mathbf{x}}_k = \begin{bmatrix} \text{position} \\ \text{velocity} \end{bmatrix}$$
$$\mathbf{P}_k = \begin{bmatrix} \Sigma_{pp} & \Sigma_{pv} \\ \Sigma_{vp} & \Sigma_{vv} \end{bmatrix}$$

Fig. 1 Example of covariance matrix with position and velocity parameters.

As a next step, filter must somehow predict the next step depending on the previous one without knowing which state is real and which is not. This is done by prediction matrix F which should transform estimation of one state to the predicted one assuming that the original state was real. This manipulations leads to covariance matrix changes, if all values in the distribution are multiplied by prediction than new covariance matrix will be computed as multiplication of prediction matrix by covariance matrix and by transposed prediction matrix.

$$\hat{\mathbf{x}}_k = \mathbf{F}_k\hat{\mathbf{x}}_{k-1}$$
$$\mathbf{P}_k = \mathbf{F_k}\mathbf{P}_{k-1}\mathbf{F}_k^T$$

Fig. 2 Example of computation the next step state and covariance matrix equations.

2.3 Extraneous known influence

In the real world it might happen that the system is affected by something not related to the systems inside world, for instance, hole in the road that will affect the way car goes or the wind blowing at some region. This knowledge of some external factors that may affect system can be described mathematically and added to the prediction calculation as a correction. They are expressed as control vector and matrix multiplied together. Control vector contains known outside parameters, for example, speed of the wind, and control matrix contains changing parameters, for example, time delta between two moments on the timeline.

$$\hat{\mathbf{x}}_k = \mathbf{F}_k\hat{\mathbf{x}}_{k-1} + \mathbf{B}_k\vec{\mathbf{u}_k}$$

Fig. 3 Example of expanded system state prediction formula.

2.4 Extraneous unknown influence

It also might happen that factors of influence are not known and they can't be described as control matrix and vector. In this case it is possible to model some uncertainty around the predicted state. This influence is treated as noise with some covariance but because of multiple possible next step noisy predictions they are treated as a single Gaussian blob with different covariance but same mean.
Thus, in order to get expanded system covariance matrix that takes into consideration possible noise it is needed to add noise blob covariance to the system covariance.

$$\mathbf{P}_k = \mathbf{F_k}\mathbf{P}_{k-1}\mathbf{F}_k^T + \mathbf{Q}_k$$

Fig. 4 Example of expanded system covariance matrix equation.

The new uncertainty is predicted with the previous one plus some correction on current uncertainty from the environment.

2.5 Prediction correction with measurements

System may contain several sensors that are giving some indirect information about its state. Data from sensors won't be the same as data tracked by Kalman filter. One fact about filter is that it is very good for handling sensor noise. Here comes sensor matrix that transforms the predicted state to a sensor reading prediction. The next step is computing the most likely state based on predicted state Gaussian distribution and sensor reading Gaussian distribution. To get the most likely state it needs to understand if two probabilities are both true. The first one is that sensor reading is estimated measurement and the second one is that previous estimate is the reading that should come from sensor. To get the most accurate estimate it multiplies those two Gaussian blobs and receives their intersection which is the estimate and the best guess. This new estimate is

the Gaussian blob with its own mean and covariance. To get these mean and covariance matrix two formulas are used (fig. 5).

$$\vec{\mu}' = \vec{\mu_0} + \mathbf{K}(\vec{\mu_1} - \vec{\mu_0})$$
$$\Sigma' = \Sigma_0 - \mathbf{K}\Sigma_0$$

Fig. 5 Example of the new mean and covariance matrix equations.

K is a Kalman gain that is computed from modeled sensor matrix H_k, system covariance matrix P_k, and the covariance of uncertainty (sensor nose) R_k.

$$\mathbf{K}' = \mathbf{P}_k\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T + \mathbf{R}_k)^{-1}$$

Fig. 6 Example of Kalman gain equation.

$$\hat{\mathbf{x}}_k' = \hat{\mathbf{x}}_k + \mathbf{K}'(\vec{z_k} - \mathbf{H}_k\hat{\mathbf{x}}_k)$$
$$\mathbf{P}_k' = \mathbf{P}_k - \mathbf{K}'\mathbf{H}_k\mathbf{P}_k$$

Fig. 7 Example of the final estimation equations.

From fig. 7 z_k is the mean of the estimated sensor distribution and it is equal to the reading from the sensor.

All of these manipulations resulted into the update stage of Kalman filter.

2.6 Conclusion

To implement linear Kalman filter it is basically needed to only implement predict and update equations. For nonlinear systems Extended Kalamn filter is used. It just linearize measurements and predictions of Kalman filter about their mean.

3. MY APPROACH

At this stage of the project the full working pipeline was done even though it is not yet optimized with known and described techniques.

3.1 Identifying unique faces order

After the identifying of 6 unique faces is done there should be some way to determine faces order. It means that each face has its own position on the cube in order to make it consistent.
To achieve this two steps are required. The first step is to analyze faces central pieces, as it is known, these items are not shuffled when the cube faces are rotating. After this step the program knows faces position but does not know face orientation. Here come the second step, there is a method for checking the Rubik's cube for integrality thus it is run for different combinations of faces orientations until the cube is recognized as consistent.

3.2 Building logical and 3D model

There was implemented the primitive 3D rendering mechanism for simple meshes using .NET Windows Forms and C# [6,7]. From the other side, logical model of the Rubik's cube was developed. All rotations, flips, shuffling and other operations are performed on this logical model while 3D model is built from logical at each rendering step.

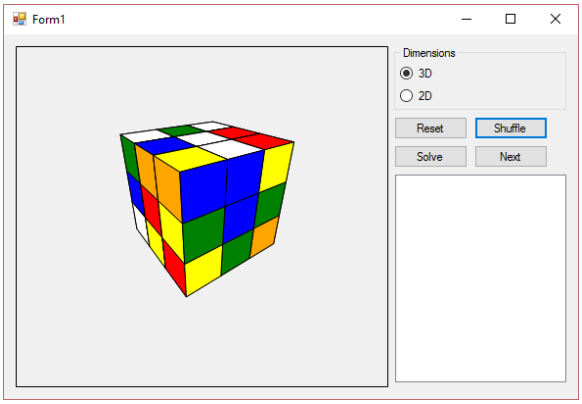

Fig. 8 Example of rendered shuffled logical model through 3D model.

3.3 Solving Rubik's cube

There was also developed an abstract solution adapter which allows to easily define solution formulas depending on the face state in json files.

The implemented adapter reads these files and build necessary infrastructure for itself. Since the Rubik's cube solution formulas for beginners [8,9] have some operations that could not be easily described in json derived adapter class was created. This class contains logic for this kind of solution building first cross, first layer, second layer, second cross, third layer and also some internal steps required to perform solution steps. If there is a need to implement another solution, it can be easily done and plugged into the pipeline. Solution adapter manipulates with cube's logical model to perform formulas and analyzing cubes state to compare to json defined states in order to perform next step formulas. As can be seen from the fig. 9 the solution is displayed as a text so the user can solve his cube by reading the formulas. The program also displays the rotations of the cube faces so the user can watch the whole solving process until the end.

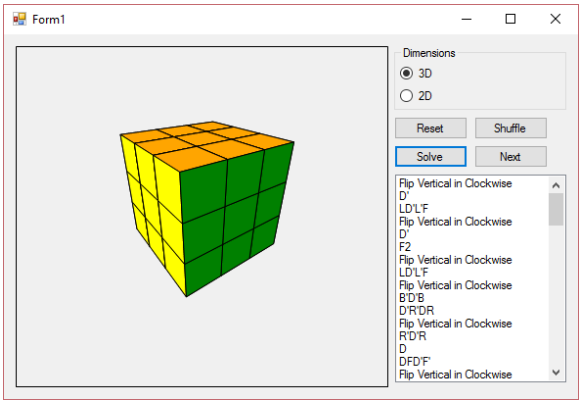

Fig. 9 Example of rendered solved logical model through 3D model.

4. RESULTS

Initial idea of finding cube face for extraction was kept due to insufficient results received from PTAM like approach. On the contrary, PTAM is pretty much suitable for tracking one of the cube faces on the screen. Despite of that, the full pipeline was built thus the cube can be solved with the data from video stream but for now it may take some time to get the whole picture of the cube depending on video conditions.

Even though the Kalman filter is researched on how to apply it to the current problem it has not been yet applied to it. The same thing is

5. FUTURE WORK

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

5.1 Kalman filter for noise removal

It is still needed to apply Kalman filter to increase the performance of the system. Kalman will replace feature matching algorithms by performing predictions.

5.2 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 Rubik's cube on the screen [5].

6. CONCLUSION

To sum up, I have implemented great application that simplifies life for people who does not know how to solve the cube they have. Working with this project gave me opportunity to get more knowledge in field of computer vision and computer graphics 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. How a Kalman filter works, in pictures (http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/)
3. An Introduction to the Kalman Filter (http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf/)
4. Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation (https://www.cl.cam.ac.uk/~rmf25/papers/Understanding%20the%20Basis%20of%20the%20Kalman%20Filter.pdf)
5. Object Tracking: Kalman Filter with Ease (https://www.codeproject.com/Articles/865935/Object-Tracking-Kalman-Filter-with-Ease)
6. 3D in Plain C# (https://www.youtube.com/watch?v=SOOsJjgHgi8)
7. Tutorial series: learning how to write a 3D soft engine from scratch in C#, TypeScript or JavaScript (https://www.davrous.com/2013/06/13/tutorial-series-learning-how-to-write-a-3d-soft-engine-from-scratch-in-c-typescript-or-javascript/)
8. How to solve the Rubik's Cube (https://ruwix.com/the-rubiks-cube/how-to-solve-the-rubiks-cube-beginners-method/)
9. How to solve 3x3 Rubik's cube (http://speedcubing.com.ua/howto/3x3sch.php)