Relative motion estimation and structure from motion

Technical report

Roman Juranek
Brno University of Technology

This document describes a simple motion estimation and structure from motion algorithm from a single calibrated monocular camera, and lists algorithms required for its implementation.

Introduction

Structure from motion (SFM) is a process of estimating 3D structure from point correspondences in a *moving camera*. It is an essential part of 3D reconstruction and environment mapping (Simultaneous Localization and Mapping, SLAM) and visual odometry. The output of SFM are 3D locations of points estimated from immediate camera motion. Here I address the situation where SFM is computed from a single monocular camera input only. However, more cameras and other data sources can be used with advantage to make the results more precise and robust. SFM pipeline consists from these steps:

1. Point detection in image
2. Point correspondences between two images
3. Relative motion estimation from the correspondences
4. Triangulation of 3D point locations

SFM inputs

The inputs to the SFM are:

- **Image sequence** - grayscale or RGB, all images same size, taken with a pinhole camera (i.e. without radial distortion).
- **Intrinsic camera calibration matrix K** - 3x3 matrix i form **K** = [f,0,0; 0,f 0; cx,cy,1], where f is a focal length in pixels and [cx,cy] principal point.

Point detection

Detection of interest points in image is one of critical steps in SFM as it forms the information known about the image. When detected points are bad, results will be also bad. The points should cover the image regularly (which is sometimes hard to achieve in real conditions) and must be repeatable, i.e. the same points must be reliably detected in other images. The choices typically used in SFM are:

- Shi-Tomasi min. eigenvalue points (or sometimes called good features to track)
- Harris features
- Keypoints like SIFT, SURF, FAST, etc.

The result of point detection is *Nx2* matrix **p** = [x,y; x,y, ...] with locations of points in the input image, and possibly *NxF* matrix **f** with descriptor for each point (in case of SIFT and similar methods). This points (and descriptors) are extracted for every frame in the sequence. Points in time step *t* are referred to as $\mathbf{p}^t$ ($\mathbf{f}^t$).

Correspondences

Point correspondences are important for estimation of camera motion. From the knowledge, how the points move in image, we can estimate how the camera moved. There are two main methods how to get the correspondences:

1. **Tracking** - points detected in an image are tracked in the subsequent images. KLT tracker is often used. In this case the correspondences are simply formed by point tracks.
2. **Matching** - Descriptors $\mathbf{f}^t$ and $\mathbf{f}^{t-1}$ are matched and index pair matrix **I** (Nx2) is formed.

The output of this step are two Mx2 matrices $\mathbf{p}_1$ and $\mathbf{p}_2$ with corresponding points from $\mathbf{p}^{t-1}$ and $\mathbf{p}^t$.

Relative motion estimation

In this step, camera relative motion from t-1 to t is calculated from the point correspondences. The outputs are 3x3 rotation matrix **R** and translation vector **T**. The algorithm is following:

1. From $\mathbf{p}_1$ and $\mathbf{p}_2$ estimate essential matrix **E** - RANSAC with five point algorithm.
2. Decompose **E** to $\mathbf{R}_i$, $\mathbf{T}_i$ (I = 1..4, four possible solutions)
3. Select feasible solution - **R**, **T**

Essential matrix

The essential matrix **E** (3x3) encodes relative rotation and translation between two cameras.

$\mathbf{E} = [\mathbf{T}]_x \mathbf{R}$

Where $[\mathbf{T}]_x$ is a 3x3 skew symmetric matrix created from **T**, and **R** is the 3x3 rotation matrix.

For corresponding points (with normalized coordinates) *a* and *b* from two images must hold:

$a \mathbf{E} b^T = 0$.

This is called an *epipolar constraint*.

Robust estimation with RANSAC

RANSAC finds **E** which minimizes the epipolar constraint for all the point pairs. It works as follows:

1. Select 5 random point pairs
2. Use five-point algorithm to calculate **E**
3. Store this solution if it is better than already found solution (better epipolar constraint value)
4. Loop from 1 until maximum iterations are reached or confidence reaches certain level
5. Select inliers ($a \mathbf{E} b^T <$ threshold)
6. Calculate final solution form all inliers

The algorithm relies on random selection of 5 points, so the number of iterations that need to be calculated is rather high. It can reach a few thousands which can be very slow. It also depends on the number of outlier points. The outputs are **E** and indices of inlier points.

Decomposition of essential matrix and select R,T

In this step, **E** is decomposed to four possible solutions for $\mathbf{R}_i$ and $\mathbf{T}_i$ ($i = \{1..4\}$). This is done by SVD based algorithm from [1]. This forms four camera pairs (one in $\mathbf{T} = [0,0,0]$, $\mathbf{R} = \mathbf{I}_{3x3}$ and the other in $\mathbf{T}_i$, $\mathbf{R}_i$). For each of this four solutions, a point is triangulated and the solution where the point lies in front of both cameras is selected - this gives the final **R**, **T**.

Triangulation

Inlier point pairs from p1 and p2 are triangulated using camera pair:

1. $\mathbf{R}_1 = \mathbf{I}_{3x3}$, $T_1 = [0,0,0]$
2. $\mathbf{R}_2 = \mathbf{R}$, $T_2 = T$

The triangulation itself can be done by SVD based algorithm from [1]. The output is Nx3 matrix **X** with homogeneous 3D coordinates of points - the final structure.

Required algorithms

- Basic linear algebra - Matrix operations, linear system solving, SVD, QR decomposition, etc.
- Feature detectors and descriptors - Harris, SURF
- KLT point tracker
- RANSAC (MSAC, or other similar robust estimator)
- Five point algorithm
- Triangulation from two point correspondences

There algorithms are already implemented in open source libraries like OpenCV, Eigen, LAPACK and others.

Application hints

The described algorithm produces the structure up to an unknown scale factor. In practice, this is not a limitation since the scale can be supplied from an external sensor (like IMU). With known velocity, the scale can be recovered by a simple solution and the depth information can be always obtained in correct units.

SFM produces points for arbitrarily oriented cameras, when the camera orientation in world is known (e.g, extrinsics w.r.t. vehicle), the point cloud can be aligned to world coordinates and used for odometry or detection of obstacles.

While the algorithm is general enough and does not impose almost any restrictions on camera location and orientation, in general it is better to use sideways facing cameras. On front/rear cameras the solution is unstable.

The only limitations which remain are: 1/ the camera must move between frames, and 2/ the movement must not be very large (so point tracking still works).

Possible improvements

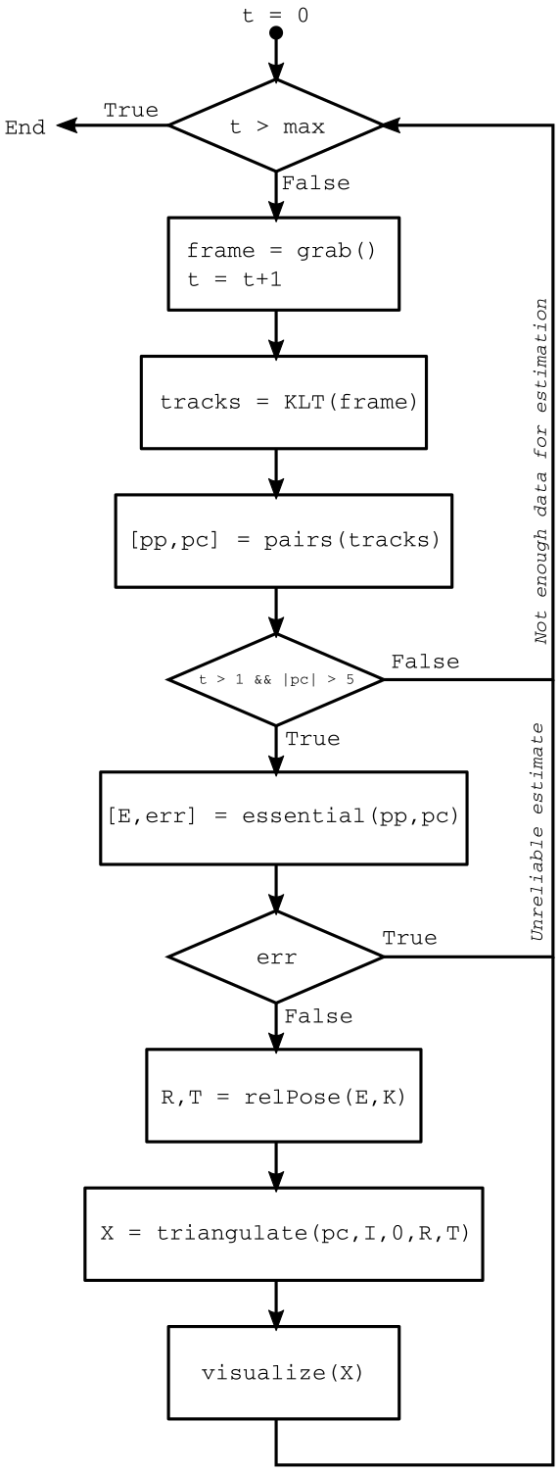
- **Motion estimation for vehicles** - The method above is general and can be used to estimate structure form arbitrary motion (6 degrees of freedom - 3 for camera position, 3 for rotation). This is unnecessary in the case of vehicles where the motion is constrained by Ackerman steering principle and (approximately) planar movement. Vehicles have 2 DOF - steering radius and displacement. There are algorithms that take advantage of

this vehicle model and can estimate the motion more precisely (and faster). The consequence of this is better estimation of 3D structure around the vehicle.

- **Incremental structure building and 2D-to-3D alignment** - This is a complete scene reconstruction pipeline. There are three steps. 1/ Initial structure estimation - basically a SFM as described in this document. 2/ Motion estimation w.r.t. the initial structure and adding more correspondences, and global bundle adjustment after each frame. This is done on few initial frames and the result is well aligned initial structure. 3/ Motion estimation w.r.t. 3D structure and windowed bundle adjustment.

References

1. R. Hartley, A. Zisserman, "Multiple View Geometry in Computer Vision", Cambridge University Press, 2003.



Control flow of KLT version of SFM as implemented in sfmKLT.m. KLT() is implemented by PointTracks class which take care of correct tracking of features and for each feature provide its movement history. Pairs are selected by constructEndpoints(). Essential matrix and relative pose are solved by internal Matlab functions which runs RANSAC (MSAC actually) with 5-point algorithm and selects most probably R,T.