Visual Odometry Project

Nicolas Filliol, Marek Landert, Luca Schwarzenbach, Joel Asper

January 6, 2024

Abstract

This report provides an overview of our implementation of a monocular Visual Odometry pipeline in Python. For feature extraction, matching or tracking, we provide Harris Corners, SIFT and KLT. The matched features of two consecutive frames and the corresponding triangulated landmarks are used for pose estimation. Our pipeline is able to produce a locally correct trajectory for the three different datasets Kitti, Malaga and Parking. However, unless parameters are finetuned for the specific dataset, it globally suffers from significant drift both in scale and space.

Contents

1 Introduction **2**

2 Visual Odometry Pipeline **2**

2.1 Building Blocks 2

2.1.1 Feature Extraction and Matching 2

2.1.2 Pose Estimation using 2D Correspondences 4

2.1.3 Pose Estimation with P3P 4

2.1.4 Triangulation 4

2.2 Implementation 5

2.2.1 Initialization 5

2.2.2 Continuous Operation 5

2.2.3 Trajectory Plot 6

3 Results **6**

3.1 Current Limitation 9

3.1.1 Scale Drift 9

3.1.2 Trajectory failure 9

3.1.3 Performance 9

4 Author Contributions **9**

1 Introduction

In recent years, robotics has seen an exponential increase in popularity and autonomous robots are being used in a growing number of applications. This increases the demand for exact motion and pose estimation of robots and other unmanned vehicles. Odometry describes the use of sensor data for estimating changes in the position and orientation of a robot. One of the simplest ways to do so is to detect the wheel spin or number of steps of wheeled or legged robots, respectively. However, these methods suffer from inaccuracies as they are very prone to undesired effects like wheel slippage or drift, and the errors accumulate over time. Furthermore, these methods do not work for aerial vehicles. A better way lies in the use of camera systems for estimating motion information, so-called *visual odometry.* The onboard vision system tracks special points like corners or edges in the surroundings. This demands a texture-rich scene to provide enough keypoints. The detected keypoints of two consecutive frames are then used to estimate the change in rotation and translation of the robot. Visual odometry is very accurate compared to wheel odometry.[1]

In this work, we develop a monocular visual odometry pipeline in Python, intending to produce locally consistent trajectories (up to scale) by tracking keypoints over consecutive frames, triangulating landmarks, and estimating poses based on 3D-2D point correspondences. We then rigorously test our pipeline across three sequences to assess its accuracy qualitatively and identify any prevailing constraints.

2 Visual Odometry Pipeline

In this section, an overview of the implemented visual odometry (VO) pipeline is provided. First, the building blocks used for the pipeline are described, and later the implementation is explained.

2.1 Building Blocks

Although the main building blocks remain the same for most VO pipelines, there exist many different methods and algorithms for the submodules themselves. Our VO pipeline supports different tracking methods, which are listed and described in this subsection.

2.1.1 Feature Extraction and Matching

Feature extraction in image processing involves two primary stages: detection and description. The detection phase focuses on identifying unique or notable points within an image, commonly referred to as keypoints. These keypoints are usually corners or blobs, chosen for their distinctiveness and repeatability, making them easily recognizable in different images. In the description phase, a descriptor is generated based on the information in the vicinity of each detected keypoint. This descriptor facilitates matching the keypoint with corresponding points in other frames.

The methods of detection and description vary widely. In our visual odometry (VO) pipeline, we have incorporated support for both Harris Corners and Scale-Invariant Feature Transform (SIFT), which are explained in subsequent sections. Additionally, to enhance the matching process, we employ an alternative to brute-force matching by using the Kanade-Lucas-Tomasi (KLT) Tracking method. This approach focuses on tracking features across consecutive frames, which should ensure more efficient and accurate feature correspondences.

[1]Davide Scaramuzza "Lecture Note 01 Introduction" pp. 60–62

Harris Corner Detector If one takes a small patch around a pixel at a corner of the image, the Sum of Squared Differences (SSD) is high when moving the patch in all directions. In contrast, at a flat region, the SSD is small in all directions, and at an edge, the score is high only when moving the patch perpendicular to the edge. The disadvantage of this corner detection method is that it is very computationally intensive.
A more efficient alternative is the Harris Corner Detection method (Harris and Stephens (1988)). Rather than shifting the area around the pixel, this method analyzes the area directly using differential calculus. The Harris corner detector method uses Sobel filters to compute the gradient of the image patch. This can be derived as follows.[2]

$$SSD(\Delta x, \Delta y) = \sum_{x,y \in P} (I(x,y) - I(x+\Delta x, y+\Delta y))^2$$

Using the first-order Taylor expansion leads to the approximation:

$$\begin{aligned} SSD(\Delta x, \Delta y) &\approx \sum_{x,y \in P} (I_x(x,y)\Delta x + I_y(x,y)\Delta y))^2 \\ &\approx \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \end{aligned}$$

where M = $\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$.

The Harris score for a pixel is given as $R = det(M) - \kappa \cdot trace^2(M)$. The cornerness function R is high only if both eigenvalues of M are large, which indicates a corner region. Then the pixels with the highest Harris scores are taken as keypoints. By applying non-maximum suppression, only the pixel with the highest Harris score in a certain region of the image is taken.

For each feature, a descriptor is extracted based on the intensities of all the pixels within a defined radius around the keypoint pixel. These descriptors are then used to match keypoints of two images. For every keypoint in one image, the two best-matching keypoints of the other image are searched by taking the one with the minimum SSD scores. Then the ratio test is applied, meaning that the best matching keypoint is only taken as a valid match if this SSD score is less than 85% of the second-best score and if the feature is not already used for another match. The computed matches are needed for pose estimation and triangulation.

SIFT The Scale-Invariant Feature Transform (SIFT) is another computer vision technique for detecting and matching features in images. The key advantage of SIFT is its ability to identify keypoints that are resistant to changes in scale, rotation, and lighting conditions. The process starts with creating a scale-space representation known as the Difference of Gaussian (DoG) pyramid. This involves progressively blurring the image with Gaussian filters, each at a different scale, and then subtracting adjacent blurred images to form the DoG pyramid. Keypoints are then located as local extrema in this pyramid by comparing each pixel with its 26 neighbors across both space and scale.
For each detected keypoint, SIFT calculates a distinctive descriptor based on the gradients and orientations around it. This descriptor, known as the histogram of oriented gradients (HOG), encapsulates the keypoint's unique characteristics, enabling effective matching between different frames.[3]
Despite its computational intensity, SIFT stands out for its robustness in feature detection and matching, offering superior global correctness compared to other algorithms. Unlike the Harris detector, which focuses on corners, SIFT functions as a blob detector. This means while it

[2]Davide Scaramuzza "Lecture Note 05 Point Feature Detection and Matching - Part 1" pp. 37–58
[3]Davide Scaramuzza "Lecture Note 06 Point Feature Detection and Matching - Part 2" pp. 40–68

may have slightly reduced localization accuracy - a potential drawback for visual odometry - its enhanced ability for quality matching might mitigate this concern.

KLT The Kanade-Lucas-Tomasi (KLT) feature tracker (Lucas and Kanade, 1981; Tomasi and Kanade, 1991) is another algorithm for tracking feature points through a sequence of images. It operates on the assumption that the motion of feature points is small and smooth over time. The core idea of KLT tracking is to find the displacement of feature points that minimize the difference in their appearance between consecutive frames. This is achieved by solving an optimization problem, where the sum of squared differences in pixel intensities within a local window around the feature is minimized.

2.1.2 Pose Estimation using 2D Correspondences

Given N 2D correspondences $(\mathbf{p_1^i}, \mathbf{p_2^i})_{i=1}^{N}$ in homogeneous coordinates found using feature matching, the relative transform between the two camera frames can be determined using the fundamental matrix $\mathbf{F}$. It relies on the epipolar constraint $\mathbf{p_2^i F p_1^i} = 0$ which must hold for all correspondences. By writing the elements of $\mathbf{F}$ into a column vector and stacking the equations for all point correspondences, this equation can be rewritten in the form $\mathbf{Q\bar{F}} = 0$, which is solvable using least-squares/singular value decomposition (SVD). Since every point contributes one equation, and Q must have at least rank 8 for a unique (up to scale) solution to exist, a minimum of 8 points are needed and give this algorithm the name *8-point algorithm* (Longuet-Higgins, 1981). The essential matrix $\mathbf{E} = \mathbf{K_2}^T\mathbf{F K_1} = [\mathbf{T}]_\times \mathbf{R}$ can be decomposed into a unit translation vector $\mathbf{T}$ and rotation matrix $\mathbf{R}$, which define the relative transform from the first camera frame into the second camera frame. This results in four up to scale solutions, which need to be disambiguated using the cheirality check after triangulating the 3D points (see 2.1.4). This suggests that estimating the pose using the 8-point algorithm estimates 3D landmarks as a byproduct.

2.1.3 Pose Estimation with P3P

The Perspective-Three-Point (P3P) algorithm is a method used for pose estimation from three plus one 3D-to-2D point correspondences. Three points in the camera frame and their corresponding points in the world frame are used to solve the P3P problem which has up to four solutions. The fourth point correspondence is then used to disambiguate the solutions and thus determine the translation vector $\mathbf{T_{CW}}$ and the rotation matrix $\mathbf{R_{CW}}$, which define the relative transform from the world frame to the camera frame.

2.1.4 Triangulation

The assumption for triangulation is that we are given a keypoint $\mathbf{p_1}$ in the first frame, a matching keypoint $\mathbf{p_2}$ in the second frame (both in homogeneous coordinates), as well as the projection matrices $M_1 = \mathbf{K_1}[\mathbf{R_1}|\mathbf{T_1}]$ and $M_2 = \mathbf{K_2}[\mathbf{R_2}|\mathbf{T_2}]$. Then, for the triangulated point $\mathbf{P}$ (homogeneous coordinates) it must hold:

$$\lambda_1 \mathbf{p_1} = \mathbf{M_1 P} \Rightarrow 0 = [\mathbf{p_1}_\times]\mathbf{M_1 P}$$
$$\lambda_2 \mathbf{p_2} = \mathbf{M_2 P} \Rightarrow 0 = [\mathbf{p_2}_\times]\mathbf{M_2 P}$$

Stacking the two constraints, which each provide three equations, results in an overdetermined homogenous system $\mathbf{AP} = 0$ which is solved using least-squares.

2.2 Implementation

The visual odometry pipeline can be divided into two parts: the initialization phase and the continuous operation phase. The initialization uses two frames to provide an initial set of keypoints, landmarks, and the starting pose of the camera. In the continuous operation, for every new frame keypoints are detected, and matched with keypoints of the old frame, and the resulting 3D-2D correspondences are then used to estimate the new camera pose. New landmarks are triangulated periodically using 2D-2D correspondences and the estimated pose. The specifics of both the initialization and continuous operation phases are described in the subsequent sections.

2.2.1 Initialization

In a visual odometry pipeline, the initialization, also called bootstrapping phase, is critical for initializing the system. Starting with loading a sequence of images, we establish a base pose with no rotation or translation.
We take the first and the third frames to perform bootstrapping, to increase the baseline between the two images and produce higher quality landmarks. First, we extract and match keypoints from these two images. Here, we tried the three different methods mentioned in 2.1.1 (also for the continuous phase). For every method, the parameters were precisely tuned to achieve the best performance. In the end, we decided to run our pipeline with the SIFT feature detector as we experienced the least scale drift there and thus got a (global) trajectory closest to the ground truth. With the matched keypoint pairs from the two initialization frames we can perform triangulation (as described in 2.1.4). The inverse transform of the computed rotation $\mathbf{R}$ and translation $\mathbf{T}$ define the initial camera pose after bootstrapping. The triangulated 3D landmarks from the keypoints and transformation matrix define the initial set of landmarks, based on which the continuous operation is performed. The plotting of trajectories and landmarks visualizes this initial step, for establishing a reference for subsequent motion tracking. This foundational phase is essential for the accuracy and effectiveness of the entire VO process, and thus we make it more robust by using RANSAC in combination with the fundamental matrix estimation and only keep the inlier landmarks for continuous operation.

2.2.2 Continuous Operation

In the subsequent phase of our visual odometry pipeline, each new frame from the sequence undergoes processing. To estimate the current pose and landmarks, the keypoints of the current and last frame, as well as the pose and landmarks of the last frame are needed. Each Python `Frame` object stores the current frame of the video sequence and a `Feature` object containing the keypoints and landmarks of this frame. A `State` object is used to store the current and previous frames and poses of the camera. The processing can be divided into three parts.
As described in 2.2.1 we use the SIFT feature detector (2.1.1) to detect features in the current frame and use the HOG descriptors to find keypoint matches between the current and the previous frame, using a brute-force matcher and the ratio test. The previous landmarks corresponding to the matched keypoints are taken over and stored as landmarks (in world coordinates) of the current frame. The newly detected keypoints with no match are stored with no corresponding landmarks.
The matched keypoints and corresponding landmarks of the current frame, as well as the camera intrinsic matrix K, are used to estimate the new camera pose (rotation matrix and translation vector with respect to world frame) of the current frame with the P3P algorithm described in 2.1.3. The pose estimator uses RANSAC to filter out outliers caused by tracking errors and uses only the inliers of the matched keypoints to compute the pose. Finally, non-linear refinement

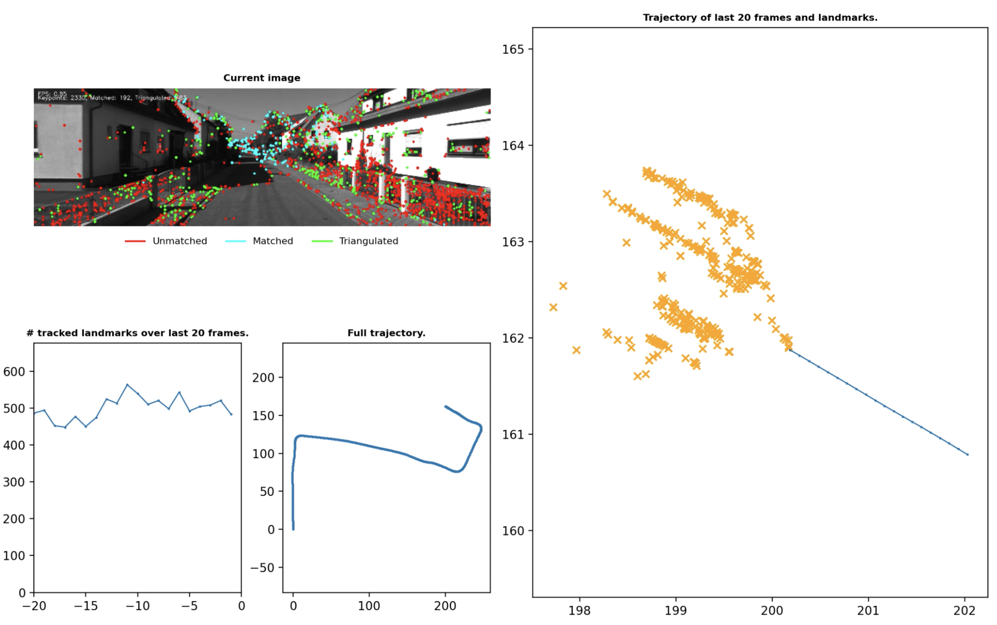

Figure 1: Python figure showing the current frame and trajectory plots of the Kitti dataset.

is applied to the pose by minimizing the sum of the squared reprojection errors using SciPy's *least_squares* function.
Keypoints that are matched with a keypoint of the previous frame but do not have a corresponding landmark yet, are stored with the history of the track, i.e. the start keypoint and camera pose, and flagged as candidates if its bearing angle between the backprojected rays of the two frames is above a certain threshold. With these candidate keypoint pairs, new landmarks are triangulated (2.1.4) and stored in the Feature object of the current and previous frame. This has to be done continuously as at some time the landmarks of the bootstrapping would not be visible anymore as the camera moves.
This process iteratively builds a detailed trajectory. By using SIFT, we are able to detect and match a lot of keypoints and thus keep a good amount of triangulated landmarks (300-500) in most cases.

2.2.3 Trajectory Plot

To show the results of the VO pipeline, different information is plotted as depicted in Figure 1. The current frame of the input sequence is shown. In green the triangulated, in blue the matched, and in red the detected but not matched keypoints are indicated. The number of points in the respective category is written on the top left of the current video frame, as well as the frame processing rate (fps) at which the VO pipeline runs. The plot on the right shows the pose of the last 20 frames as well as the triangulated landmarks of the current frame. The two smaller plots on the left show the amount of triangulated landmarks in the last 20 frames and the entire trajectory tracked by the VO pipeline.

3 Results

The implemented pipeline was tested on the three provided datasets *KITTI 05*, *Malaga 07*, and *parking garage*. As described in 2.2.1, the SIFT feature detector is chosen as it performed best over all three datasets. The parameters used for SIFT and RANSAC were chosen to get the best performance over all three datasets, and not individually for every dataset, to show the

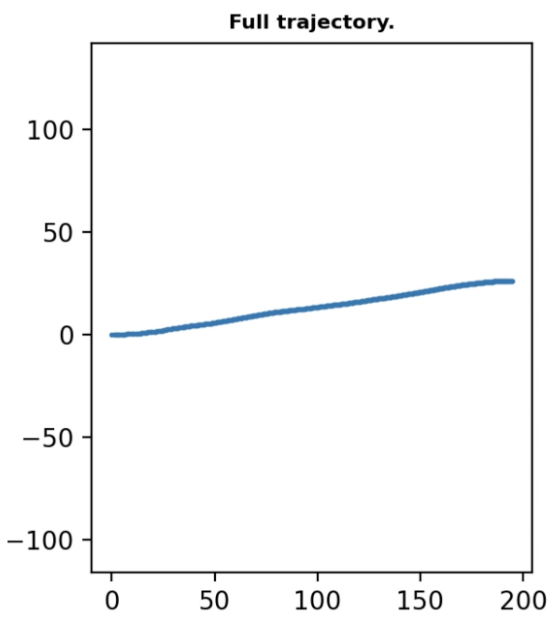

Figure 2: Estimated trajectory of the Parking garage dataset.

robustness of our VO pipeline. We were surprised by the sensitivity of the RANSAC parameters, but finally figured that very small inlier threshold for initialization and a slightly more relaxed threshold for P3P pose estimation works best, such that a good amount of landmarks is kept as inliers.

Our VO pipeline runs at about 0.7 to 0.8 frames per second (fps) on a 2020 MacBook Pro (Intel i5, 4 Core) and at around 1.5 to 2 fps on a newer M2 MacBook Air. The resulting trajectories are shown in Figures 2, 3, and 4 respectively. It's important to note that a significant portion of the processing time is attributed to visualization; without this graphical output, the pipeline's speed increases to about 10 -15 fps on the M2 Mac.

We further go on with a qualitative assessment of the resulting trajectories for each dataset. The ground truth of the parking garage dataset should be a straight horizontal line. It can be seen that the full trajectory dataset looks quite well and we nearly achieved global consistency apart from a directional drift. On a local scale, the trajectory remains consistent throughout the dataset.
In the cases of the KITTI and Malaga datasets, however, we observed significant scale drift, leading to trajectories that deviate from the ideal. Despite this, the KITTI dataset shows local consistency in its trajectory, as evidenced in the screencast recording titled ”trajectory of last 20 frames and landmarks.”
Similarly, the Malaga dataset maintains local consistency for most of the sequence. Nevertheless, it experiences sudden scale changes and occasional local inconsistencies, particularly in areas where feature matching proves challenging, such as in high dynamic range scenarios caused by sunlight in the background, or scenes with minimal distinctive textures like those involving trees. These challenging conditions often result in strong fluctuations in the number of triangulated landmarks from frame to frame, oscillating from nearly none to several hundred, and then back again. Such rapid changes pose significant challenges for the P3P algorithm to accurately estimate the camera pose.

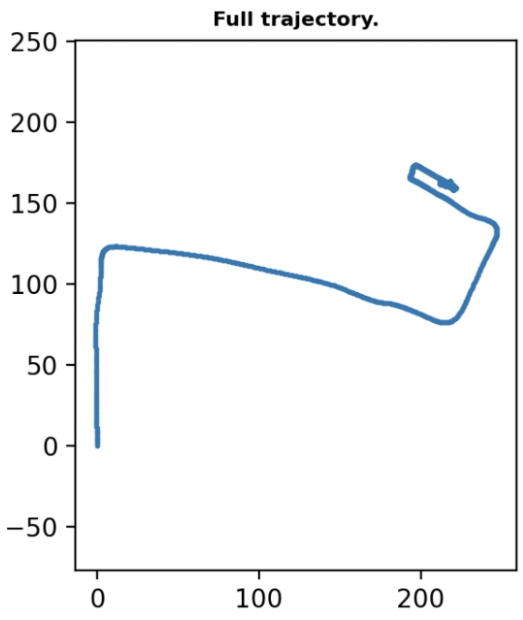

Figure 3: Estimated trajectory of the KITTI 05 dataset.

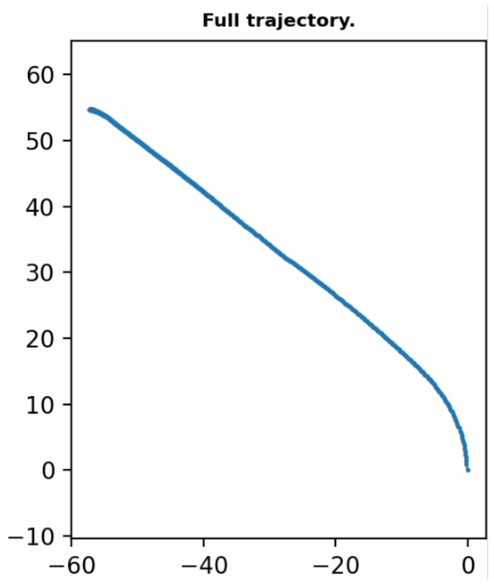

Figure 4: Estimated trajectory of the Malaga 07 dataset.

3.1 Current Limitation

As described above, the implemented VO pipeline works reasonably well when considering local consistency. In this section, the above-mentioned limitations of our VO pipeline are depicted in more detail.

3.1.1 Scale Drift

Every now and then, the distance between two estimated camera poses is decreased in the KITTI and Malaga dataset and thus, the scale of the entire plot decreases. This scale drift could be tackled by bundle adjustment (BA). This can be considered as a possible future work. Furthermore, there are sometimes abrupt changes in scale, which are considered together with trajectory failures.

3.1.2 Trajectory failure

The pose estimation or trajectory failure in the Malaga dataset most likely occurs due to the small amount or wrongly matched keypoints. Because of this (i) a bad pose from P3P is estimated and (ii) not enough or bad landmarks are triangulated, which then leads to an inaccurate camera pose estimation again in the next iteration and leads to kind of a vicious circle which is hard to recover from as long as there keep to be many wrong matches. One could try to tune the RANSAC parameters of the pose estimator or the parameters of the keypoint detection algorithm to overcome the trajectory failure of the VO pipeline.

3.1.3 Performance

Our pipeline runs at about 0.8 or 2 fps, depending on the processor. Most of the time spent during one iteration is needed for plotting. When removing the plots, the VO pipeline runs at up to 4 fps on older MacBooks and up to 15 fps on a newer M2 MacBook Air. To improve the speed, one could use multithreading and run plotting in separate threads at a slower update rate than the actual VO pipeline.

4 Author Contributions

- Nicolas Filliol: Contribution to template classes, bootstrapping and continuous operation loop. Implementing SIFT detector, triangulation and 8-point algorithm, RANSAC. Recordings. Parameter Tuning. Writing Report.
- Marek Landert: Contribution to template classes, continuous operation loop, visualization. Implementing Harris corner detector. Recordings. Parameter Tuning. Writing Report.
- Luca Schwarzenbach: Implementing pose estimation (P3P), detection and tracking of candidate keypoints. Recordings. Parameter Tuning. Writing Report.
- Joel Asper: Contribution to template classes, data loading, visualization. Implementing KLT. Recordings. Parameter Tuning. Writing Report.

References

Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151.

Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133–135.

Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, page 674–679, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Tomasi, C. and Kanade, T. (1991). Detection and tracking of point features.