Visual Odometry

Dhruv Patel July 16, 2020

Contents

1 Introduction : Problem Formulation

What is odometry?
Normally, a wheel odometer (or simply *odometer*) measures the number of rotations that the wheel is undergoing, and multiplies that by the circumference to get an estimate of the distance travelled by the car. **Odometry** is used as a more general term in Robotics, and often refers to estimating not only the distance traveled, but the entire trajectory of a moving robot. So, for every time instance t, there is a vector $[x^t\ y^t\ z^t\ \alpha^t\ \beta^t\ \gamma^t]^T$ which describes the complete **pose** of the robot at that instance.
Note : $\alpha^t, \beta^t, \gamma^t$ here are the **Euler Angles**, while x^t, y^t, z^t are **Cartesian Coordinates** of the robot.

What is Visual Odometry?
In Visual Odometry, we have a camera (or an array of cameras) rigidly attached to a moving object (such as a car or a robot), and our job is to construct a **6-DOF** trajectory using the video stream coming from this camera(s). When only one camera is used, it's called **Monocular Visual Odometry** and when two (or more) cameras, it's referred to as **Stereo Visual Odometry**.

Input:
We have a stream of images (i.e. video - grayscale/color) coming from a camera. Let the frames, captured at time t and $t+1$ be referred to as I^t, I^{t+1}. We have prior knowledge of all the intrinsic parameters, obtained via calibration, which can be done in OpenCV.

Output:
For every pair of images, we need to find the Rotation Matrix R and the Translation Vector t, which describes the motion of the vehicle between the two frames. The vector t can only be computed up to a scale factor in our monocular scheme.

2 Algorithm : Overview

1. Capture Images : I^t, I^{t+1}
2. Feature Detection
3. Feature Tracking
4. Estimating the Essential Matrix
5. Estimating R, t from the essential matrix
6. Taking scale information from some external source and concatenate the translation

vectors, and rotation matrices.

2.1 Feature Detection

There are many different approaches for "Feature Detection" and it yearns more research into this as we head deep down into VO. For now, I am using **FAST** corner detector. Let's see how this detector works, though it will be useful to look at the original paper and source code to understand how it works.

Let's suppose there is a point **P** which we want to test if it is a corner or not. A circle is drawn of 16px circumference around this point as shown in figure below. For every pixel which lies on the circumference of this circle, we see if there exists a continuous set of pixels whose intensity exceed the intensity of the original pixel by a certain factor **I** and for another set of continuous pixels if the intensity is less by at least the same factor **I**. If **yes**, then we mark this point as a corner. This would generate many points and because we have to track all these features, it is recommended that we filter out some points to reduce computer resources. A heuristic for rejecting the vast majority of non-corners is used, in which the pixel at $1, 9, 5, 13$ are examined first, and at least three of them must have a higher intensity by amount **I**, or must have an intensity lower by the same amount **I** for the point to be corner. This particular approach is selected due to its computational efficiency as compared to other popular interest point detectors such as **SIFT** detector.

```
1 # FAST corner detector
```

2.2 Feature Tracking

The FAST corners detected in the previous step are fed to the next step, which uses a KLT Tracker. The KLT Tracker basically looks around every corner to be tracked, and uses this local information to find the corresponding corner in the next image. The corners detected in I^t are tracked in I^{t+1}. Let the set of features detected in I^t be $\mathcal{F}^t$, and the set of corresponding features in I^{t+1} be $\mathcal{F}^{t+1}$.

2.3 Feature Re-detection

While doing KLT Tracking, there will be some points which will eventually move out of the field of the view of the camera and we would lose tracking on them. Hence, a trigger

for re-detection is necessary whenever the total number of features go below a certain threshold.

2.4 Essential Matrix Estimation

Once we have point-correspondences, we have several techniques for the computation of an essential matrix. The essential matrix is defined as : $y_1^T * E * y_2 = 0$. Here, y_1, y_2 are homogenous normalized image co-ordinates. While a simple algorithm requiring eight point correspondences exists, a more recent approach that is shown to give better results is the five point algorithm. It solves a number of non-linear equations, and requires the minimum number of points possible, since the Essential Matrix has only five degrees of freedom.

2.5 RANSAC

If all of our point correspondences were perfect, then we would have need only five feature correspondences between two successive frames to estimate motion accurately. However, the feature tracking algorithms are not perfect, and therefore we have several erroneous correspondence. A standard technique of handling outliers when doing model estimation is RANSAC. It is an iterative algorithm. At every iteration, it randomly samples five points from the set of correspondences, estimates the Essential Matrix, and then checks if the other points are inliers when using this Essential Matrix. The algorithm terminates after a fixed number of iterations, and the Essential Matrix with which the maximum number of points agree, is used.

```
1 # Implementing RANSAC using OpenCV
```

2.6 Computing R, t from the Essential Matrix

Another definition of the Essential Matrix is as follows : $E = R[t]_x$. Here, R is the rotation matrix, while $[t]_x$ is the matrix representation of a cross product with t. Taking the SVD of the essential matrix, and then exploiting the constraints on the rotation matrix, we get the following :

$$E = U\Sigma V^T \tag{1}$$

$$[t]_x = VW\Sigma V^T \tag{2}$$

$$R = UW^{-1}V^T \tag{3}$$

```
# Implementing extraction of R, t from Essential Matrix using OpenCV
```

3 Constructing Trajectory

Let the pose of the camera be denoted by R_{pos}, t_{pos}. We can then track the trajectory using the following equation :

$$R_{pos} = RR_{pos} \tag{4}$$

$$t_{pos} = t_{pos} + tR_{pos} \tag{5}$$

4 The Pin-Hole Camera Model

The *pin-hole camera* model (or sometimes *projective camera* model) is a widely used camera model in computer vision. It is simple and accurate enough for most applications. The name comes from the type of camera, that collects light through a small hole to the inside of a dark box or room. In the pin-hole camera model, light passes through a single point, the *camera center*, **C**, before it is projected onto an *image plane*. Figure below shows such instance where the image plane is drawn in front of the camera center. The image plane in an actual camera would be upside down behind the camera center, but the model is the same.

The projection properties of a pin-hole camera can be derived from this illustration and the assumption that the image axis is aligned with the x and y axis of a 3D coordinate system. The *optical axis* of the camera then coincides with the z axis and the projection follows from similar triangles. By adding rotation and translation to put a 3D point in this coordinate system before projecting, the complete projection transform follows.

With a pin-hole camera, a 3D point **X** is projected to an image point **x** (both expressed in homogeneous coordinates) as

$$\lambda \, \mathbf{x} = P \, \mathbf{X} \tag{6}$$

Here, the 3 x 4 matrix *P* is called the *camera matrix (or projection matrix)*. Note that the 3D point **X** has four elements in homogeneous coordinates, $X = \Big[X, Y, Z, W\Big]$. The scalar λ is the *inverse depth* of the 3D point and is needed if we want all coordinates to be homogeneous with the last value normalized to one.

4.1 The Camera Matrix

The camera matrix can be decomposed as

$$P = K \, [R \,|\, t], \tag{7}$$

where,
R = rotation matrix describing the orientation of the camera,
t = 3D translation vector describing the position of the camera center
K = intrinsic that describes the projection properties of the camera.

The calibration matrix depends only on the camera properties and is in a general form written as,

$$K = \begin{bmatrix} \alpha f & s & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

The *focal length*, f, is the distance between the image plane and the camera center. The skew, s, is only used if the pixel array in the sensor is skewed and can in most cases safely be set to zero. This gives

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

where, we used the alternative notation f_x and f_y with $f_x = \alpha f_y$.
The *aspect ratio*, α is used for non-square pixel elements. It is often safe to assume $\alpha = 1$. With this assumption, the matrix becomes

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

Besides the focal length, the only remaining parameters are the coordinates of the *optical center* (sometimes called the *principal point*), the image point $c = [c_x,\, c_y]$ where the optical axis intersects the image plane. Since this is usually in the center of the image and image coordinates are measured from the top-left corner, these values are often well approximated with half the width and height of the image. It is worth noting that in this last case the only unknown variable is the focal length f.

4.2 Factoring the Camera Matrix

If we are given a camera matrix P of the form in equation (7), we need to be able to recover the internal parameters K and the camera position and post t and R. Partitioning the matrix is called *factorization*. In this case, we will use a type of matrix factorization called *RQ-factorization*.

Note : RQ-factorization is not unique - meaning - there is a sign ambiguity in the factorization. Since we need the rotation matrix R to have positive determinant (otherwise the coordinate axis can get flipped), we can add a transform T to change the sign when needed.

4.3 Computing the Camera Center

Given a camera projection matrix, P, it is useful to be able to compute the camera's position in space. The camera center, C, is a 3D point with the property $P\,C = 0$. For a camera with $P = K\,[R \mid t]$, this gives

$$K[R \mid \mathbf{t}]\,\mathbf{C} = K\,R\,\mathbf{C} + K\mathbf{t} = 0 \tag{11}$$

and the camera center can be computed as

$$\mathbf{C} = -R^T \mathbf{t} \tag{12}$$

Note :: The camera center is independent of the intrinsic calibration K, as it should be.

4.4 Camera Calibration

Calibrating a camera means determining the internal camera parameters, in our case the matrix K. It is possible to extend this camera model to include radial distortion and other artifacts if your application needs precise measurements. For most applications, however, the simple model in equation (9) is good enough. The standard way to calibrate cameras is to take lots of pictures of a flat checkerboard pattern.