Visual Odometry

Miguel Angel Maestre Trueba and Sudarshan Raghunathan – ENPM673

The objective of this part of the project is to localize the camera of the car by using the information given by the images.

- The first step in the algorithm is to check for the camera intrinsic parameters. This is done by using the given function `ReadCameraModel`, provided by the Oxford Dataset. This generates the focal length, the principal point of the camera and the undistortion lookup table. This last one is used in one of the following steps.

- The images come in Bayer format. To convert the image into a standard RGB image, the function `demosaic with GBRG alignment` is used. This is done for both the current frame and the next frame.

- Once the image is in RGB format, it has to be undistorted. We use the LUT parameters given by `ReadCameraModel`, and plug them into the `undistortImage` function, also provided by the Oxford Dataset.

- As always, the image is denoised by applying a Gaussian filter to it. This helps to detect better features in the next step.

- Once the image has been prepared and filtered, we have to detect features in the current frame and in the next frame and then match them. There are multiple feature extractors. For our experiments, we tried Harris, SIFT, SURF and FAST. The results with SURF and FAST are the best, so we decided to go use these as the main ones for the rest of the experiments. The figure shown below represents two consecutive frames of the video with SURF matched features.

- The MATLAB functions for the feature matching give a list of the matched points in both images. Before getting to calculate the Fundamental Matrix, we need to remove bad matches with RANSAC. The difference between using RANSAC and not using it is very big. To use RANSAC, we used the *ransac.m* function from Peter Kovesi's toolbox since it's a complex algorithm. The list of inliers are used as input to calculate the Fundamental Matrix.

- Once the inliers have been stored, it's time to calculate the Fundamental Matrix. There are several ways to determine this matrix. MATLAB has a function called `estimateFundamentalMatrix`. This function can compute the matrix by using RANSAC to discard outliers (by doing this, the previous step is not needed). But for our own results, we used the Eight-Points Algorithm method. The first step for this is normalizing the points and get the following expression:

$$\begin{bmatrix} x_1x'_1 & x_1y'_1 & x_1 & y_1x'_1 & y_1y'_1 & y_1 & x'_1 & y'_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_mx'_m & x_my'_m & x_m & y_mx'_m & y_my'_m & y_m & x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

x are the RANSAC inliers in the current frame and x' are the RANSAC inliers in the next frame. The f values are the fundamental matrix elements. Once the matrix on the left is calculated, we have to apply Singular Value Decomposition to it. The Fundamental matrix is extracted from the smallest singular value from this decomposition. Now we run a new Singular Value Decomposition and enforce the matrix to have rank 2. This is done with the following equation:

$$F = U * diag(1, 1, 0) * V'$$

- Once the Fundamental Matrix is determined and the K matrix is known, the Essential Matrix is easy to calculate. We just need to do as the following:

$$E = K' * F * K$$

- From the Essential Matrix, the rotation and translation between matched features can be determined. Again, it is done by applying a singular value decomposition to the essential matrix and enforcing E to be like the following:

$$E \sim U\ diag(1, 1, 0)\ V'$$

We also define the W matrix as:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And with this we get R and t:

$$R1 = U\ W\ V'$$
$$R2 = U\ W'\ V'$$
$$t1 = U(:,3)$$
$$t2 = -U(:,3)$$

With these, we get 4 solutions for the rotation and translation of the camera pose, but only one is the feasible solution. To solve this, we apply the triangulation method.

- The triangulation method is based on the method used in the function `cameraPose`. The idea is to find the solution that reconstructs the 3D point in front of the camera. There is only one solution that does that. The Richard Hartley's triangulation algorithm relates the intrinsic parameters (world frame) and the current frame (given through the rotation and translation matrices). It passes all the matched points through the algorithm and the four point ratio between the two matched point co-ordinates in the two frames and the two camera centers in those frames. This ratio allows us to find the angle between the perpendiculars of the two frames and hence the direction between two frames can be analyzed. The combination of rotation and translation that relates these set of points is then the solution for the camera poses.

- Once we get the right rotation and translation for the pair of frames, we update the location and rotation of the camera as follows:

$$orientation = R * prev\_orientation$$
$$location = prev\_location \ + \ t * \ orientation$$

Now that we have the camera pose, we can plot it to see how good the algorithm estimates it.

To compare the results, we plot three different versions of the trajectories: The black is the trajectory given by the eight-point algorithm. The red one is the trajectory of the eight-point algorithm refined with Kovesi's RANSAC and lastly, the blue trajectory is the one given by the MATLAB estimateFundamentalMatrix. Triangulation was applied to all of the versions.

It can be seen that the different trajectories start at the same point but they drift with time.