Project 5 ENPM 673

Rene Jacques and Zachary Zimits

May 9, 2019

1 Pipeline

To start the image processing for this project we began by prepossessing all of the images by converting them to color using the *cv2.COLOR_BayerGR2BGR* flag with the *cv2.cvtColor()* function. Next we undistorted the image using the *UndistortImage* function provided with the project data. We then saved all of these frames the naming convention "frame_¡frame number¿.png". This allows for quicker and easier access to this data later in the project.

After all the images have been preprocessed we ignor the first 20 frames as they are over saturated and there is poor matching from the swift algorithm. Starting from frame 20 we process the image with a sift detector and the *detectAndCompute()* function to find keypoints in the frame and save these as the current frame. The process is repeated for the next frame. Using a brute force matcher we find the matches between the two sets and save them in our custom correspondences data set.

2 Fundamental Matrix

For a single pairs of points $[x_i, y_i, 1]$ (image points from first image) and $[x_i^{'}, y_i^{'}, s1]$ (image points from second image) The fundamental matrix can be described by the following equation:

$$\begin{bmatrix} x_i^{'} & y_i^{'} & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0 \tag{1}$$

which can be expanded to:

$$x_i x_i^{'} f_{11} + x_i y_i^{'} f_{21} + x_i f_{31} + y_i x_i^{'} f_{12} + y_i y_i^{'} f_{22} + y_i f_{32} + x_i^{'} f_{13} + y_i^{'} f_{23} + f_{33} = 0 \tag{2}$$

This equation has 9 unknowns meaning that at least 8 point pairs are necessary to solve the system of equations. Therefore the system can be generalized to work with $N/geq8$ point correspondences as follows:

First the center of the data is calculated for the points in both frames.

$$\begin{aligned} c_x &= \tfrac{1}{8} \sum x_i \\ c_y &= \tfrac{1}{8} \sum y_i \\ c_x^{'} &= \tfrac{1}{8} \sum x_i^{'} \\ c_y^{'} &= \tfrac{1}{8} \sum y_i^{'} \end{aligned} \tag{3}$$

Two scaling factors are calculated using the distance between the given points and the center of the data.

$$\begin{aligned} s_1 &= \tfrac{1}{8} \sum \sqrt{(x_i - c_x)^2 + (y_i - cy)^2} \\ s_2 &= \tfrac{1}{8} \sum \sqrt{(x_i^{'} - c_x^{'})^2 + (y_i^{'} - c^{'} y)^2} \end{aligned} \tag{4}$$

These values are then used to calculate the A matrix.

$$\begin{aligned} u_1(i) &= (x_i - c_x)s_1 \\ v_1(i) &= (y_i - c_y)s_1 \\ u_2(i) &= (x_i^{'} - c_x^{'})s_2 \\ v_2(i) &= (y_i^{'} - c_y^{'})s_2 \end{aligned} \tag{5}$$

$$A = \begin{bmatrix} u_2(1)u_1(1) & u_2(1)v_1(1) & u_2(1) & v_2(1)u_1(1) & v_2(1)v_1(1) & v_2(1) & u_1(1) & v_1(1) & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_2(8)u_1(8) & u_2(8)v_1(8) & u_2(8) & v_2(8)u_1(8) & v_2(8)v_1(8) & v_2(8) & u_1(8) & v_1(8) & 1 \end{bmatrix} \tag{6}$$

$$A * \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0 \tag{7}$$

This matrix is defined as A, and the 0 unknowns can be obtained by solving $Ax = 0$. This can be accomplished by applying SVD to A, giving the decomposition USV^T. The last column of V is the solution of the F matrix. However this solution can sometimes lead to F having a rank of 3, which is not the desired rank of F (the desired rank is 2). To enforce the rank 2 constraint we decompose F by applying SVD to the matrix. Then the last singular value of the S matrix is set to 0, meaning that when U, S, and V^T are recombined the resulting F matrix will have a rank of 2.

3 RANSAC

The RANSAC function works in combination with the fundamental matrix calculation. It is initialized with the number of iterations to try, and error value ϵ and the list of correspondences. The algorithm begins by choosing 8 correspondences at random and calculates the fundamental matrix as described in section 2. The algorithm then evaluates the possible fundamental matrix by testing it against each correspondence using the Sampson Error (equation 8) where x_2 is the point in the next image and $x_1$is the point in the current image.

$$\frac{x_2^T F x_1}{(Fx_{1x})^2 + (Fx_{1y})^2 + (F^T x_{2x})^2 + (F^T x_{2y})^2} \leq \epsilon \tag{8}$$

If a point meets this condition it is added to an inlier array. The epsilon value used for our RANSAC function is 0.05. Once all the correspondences have been

checked the algorithm checks to see how many inliers it has found. If that number is higher than any previous iteration than the inliers and the fundamental matrix are saved. The main RANSAC loop runs until the variable N is less than the number of loops measured by the *count* variable. N is initialized as the maximum value contained within the *sys* library, and is therefore effectively infinity. At the end of each loop the ratio of inliers for the loop to number of correspondences is calculated:

$$ratio = \frac{len(S)}{len(correspondences)} \tag{9}$$

This ratio is then used to recalculate N:

$$N = \frac{log(1 - confidence)}{log(1 - (ratio^8)} \tag{10}$$

The $confidence$ value is set as 0.99, meaning that we are 99% sure that all eight of our random points are inliers at least once during the RANSAC function. The *ratio* value is raised to the eight power because we are selecting 8 random points. After N is updated the *count* variable is increased by one.

Using the condition that $N > count$ we can greatly speed up how fast the RANSAC function finds a valid fundamental matrix. Running the RANSAC function for 500 loops and never breaking out results in a processing time of about 30 seconds per frame, while adding the $N > count$ condition results in a processing time of about 4 to 9 seconds per frame.

4 Essential Matrix

The essential matrix calculation uses the F matrix calculated using the RANSAC function described above. The E matrix is calculated as follows:

$$E = K^T F K \tag{11}$$

where K is the camera calibration matrix. In order to correct for possible noise due to K the essential matrix is decomposed using SVD and then recomposed with the singular values $(1, 1, 0)$:

$$SVD(E) = USV^T$$

$$E = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T \tag{12}$$

5 Decomposing E

This newly computed E matrix can now be decomposed into four possible poses. Only one of these poses is valid, which will be determined using linear triangulation (explained in the next section). The poses are created using the U and

V matrices from the SVD of E. The position parts of each pose are determined as follows:

$$\begin{aligned} t_1 &= U[:,2] \\ t_2 &= -U[:,2] \\ t_3 &= U[:,2] \\ t_4 &= -U[:,2] \end{aligned} \tag{13}$$

The last column of U is assigned to each t value while alternating the signs of the vector. This creates all possible translations, which will be combined with all possible rotations to create all possible poses. The rotation matrices are determined as follows:

$$\begin{aligned} R_1 &= UWV^T \\ R_2 &= UWV^T \\ R_3 &= UW^TV^T \\ R_4 &= UW^TV^T \end{aligned} \tag{14}$$

where $W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Therefore the four possible configurations are:

$$\begin{aligned} P_1 &= [R_1, t_1] \\ P_2 &= [R_2, t_2] \\ P_3 &= [R_3, t_3] \\ P_4 &= [R_4, t_4] \end{aligned} \tag{15}$$

The final step was to check the determinant of each R. If $det(R) = 1$ then everything is valid and no changes are necessary. However if $det(R) = -1$ then the signs of both R and t need to be changed (Note: the actual value of R is not guaranteed to be exactly 1 or -1; therefore the actual determinant check is whether $det(R) < 0$).

6 Triangulation

In order to determine which pose is the correct one it is necessary to convert all of the points from 2D coordinates within the images to 3D coordinates within the world. These 3D points will be used later (section 7) to determine the valid pose. For each configuration the equation $Ap = 0$ must be solved in order to get the 3D points p. The basic equations describing the relationship between a set of points X_1 and X_2 and two poses P_1 and P_2 found in equation 17 can be used to create matrix A.

$$A = \begin{bmatrix} X_{1,x} * P_1[2,:] - P_1[0,:] \\ X_{1,y} * P_1[2,:] - P_1[0,:] \\ X_{2,x} * P_2[2,:] - P_2[0,:] \\ X_{2,y} * P_2[2,:] - P_2[0,:] \end{bmatrix} \tag{16}$$

$$P = K * [R| - R * C] \tag{17}$$

Where K is the camera calibration matrix R is the Rotation to get the new pose and C is the translation.
The 3D point coordinates can be found by using SDV shown in **??**

$$\begin{gathered} U, S, V^T = SVD(A) \\ point_{3D} = V^T[3, :] \\ point_{3D} = [x', y', z', c]/c \\ point_{3D,h} = [x, y, z, 1] \end{gathered} \tag{18}$$

The function calculates $point_{3d,h}$ which is a 4x1 homogeneous point for every set of matches passed into the function. It then returns a list of all of the 3D points to by checked by the Cheirality Condition in section 7

7 Cheirality Condition

After the set of triangulated points is returned we reshape them into a nx4 matrix and plug them into (19) with the R and C matrices for a given pose:

$$R[2, :] * (x_i[0:3] - t) \tag{19}$$

We total the number of points in this array with a value and a z value ($x_i[2]$) greater than zero and compare this value to the value from the other poses. Whichever pose had the greatest number of positive terms we take as the best pose and use to find the translation and rotation of the car.

8 Update Pose

From the best pose in section /refcheck we create a 4x4 transformation matrix with the following form

$$H = \begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_{1,1} \\ R_{2,1} & R_{2,2} & R_{2,3} & t_{2,1} \\ R_{3,1} & R_{3,2} & R_{3,3} & t_{3,1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

Multiplying the H with our current pose we get an updated pose where we can extract a new x and z values for the car from $x_{new} = H_{new}[1, 4]$ and $y_{new} = H_{new}[3, 4]$

9 Results

The first version of the code that we thought was working took 32 hours to completely process the entire video. This made debugging and fixing errors in the mapping very difficult. The reason that the code took so long to run is that

our RANSAC function was running for 500 loops for every frame. This resulted in 30 seconds to process each frame. The changes we made to our RANSAC algorithm, as described in Section 3, sped up the processing of each frame to 4-9 seconds, decreasing the full video processing time to about 7 hours. Using a higher epsilon threshold within RANSAC would have decreased the run time further, but 0.05 was chosen due to the consistent results returned by RANSAC while using this threshold.

Our RANSAC returns a fundamental matrix that includes many outliers when graphed (fig 1) but these outliers do not seem to have an effect on the performance of the algorithm. On the contrary when we tried to increase the error term in RANSAC we found the the code ran significantly longer while the odometer did not improve. Because of this we kept a lower error value to speed up the program.

Figure 1: Matches

Our algorithm was able to return a very good result that captured all four turns and straight portions but was unable to close the loop due to accumulated error. The output map matches the path of the car very closely, and also matches the outputs returned from the built in OpenCV functions, which are described in the next section.

9.1 Comparison with OpenCV

We were able to get satisfactory performance with the built in functions. Our output from the builtin functions can be seen in (2). The path generated using our functions is very similar to the path generated using built in functions, and differs due to slight changes in the way that OpenCV calculates the visual odometry compared to the way that we have implemented it. Our map is shown in (3) and the OpenCV map plotted against our map is shown in (4).

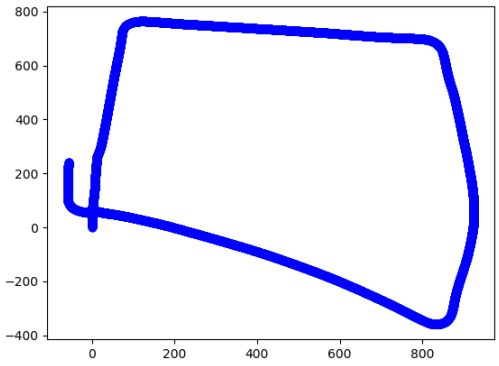

Figure 2: Builtin Output

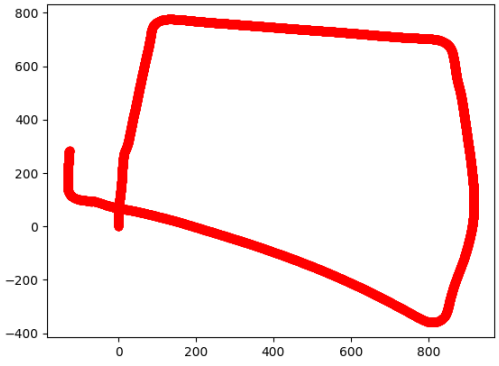

Figure 3: Our Output

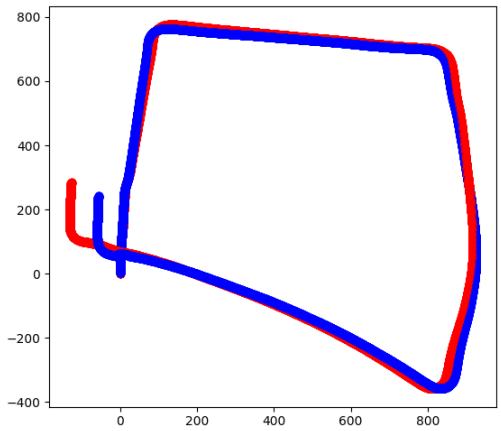

Figure 4: Comparison Output: Blue is the OpenCV map; Red is our map

10 Video Output

Video of the car driving with the outputs from the OpenCV built in functions plotted against our functions can be found here:
https://drive.google.com/drive/folders/1JW8u09emPAmcxrbnH5dZts8AniCMqMU6?usp=sharing)