The Problem

Traditionally stereo vision (generating disparity maps) is achieved by using two synchronized cameras taking the pictures at the same time and in which we know how far the cameras are (the translational matrix **T***) and how are the cameras orientated with respect to each other (The rotational matrix* **R***) by doing the camera calibration once at the start. Now the problem is that we don't want to use two cameras but we want to achieve the exact same thing using only one camera and the* **R** *and* **T** *matrix change with every set of images and plus for camera calibration we traditionally use chess board in which we know the real world distances, while right now we don't know that. What we intend to do is to take a frame, then move the camera and take another frame. So now we have two images and technically we can apply stereo vision. The only problem now is that we don't have the* **R** *and* **T** *matrix in this case. So we can't compute the fundamental matrix* **F***. We now need to figure out how to find* **F, R and T***.*

Mathematical Theory

Let's the vector **x** represent any point in the left image and the vector **x'** represent any point in the right image. The matrix relation between these is:

$$x^T F x' = 0 \tag{1}$$

Also let the matrix **F** be:

$$\begin{bmatrix} \mathrm{a} & \mathrm{b} & \mathrm{c} \\ \mathrm{d} & \mathrm{e} & \mathrm{f} \\ \mathrm{g} & \mathrm{h} & \mathrm{i} \end{bmatrix}$$

So as of now, the problem boils down to finding a, b, c, d, e, f, g, h, i.

No of distinct points required

As the equation (1) suggests, if we have the coordinate of many points in left image and their corresponding points in the right image, we can easily figure out the value of the matrix F. Let's just start off by taking a single point in the left image (x_1, y_1) and the corresponding point in the right image (x'_1, y'_1).

On plugging these values in equation (1), we get

$$\begin{bmatrix} \mathrm{x}_1 & \mathrm{y}_1 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{a} & \mathrm{b} & \mathrm{c} \\ \mathrm{d} & \mathrm{e} & \mathrm{f} \\ \mathrm{g} & \mathrm{h} & \mathrm{i} \end{bmatrix} \begin{bmatrix} \mathrm{x}'_1 \\ \mathrm{y}'_1 \\ 1 \end{bmatrix} = 0$$

After simplifying, we get

$$(x_1x_1')a + (y_1y_1')d + (x_1')g + (x_1y_1')b + (y_1y_1')e + (y_1')h + (x_1)c + (y_1)f + i = 0$$

where we have 9 unknowns but only one equation. Interestingly if we try for 9 points, we can have 9 such equations which can be written as

$$\begin{bmatrix} \mathrm{x}_1x_1' & \mathrm{y}_1y_1' & \mathrm{x}_1' & \mathrm{x}_1y_1' & \mathrm{y}_1y_1' & \mathrm{y}_1' & \mathrm{x}_1 & \mathrm{y}_1 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \mathrm{x}_9x_9' & \mathrm{y}_9y_9' & \mathrm{x}_9' & \mathrm{x}_9y_9' & \mathrm{y}_9y_9' & \mathrm{y}_9' & \mathrm{x}_9 & \mathrm{y}_9 & 1 \end{bmatrix} \begin{Bmatrix} \mathrm{a} \\ \mathrm{b} \\ \mathrm{c} \\ \mathrm{d} \\ \mathrm{e} \\ \mathrm{f} \\ \mathrm{g} \\ \mathrm{h} \\ \mathrm{i} \end{Bmatrix} = 0 \tag{2}$$

which is of the form $AX = 0$. For this system of equations, we should avoid the trivial solution of every variable being zero thus we will only require 8 points.

Note: We will use SVD to solve this system of equations. While the minimum number required is 8, a higher number of points will have high accuracy.

Getting the value of Fundamental matrix F

Thus we can effectively get the solution of this by using just 8 points. On solving, we can easily obtain the values of a, b, c, d, e, f, g, h and i. Please note, it is very essential to make sure that the fundamental matrix is of rank 2 ie. $rank(F) = 2$.

Getting the Essential Matrix E

Fundamental matrix **F** represents the 2D relationship between the point on the left image and the right image while Essential Matrix **E** tries to actually map the relationships in 3D space. In traditional stereo vision, fundamental matrix is obtained from essential matrix using,

$$F = K^{-T}EK^{-1} \tag{3}$$

where **K** is the camera calibration matrix. Using the equation (3) we can easily reverse engineer to get the value of **E**.

$$E = K^TFK \tag{4}$$

It is essential to note here that Essential Matrix also has rank 1 ie. $rank(E) = 2$.

Calculate Projection

Let P_1 and P_2 be the projection vectors from both the images. Now assuming P_1 to be our reference point, we can get the assume P_2 in terms of the Rotational Matrix **R** and **T**.

$$P_1 = K\left[\begin{array}{cc} \mathrm{I}_{3x3} & 0 \end{array}\right] \tag{5}$$

$$P_2 = K\left[\begin{array}{cc} \mathrm{R} & \mathrm{T} \end{array}\right] \tag{6}$$

Relation between E, R and T

Now that we have got E, we can decompose that into **R** and **T**. The relation between these is:

$$E = [T]_\times R \tag{7}$$

Note: Here we take a cross product between the two matrix.

Reconstructing T

This can be visualized with the help of Epipolar Geometry. The ray between the two camera locations plus the rays pointing to the point from both the camera locations form a plane. As you sweep a plane across these, all the epilines will converge into a single point a.k.a **epipole** which will be somewhere in between the line joining the two camera locations. With both the epipoles we can now know the vector **T**. Because the epipole is the projection of the camera location on the image plane of the other camera image on the line joining the two locations, we can say that,

$$P_2\left[\begin{array}{c} 0 \\ 1 \end{array}\right] = \left[\begin{array}{cc} \mathrm{R} & \mathrm{T} \end{array}\right]\left[\begin{array}{c} 0 \\ 1 \end{array}\right] = 1 \tag{8}$$

Now we can also say using principles in traditional stereo vision,

$$T^T E = 0 \tag{9}$$

Which basically means that left nullspace of the essential matrix is the epipole in right image. Thus we can decompose **E** by using SVD as,

$$E = U\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{array}\right]V^T \tag{10}$$

where

$$U = \left[\begin{array}{ccc} \mathrm{u}_1 & \mathrm{u}_2 & \mathrm{u}_3 \end{array}\right]$$

and thus we can say that $T = -u_3$ or $T = u_3$ which are two values.

Reconstructing R

Once the value of **T** is successfully extracted from **E** matrix, we can show using some linear algebra that

$$R = U\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}V^T \tag{11}$$

or

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

where **U** and **V** are the matrix found from SVD of **E** matrix.

Removing extraneous values

Since we are getting two values of **R** and **T**, we have total of 4 permutations of these while only one is valid. Now, it is actually interesting to note what all of these values represent. The correct value represents the point actually in front of both the images. The other values represent the point at back in either of the images or both of them. So to get this we can get the 3D location of the point (this is called as **Point Triangulation** that we are seeing. If the z-coordinate is positive then we have found our correct solution because all other values should be negative since we have assumed left image to be our reference point.

Point Triangulation

We can get the point triangulation by,

$$\lambda\begin{bmatrix} \mathrm{X}_1 \\ 1 \end{bmatrix} = P_1\begin{bmatrix} \mathrm{X} \\ 1 \end{bmatrix} \tag{13}$$

This implies that the cross product of the two vectors must be zero. Thus,

$$\begin{bmatrix} \mathrm{X}_1 \\ 1 \end{bmatrix}P_1\begin{bmatrix} \mathrm{X} \\ 1 \end{bmatrix} = 0 \tag{14}$$

And,

$$\begin{bmatrix} \mathrm{X}_2 \\ 1 \end{bmatrix} = P_2\begin{bmatrix} \mathrm{X} \\ 1 \end{bmatrix} = 0 \tag{15}$$

Thus we will get the x, y, and z coordinate of all the points. Now we can discard the solutions by checking their z coordinate as described in the above section. Note: To get more accurate values, we can take multiple points and use least squares method to get better results.

Actual Implementation

Getting corresponding feature points

As we said above that we can find the fundamental matrix if we get the corresponding points from both the images, we first need to find all of those points. Technically we need only 8 points to uniquely determine the fundamental matrix but since we are dealing with real world coordinates from camera (which aren't perfect) so we will take as many points as possible. We will use linear least square algorithms which can be solved using SVD, with RANSAC to get more accurate results which will work better with more number of points.

To get those points, we can use either **SIFT (Scale-invariant Feature Transform** or **ORB (Oriented FAST and rotated BRIEF)** or **DAISY** algorithm to get the feature points. In this SIFT algorithm is generally more accurate but it takes more computational resources and since we are dealing with a real time problem, it will be better to use **ORB** which is fast and reliable and comes with opencv unlike Daisy.

Why not cv::stereoCalibrate()

This function could have been used which does all of our work to calculate the **R**, **T**, **F**, **K** and **E** matrix for us but unfortunately we can't use this as we need the corresponding points in the actual 3D world that we are going to pass which we don't know. In traditional stereo vision using two cameras, this is not a problem since we use a chessboard in which we know the actual world distances between the two points and we just do it once so as to get all the values. For our problem, we need to calibrate the cameras again and again for every frames that we take in and we won't even know the actual 3D coordinate for the same.

Finding the Fundamental Matrix

Now that we have an array of points of the left image as well as the right image, we can now easily calculate the fundamental matrix using the equation 2. **Luckily** we have an opencv function which can do this for us and also suits are purpose named

```
cv::Mat cv::findFundamentalMat()
```

which takes in two input arrays each of which correspond to their respective points in the left and the right image and gives us Fundamental Matrix **F**. This method uses RANSAC to compute the fundamental matrix and gives it to us after cleaning it up so this is finally of rank 2 as we require.

Decomposing F into E, R, T, P_1 and P_2

This is the most challenging part of this problem. We now have **F** but we don't have **E, R**, **T**, P_1 and P_2. Unfortunately there is no inbuilt method in opencv to actually implement this for us so we will have to type this out all by ourselves. Since we know the mathematics behind it, we can easily use the **eigen** library for linear algebra and compute this. Again in this since we have many points and not just one we will employ SVD algorithm to help us in solving the various matrix equations which in turn gives us an approximate solution. While doing this we will have to be careful about which format we choosing our data to be because that format has to be compatible with the rest of the opencv functions as well because we intend to use the other methods in opencv to simply our job afterwards.

Note: This gives us 4 values of **R** and **T**. We need to remove these by triangulation of points.

Removing extra points by triangulation

As we mentioned above, we are getting 4 values of **R** and **T** while we only require one. Now we choose a point and try to triangulate it with respect to both the image planes so that we can know whether they lie in front of them or at the back. To do this there exists a function named

```
void cv::triangulatePoints()
```

which reconstructs the point by triangulation. We have mentioned the maths behind it above. It takes Projection Matrix P_1 and P_2 and arrays of points corresponding to each image in two separate arrays and outputs the 3D coordinates in a 4D array in which the last one is the homogeneous coordinate. Now we can easily see which points are in the front and thus we can take that particular value of **R** and **T** while we discard the rest.

Image Rectification parameters

Now we have all the values that we require but still the images have a definite epipole. This means that the images aren't exactly "straight" and thus we will need to straighten this out so that we get epipoles at infinity. The benefit of doing this is that now we will need to search for the corresponding point in the x-coordinate only instead of seaching in the whole area. Luckily, we have an opencv function which can do this for us, namely

```
void cv::stereoRectify()
```

which gives us the new R_1, R_2, P_1 and P_2 for each of the image. The better part about this function is that this can even get the value of **Q** which gives us the relation between the disparity map and the depth map which we will need further to show the image in 3D.

Rectified Image

Now we have got the rectification parameters. After this we will have to make the corresponding changes in the image according to those parameters. Luckily, we have an opencv function namely

```
void cv::undistortPoints()
```

which maps the initial point coordinates to their final destination coordinates.

Generating disparity map with Blob Matching

Now we can finally the opencv function,

```
cv::StereoBM::StereoBM()
```

and

```
void cv::StereoBM::operator()
```

which operates only on rectified images to generate the disparity map for us.

Getting the depth map

Since we now have the disparity map and the **Q** matrix we can easily compute the depth map. Luckily there exists an opencv function for the same namely,

```
void cv::reprojectImageTo3D()
```

Finally we now have all of the things required as mentioned in the document that you have shared with us. Hope we understood your problem well and provided the best possible solution we both (Pranit Bauva and Shivang Agrawal) could think of.