Stereo Report

November 29, 2024

1 Introduction

The purpose of this task was to use the various methods taught throughout the Computer Vision section of the IPCV module in order to reconstruct spheres in a scene given two views. Then the model is evaluated against different parameters and weaknesses in the implementations are found. The key stages of the assignment were:

- Using Hough Transform to detect circles.
- Understanding the different coordinate frames and image spaces
- Using epipolar lines to find correspondences.
- Use these lines to use triangulation to find the 3D coordinates.
- Evaluating the scenarios in which the model performs well and poorly.

As a consequence of this task, it was discovered that there are some scenarios where stereo vision alone cannot produce an accurate reconstruction of a scene.

2 Methodology

2.1 Hough Circles

The output of the Hough Circles function provides the centres and radii of all detected circles in view 0, which are necessary for 3D reconstruction.
- Setting the minimum radius too low causes smaller circles to be detected inside spheres due to the shading of the sphere's surface, tricking the Hough detector into thinking the surface curving is the edge.
- Setting the maximum radius too high causes false negative detections of spheres that are partially merged. This is due to the minDist parameter preventing a detection inside another detection. - The quality of input image greatly affects the performance of the circle detector. The image is automatically converted into greyscale. In our case, the spheres are teal, but putting them into the detector converts them to grey. The lack of contrast against the background will make edge detection less reliable.

With the detection of pixel coordinates of the spheres completed, the first step in constraining correspondences between the two views can be completed.

2.2 Epipolar Lines

Camera 0 was assigned the reference view and Camera 1 as the target view.

The transformation matrix was calculated to get from Camera 0 to Camera 1 to allow me to calculate the coordinates of the view 0 circles in view 1. To do this, camera-world and world-camera transformations need to be combined in sequence:

$$H_{01} = H1_{wc}\ H0_{wc}^{-1}$$

The nature of the extrinsic matrices (denoted by H followed by the respective frame transformation) enables the rotation matrix, R, and translation vector, T, to be extracted.

Where T is a vector of the first three elements of the fourth column, and R is the matrix of first three elements of the first three columns.

$$H = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

I calculated the skew-symmetric matrix S using T as defined in lecture

$$(\mathbf{T} \otimes \mathbf{P}_L) = S\,\mathbf{P}_L$$

$$S = \begin{bmatrix} 0 & -T_Z & T_Y \\ T_Z & 0 & -T_X \\ -T_Y & T_X & 0 \end{bmatrix}$$

This enabled me to calculate the Essential Matrix using

$$E = SR$$

From this, the Fundamental matrix was calculated using

$$F = K^{-T} E K^{-1}$$

Where K is the intrinsic matrix (defined by the focal length and image dimensions), and it transform image coordinates into pixel locations. The inverse is used to transform from pixel to image coordinates.

The Fundamental matrix applies the same transformation as the Essential matrix but directly into the pixel coordinate system.

The coefficients of the epipolar line constraint were then calculated using

$$\mathbf{u}_0 = F\mathbf{p}_0 = [u_{01}, u_{02}, u_{03}]$$

which is an adaptation of this formula from the 3D reconstruction lecture, but using the fundamental matrix since we are dealing with pixel coordinates.

$$\boxed{\text{Let} \quad \mathbf{u}_L = E\,\mathbf{p}_L = \begin{bmatrix} u_{L1} \\ u_{L2} \\ u_{L3} \end{bmatrix}}$$

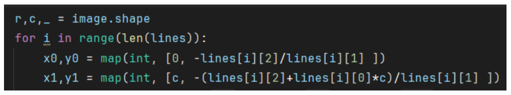
The epipolar constraint is defined as follows:

$$\mathbf{p}_R^T E \mathbf{p}_L = 0$$

or working with pixels:

$$\hat{\mathbf{p}}_R^T F \hat{\mathbf{p}}_L = 0$$

This allows us to create a linear system to solve for all possible values for x and y. This code constrains the variables to within the image dimensions, while also solving the system.



```
r,c,_ = image.shape
for i in range(len(lines)):
    x0,y0 = map(int, [0, -lines[i][2]/lines[i][1] ])
    x1,y1 = map(int, [c, -(lines[i][2]+lines[i][0]*c)/lines[i][1] ])
```

For each of the lines, the program extracts the endpoints, which allows for the lines to be displayed. This implementation of epipolar lines enables subsequent triangulation.

2.3 Correspondences

The program iterates through the detected centres from view 1 and finds the epipolar line that has the shortest distance.

A drawback of this is that multiple circles could be assigned to the same epipolar line. To combat this, an adapted version of the Hungarian Algorithm should be implemented to find the minimal distance matchings.

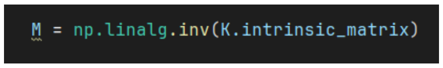
The view 1 centres and radii arrays have now been assigned an epipolar line, and been reordered accordingly.

2.4 3D Reconstruction

The M matrix is calculated as defined in the lectures:

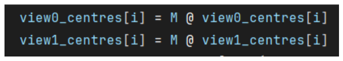
$$\begin{matrix} x = s_x(\hat{x} - \hat{o}_x) \\ y = s_y(\hat{y} - \hat{o}_y) \end{matrix} \Rightarrow \mathbf{p}_L = \begin{bmatrix} x_L \\ y_L \\ f \end{bmatrix} = M_L \begin{bmatrix} \hat{x}_L \\ \hat{y}_L \\ f \end{bmatrix} = M_L \hat{\mathbf{p}}_L$$

where it represents the transformation applied to pixel coordinates to get to image coordinates (which is the opposite of the intrinsic matrix). So this code achieves it:



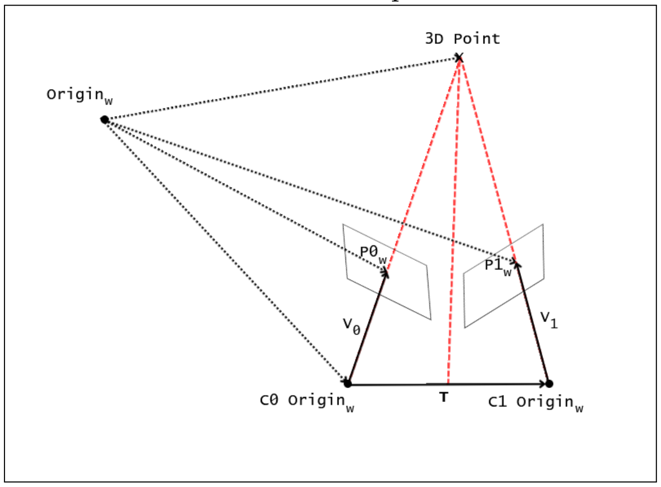
```
M = np.linalg.inv(K.intrinsic_matrix)
```

From this point, I transformed all the centres of both view 0 and 1 into image space by applying transformation matrix M



```
view0_centres[i] = M @ view0_centres[i]
view1_centres[i] = M @ view1_centres[i]
```

The goal of 3D reconstruction is to get the circle centres into the world view as 3D points.



Reconstruction Diagram

The next stage is converting all the homogeneous centres from each camera view into world view. This is done using the inverse of the extrinsic matrices. The origin of each camera view (from the world view) is subtracted from the centre points to transform $P0_w$ and $P1_w$ into v_0 and v_1 vectors. All the transformations had now been applied to be able to triangulate the 3D point. This is equivalent to the stereo transformation $P_0 = R(P_1 - T)$

The following sequence of calculations from the lectures were then carried out on the transformed points:

$$a \underset{3\text{x}1}{\begin{bmatrix} \bullet \\ \mathbf{p}_L \\ \bullet \end{bmatrix}} - b \underset{3\text{x}1}{\begin{bmatrix} \\ R^T \mathbf{p}_R \\ \\ \end{bmatrix}} - c \underset{3\text{x}1}{\begin{bmatrix} \\ \mathbf{p}_L \otimes R^T \mathbf{p}_R \\ \\ \end{bmatrix}} = \underset{3\text{x}1}{\begin{bmatrix} \\ \mathbf{T} \\ \\ \end{bmatrix}}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = H^{-1}\mathbf{T}$$

$$\hat{\mathbf{P}} = (a\,\mathbf{p}_L + b\,R^T\mathbf{p}_R + \mathbf{T})/2$$

Resulting in correct 3D point of each point pair in the world view.

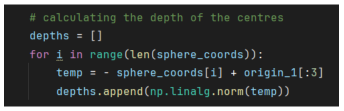
2.5 Displaying reconstruction in 3D

The code calculates the closest ground truth to each 3D reconstruction. Like the epipolar lines distances, this has a chance of assigning multiple spheres to a single ground truth, so a fix for this would improve robustness of the model especially when the spheres are close together.

The error calculated is the Euclidean distance between the correspondences. Then, it just adapts the code from the skeleton to display both ground truths and estimated centres in the visualiser.

2.6 Estimating Radii

To calculate the radii of the spheres, we have to use the radii output from the Hough circles function. First the depth of the image needs to be calculated. Given that the estimated radii we have are in view 1's frame, the centres need to be adjusted so their origin is at view 1's origin. The depth is just the magnitude of this vector.



```
# calculating the depth of the centres
depths = []
for i in range(len(sphere_coords)):
    temp = - sphere_coords[i] + origin_1[:3]
    depths.append(np.linalg.norm(temp))
```

The radii are then calculated by applying the formula

$$\mathbf{P}_1 = \frac{Z_1 \mathbf{p}_1}{f}$$

where $\mathbf{P}_1$ is the real 3D vector and $\mathbf{p}_1$ is the 2D homogeneous pixel vector. The result of this is a 3D vector representing the vector from the centre of the circle to an arbitrary point on the circumference. The magnitude of this is the radius.

2.7 Displaying Spheres

Displaying the spheres adapts the skeleton code, creating new spheres with the calculated radii and centres. They were rendered in wiremesh to allow for easy comparison of the ground truths and estimated spheres.

2.8 Adding Noise

Standard normal distributions were sampled from with a adaptable scalars to get noise for the rotations and translations. Rotation matrices were defined as follows:

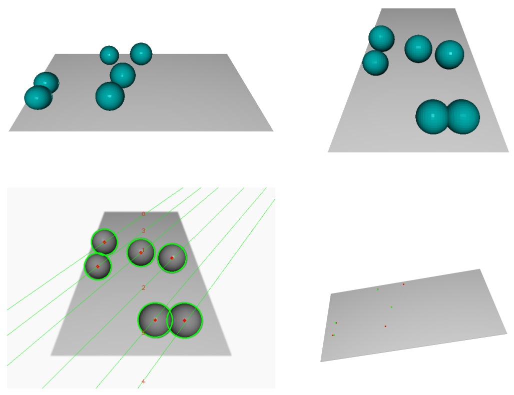
$$R = R\,X\,Y\,Z$$

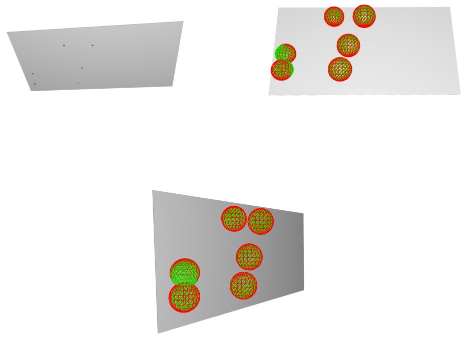
Where X, Y, Z are the random rotations about their respective axis.

3 Experiments and Results

3.1 Standard Results

These are the results expected with no additional command-line parameters, showing each stage of the 3D reconstruction.

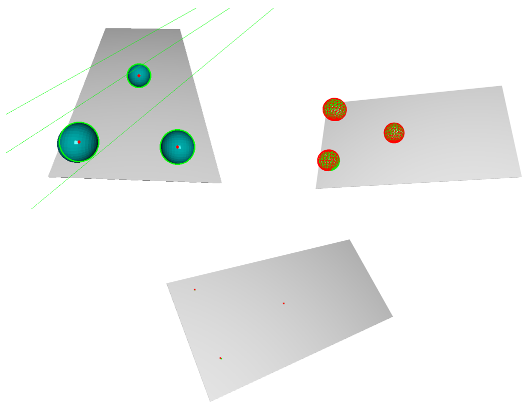




3.2 Sphere Number

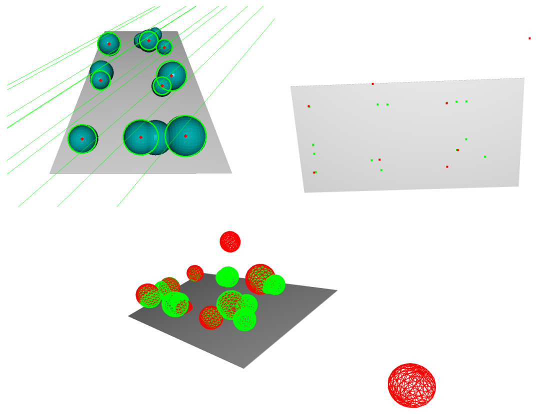
Error data and analysis from varying the number of spheres and calculating the error.

3.2.1 Low



Error between correspondences: 0.06199126 **Radius error:** 0.16073348 The performance was generally good on a low (1-7) number of spheres. The fewer the sphere there are, the lower the chance of (1) spheres obscuring each other in at least one of the images, (2) spheres being close to one another's epipolar line, at which point a sphere gets assigned to an incorrect epipolar line.

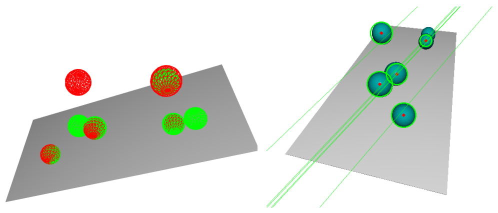
3.2.2 High



Error between correspondences: 5.04774154 **Radius error:** 0.90037283 These error rates are roughly twice of that than with a low number of spheres. The error is being skewed by the extrema, which are the mismatches. The performance is poor. There are many spheres that have not been highlighted by the hough circle function and many epipolar lines pass through multiple spheres. Spheres not identified in the first view won't have an epipolar line in the second view. As a consequence, the correspondences could be mixed up.

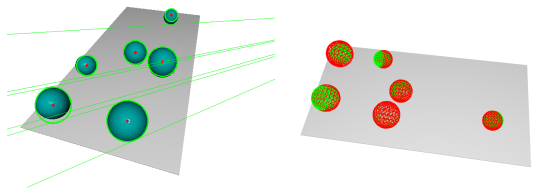
3.3 Sphere Distance

3.3.1 Close spheres



Error between correspondences: 0.48999840 **Radius error:** 0.13689976

3.4 Far spheres



Error between correspondences: 0.11114029 **Radius error:** 0.26040827 Error rate is much lower if the spheres are further apart (as expected).

3.5 Effect of Noise

The program calculates the error rate before noise is applied and after noise is applied for the same visualisation state.

3.5.1 No Noise

Error between correspondences: 0.114 **Radius error**: 0.192

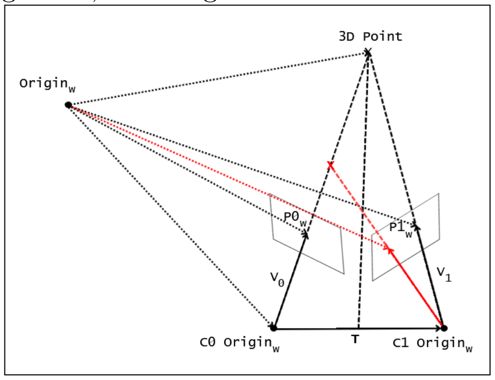
3.5.2 With Noise

Error between correspondences: 7.44818798 **Radius error:** 0.56691320

3.6 Interpretation & Analysis

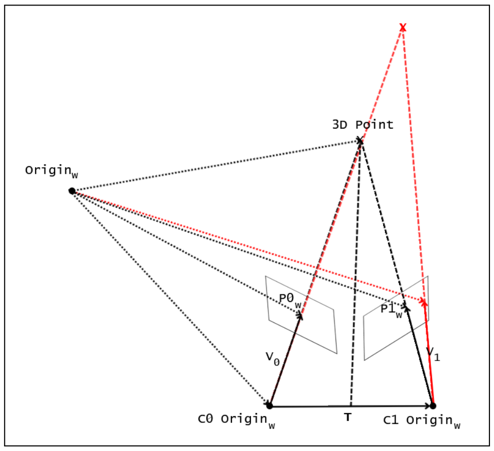
If two spheres are incorrectly matched by the epipolar lines, the consequences will affect the 3D position and radius of the output sphere. In the case where the corresponding vector intersects with camera 0's vector, v_0, closer than ground truth does, then calculating the depth using the Euclidean distance will result in a sphere that is closer to camera 1's origin.

Calculating the radius using $\mathbf{P}_1 = \frac{Z_1 \mathbf{p}_1}{f}$ with the distance to camera 1, Z_1, decreasing will result in the estimated radius being lower than actual. If the observer views a such a sphere from roughly the same angle and position as camera 0 or 1, it will appear to be a normal-sized sphere on top of the plane. As soon as the camera is moved, the parallax effect will cause the sphere to move across the image faster than the background, breaking the illusion.



The results of the error in red, appearing closer to the camera

When the corresponding vectors intersect further away than they should be, the opposite occurs, the 3D coordinate is further away from camera 1, so the radius will have to increase to compensate.



The results of the error in red, appearing further from the camera

Another scenario in which epipolar geometry has poor performance is when all the spheres lie on or close to the same epipolar line. At which point, stereo vision is not an appropriate method to use.

4 Conclusion

To conclude, stereo vision is a powerful method to reconstruct the 3D scenes given that there are qualities which are favourable (highly spread points to reconstruct, little or no noise, fewer spheres to detect). The performance of the models is highly dependent that the spheres are spread out enough. The code has oversights when there are fewer spheres than actual are detected. A real-life camera system may have noise involved, so adjusting the code to account for this could be an extension for this. For example, a probabilistic machine learning approach to estimate the distribution the noise in order to reduce it.

References