COMPUTER VISION COURSEWORK 2023
George Atkinson

Part II – Stereo Vision

Introduction

In this section I will explain my solution to the stereo vision problem. The task at hand is when given two 2-Dimensional views of a plane containing a number of randomly positioned spheres of varying radii – how can we use epipolar geometry and stereo vision to reconstruct this scene in 3D.

When running the program – you can optionally give it 7 different arguments: Firstly, there is 'num' which is the number of spheres (default 6) being plotted on a plane in the scene. Secondly, there is 'sph_rad_min' (default 10) and 'sph_rad_max' (default 16) which are the minimum and maximum radii of the spheres, respectively. Then, there is 'sph_sep_min' (default 4) and 'sph_sep_max' (default 8) which are the minimum and maximum separations of our spheres. Finally, there are 'display_centre' and 'coords.' These are both available for visualization support: with 'display_centre' indicating to open an extra visualization window showing just the centres of the circles, and 'coords' indicating the axis directions on the plane.

From performing the reconstruction of this scene, I have discovered how a cameras intrinsic and extrinsic matrix can be used to determine geometric relationships between points in different cameras. In this report, I will give an overview of how I successfully implemented a system to reconstruct the 3D scene with high accuracy.

To begin with, I will complete the tasks for the default values, and then afterwards I will study how changing these values will affect the performance of my model. Upon completion of this section, I will have a robust system for reconstructing spheres in 3D scenes.

Task 1 - Hough Detection

The initial task given a 2D image from each camera, is to detect the circles in this image. Unlike in section 1, I will detect Hough circles using cv2.HoughCircles. To prepare the image for Hough circle detection – I first gray scaled the image and then applied a 9x9 gaussian filter to it with the aim of removing the contour lines of the spheres from the image views. With this, I proceeded to use cv2.HoughCircles on my blurred grey image.

To do this, I used standard threshold parameters and made estimates on the minimum and maximum radius and the minimum distance between spheres. The results I found were disappointing as my system would quite often detect less than 6 spheres, or even detect the same sphere twice (as can be seen in figure 1). As well as this, the radius was often slightly inaccurate due to the heavy blurring I was doing.

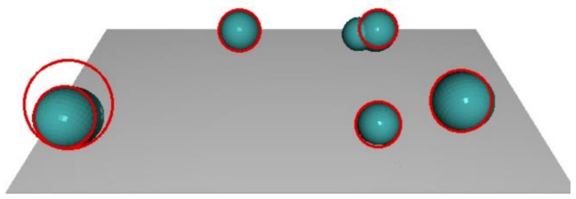

Figure 1: example of Hough circles after blurring image too much.

My first goal for improving this was that I wanted to always detect 6 circles in the image. To do this, I made my param1 argument lower than I would usually. By lowering this threshold, it means that more circles are picked up by cv2.HoughCircles. Then, knowing that the returned list of circles from cv2.HoughCircles is in descending order of votes, I take the first six results from the list. This means that, in theory, I should not miss any circles.

Then, to improve my radius estimates I changed my methods for blurring. I noticed that the image from camera 1 needs more blurring due to the angle it is viewing the circles from giving the contours more definition. So, for image 1 I did 2 5x5 median blurs followed by a 5x5 gaussian blur and for camera 0 I did only 1 5x5 median blur followed by a 5x5 gaussian blur using the following kernel:

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

With this, I am now very consistently detecting all 6 circles to a good accuracy (as can be seen in figure 2).

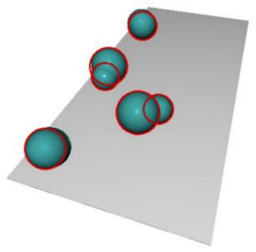

Figure 2: accurate Hough circle detections, even on occluded spheres.

Task 2 – Finding epipolar lines

With my model now successfully detecting the circles in the camera images, I aimed to draw an epipolar line from the circle centers in the image plane of camera 0, to the center of each circle in the reference image of camera 1.

To fully understand all of the transformations that happen in the process of scene reconstruction, you need to understand the different coordinate systems at play. Firstly, there are the world coordinates – these are actual coordinates of the scene. Then, there is the camera coordinates. These coordinates are aligned with the cameras position in the scene. Finally, there are the image coordinates – these are the 2D coordinates of an image captured by one of the cameras.

Epipolar geometry entails dealing with the geometric relationship between two cameras observing a 3D scene. An epipolar line, is the intersection of the image planes of the two cameras with a common point in the 3D scene. An example of this can be seen in figure 3, with the epipolar lines projecting through each image plane and intersecting in the world. In this case, the points of intersection in question are on the centres of the spheres.

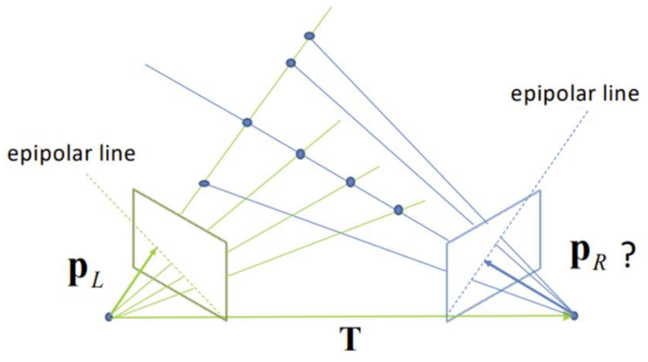

Figure 3: a visualization of the intersection of image planes for a left camera and a right camera observing the same scene.

It follows that if you were to draw an epipolar line from the right camera to the scene – then any image taken from the right camera would not contain this line since it is completely orthogonal to the image plane.

To solve this problem, we need to understand the properties of the cameras that we are using. These cameras contain an intrinsic matrix which is used to project 3D camera coordinates into 2D image coordinates on the cameras image plane. The intrinsic matrix of our pinhole camera model is:

$$\begin{bmatrix} f & 0 & w \\ 0 & f & h \\ 0 & 0 & 1 \end{bmatrix}$$

Where f is the focal length, with w and h representing the width and height of the image plane respectively.

Furthermore, they have an extrinsic matrix describes the cameras position in the world coordinates. This matrix projects worlds coordinate system into camera coordinates. The extrinsic matrix of camera 0 in this case looked like:

$$H0_{wc} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 20 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where theta is a varying value between 70 and 90 degrees. This matrix is built with two sections. The first 3 rows and columns represent a rotation matrix, and the first three rows of the last column is a translation matrix. Breaking this up into two matrices R and T looks like:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} T = \begin{bmatrix} 0 \\ 0 \\ 20 \end{bmatrix}$$

Camera 1's extrinsic matrix is also a 4x4 matrix with the same format, just different rotation and translation matrices due to its different location in the world.

Then, to draw the epipolar lines, I needed to create a matrix that can transform camera 0 coordinates to camera 1 coordinates. To do this, I first recalled that the extrinsic matrix of each camera converts the world coordinates to camera coordinates, and that inversing the extrinsic matrix would convert camera coordinates to world coordinates.

With this, I multiplied the extrinsic matrix of camera 0 by the inverse of the extrinsic matrix of camera 1 to receive a matrix that converts camera 1 coordinates to camera 0 coordinates – let's call this matrix H_10.

$$H_{10} = H0_{wc} * H1_{wc}^{-1}$$

Now, I can divide H_10 into a rotation and a translation matrix as mentioned before, let these be R and T. Now, I can use T to form the matrix S – which is the matrix that satisfies the equation:

$$(T \times P_L) = SP_L$$

From this, I can then derive S by using the properties of the cross product such that:

$$T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \; S = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

From here the essential matrix, E, is simply the result of multiplying R and S. The essential matrix is a useful way to encode the relationship between the two cameras – and how one camera can be transformed into the other.

$$E = RS$$

With the essential matrix I could now calculate the fundamental matrix, F. The fundamental matrix can be used to map a point in one camera's image plane, to its corresponding epipolar line in the other image. The fundamental matrix is calculated by:

$$F = M_0^T E M_1$$

Where M_0 and M_1 are the intrinsic parameters of camera 0 and camera 1 respectively.

To get the fundamental matrix. I first needed to create a matrix that can transform the pixel coordinates from image 0 to camera 0 coordinates. To do this I defined M_{pi} as:

$$M_{pi} = \begin{bmatrix} sx & 0 & -\frac{ox}{f} \\ 0 & sy & -\frac{oy}{f} \\ 0 & 0 & 1 \end{bmatrix}$$

Luckily, M_{pi} is the same for both camera 0 and camera 1 as they have the same intrinsic values. To compute the fundamental matrix, I simply used the equation:

$$F = M_{pi}^T E M_{pi}$$

With the fundamental matrix – for any point in either image, I can find the corresponding epipolar line in the other image. From here, all that is required is to multiply each circle center in the reference (from our Hough detections) by the fundamental matrix. As a result, for each detected circle center in one image, in the reference image we get an epipolar line u such that:

$$u = F \begin{bmatrix} x \\ y \\ f \end{bmatrix}$$

Finally, I had my epipolar line in the reference view, I just needed to plot it onto our 2D reference view. To achieve this, I used line u's gradient to plot it in 2D by computing the following:

$$p_0 = \begin{bmatrix} 0 \\ -f\frac{u_x}{u_y} \end{bmatrix} \; p_1 = \begin{bmatrix} w \\ \frac{-fu_z + u_x w}{u_y} \end{bmatrix}$$

Finally, with p0 and p1 being the ends of my line, I used cv2.line to achieve the results in figure 4 which shows the 6 successfully drawn epipolar lines.

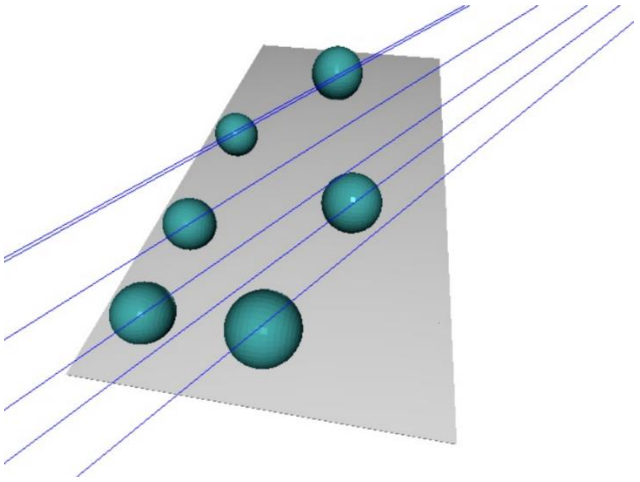

Figure 4: epipolar lines drawn from center of projection of camera 0 to the centers of spheres in reference image.

Task 3 – Finding corresponding circles

Now to find out which circles detected in each respective Hough image corresponded with each other, I needed to do epipolar correspondence. To do this, I needed to compute:

$$p_1^T F p_0 = 0$$

If I find two points from separate images, p0 and p1, that give 0 when substituted in, then they correspond to each other in the world coordinates. This is because by multiplying a point, p0, in image 0 by the fundamental matrix, we get an epipolar line in image 1 (u = F.p0). Then, by multiplying this with the corresponding point in image 1, p1, we know that if this p1.F.p0 is equal to zero, then it means that the p1 is coplanar with the line F.p0. As a result, we can deduce that p0 and p1 are corresponding.

Finally, I computed this for every pair of points but none of my values were coming anywhere near to zero. I put this down to the large values of our intrinsic camera matrices (such as focal length of 415) as well as slight inaccuracies in calculating pixel values. Instead, I found that minimizing the result of p1.F.p0 still finds two corresponding points, so for each point I found it a pair that minimized p1.F.p0.

Task 4 – 3D Reconstruction of sphere centers

Now knowing which circles correspond to each other, the next task was to project their centres into world coordinates. To do this, I set up the following simultaneous equation where we must find a, b, and c such that:

$$ap_1 - (bR^Tp_0 + T) - c(p_1 \times R^Tp_0) = 0$$

This equation is basically saying that these vectors will go in a circle and get back to the starting point if summed. This can be visualized by the following diagram:

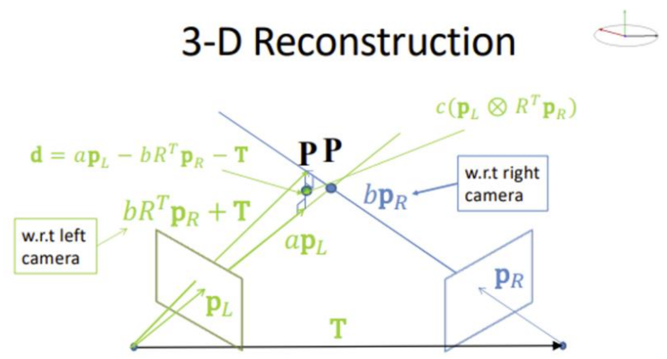
3-D Reconstruction
$c(\mathbf{p}_L \otimes R^T\mathbf{p}_R)$
$\mathbf{d} = a\mathbf{p}_L - bR^T\mathbf{p}_R - \mathbf{T}$
P P
w.r.t right camera
$b\mathbf{p}_R$
$bR^T\mathbf{p}_R + \mathbf{T}$
w.r.t left camera
$a\mathbf{p}_L$
$\mathbf{p}_R$
$\mathbf{p}_L$
T

Figure 5: a visualization of the 3D reconstruction simultaneous equation

Then, I solved this simultaneous equation by setting it up with matrices such that:

$$\begin{bmatrix} p_1 \\ -bR^Tp_0 - T \\ -p_1 \times R^Tp_0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = T$$

And then I solved for a, b, and c by doing the following:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} p_1 \\ -bR^Tp_0 - T \\ -p_1 \times R^Tp_0 \end{bmatrix}^{-1} T$$

Now with values of a, b, and c, I just needed to substitute these into the following equation to get P.

$$P = \frac{(ap_1 + bR^Tp_0 + T)}{2}$$

Finally, I repeated this process on each Hough circle center in the reference image to compute an estimated sphere center for each pair of corresponding circles. As a result, I have a list of reconstructed sphere centers for all 6 spheres.

Task 5 – Plotting centres and computing error

To plot my calculated sphere centres, I simply iterated through the list that I curated in the previous section, plotting a small green sphere (radius 0.01) on every point. Then, I accessed the ground truth values through GT_cents and drew a small red sphere on each of these points. Finally, I calculated the Euclidean distance using the following equation:

$$Dist(\mathbf{p_1}, \mathbf{p_0}) = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

I saved these results in a csv file so that I could later analyze the radius error in my reconstruction.

To finish my visualization of the error, I drew a blue line between the centres of these spheres to represent the Euclidean distance between them. These results can be seen in figure 6.

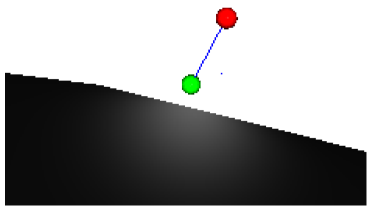

Figure 6: a 3D visualization showing the ground truth sphere center (red), the reconstructed sphere (center), and the normal line between them (blue).

Task 6 – Computing sphere radii

With the Hough circle radius being on the scale of the image plane, to calculate the world radius of the sphere all I had to do was find how much the circle is scaled once projected into the world. To do this, all I had to do was multiply the circle radius from each camera by the reciprocal of the extrinsic matrix for each respective camera.

I did this for the radius in each Hough image, and then I took the mean of the two values to reduce error in my calculation. I also experimented with taking the maximum of the two calculated radii, but I found taking the mean worked better because my Hough circles tend to be quite accurate.

Then, I used these values as well as the previously calculated centres to plot the reconstructed spheres in green next to the ground truth spheres which are in red. However, I found that in this form it was hard to see how good my results were. An example of this is when a reconstructed sphere had its center calculated accurately but its radius was calculated too large. This can be seen in figure 7.

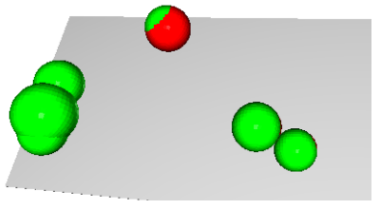

Figure 7: Reconstructed spheres plotted with ground truth spheres – but unable to see most ground truths as they are engulfed by the reconstructed spheres.

This meant that you could not see the reconstructed sphere since it was completely hidden by the ground truth sphere. To counteract this, I made a point cloud using TriangleMesh.sample_points_poisson_disk. Then, I plotted these points in green and red for the reconstructed and ground truth spheres respectively achieving the results in figure 8.

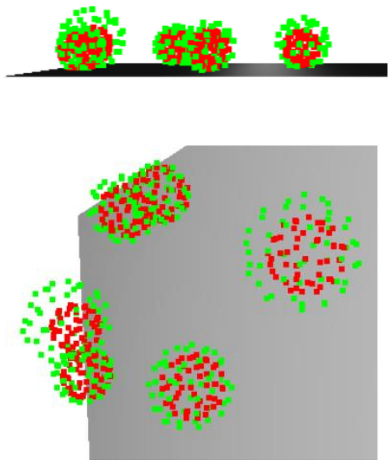

Figure 8: Reconstructed spheres plotted with ground truth using point clouds

These results are for easier to interpret and they show that the system has successfully reconstructed the spheres in the scene.

Task 7 - Performance evaluation

Now with my model working remarkably well by eye test, it was time to gather data to evaluate the performance of my system.

The first thing I analyzed was the error in sphere center calculations. To do this – I found the Euclidean distance between the ground truth and reconstructed center for each sphere, and then I averaged them. I repeated this 5 times and gathered the results in figure 9.

Run	Mean Center Error
1	0.165
2	0.440
3	0.186
4	0.545
5	0.380

Figure 9: A table showing the mean error in center calculation.

From my experiment, I found that the centers calculated were very accurate for the most part. Some of the mean values were slightly inflated by an anomaly (can be as high as 3) that has occurred in a situation like can be seen in figure 10.

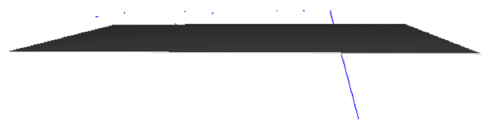

Figure 10: a blue line showing an extremely miscalculated sphere center.

After this, I proceeded to analyze the error in radius calculations in a similar way. I compiled these results in figure 11 and I found that my radii errors were generally good, but my model tended to slightly overestimate the ground truth radius.

Run	Mean Radius Error	Mean Radius Percentage Error
1	0.11	9%
2	0.20	17%
3	0.32	27%
4	0.20	15%
5	0.25	21%

Figure 11: A table showing the error in the reconstructed radii.

In addition to these calculations, I also made estimates of the volume overlap between the reconstructed spheres and the ground truth spheres. I then calculated the percentage volume overlap by dividing the overlap with the ground truth volume. From this, I classified a reconstruction as a true positive if the volume overlap was over 50% of the ground truth volume. From this, I ran the simulation 5 times and got an impressive average TPR of 0.83.

Task 8 – Model weaknesses

Although my model works well on the default parameters – it is unfortunately not very likely that I will want a model to detect exactly 6 similarly sized spheres.

For this reason, it was time to try and break my model by changing the input parameters of the simulation.

To begin with, I began by adding more spheres to the simulation – meaning that my program had to detect 10 spheres instead of the typical 6. The results of this can be seen in figure 12, with my system acting robustly and potentially performing even better.

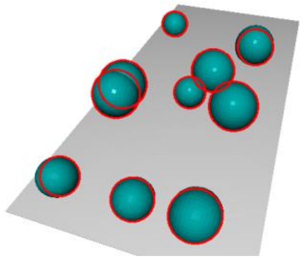

Figure 12: Hough circles of 10 spheres

I did similar calculations as in the previous section and calculated an excellent TPR of 0.95 across 5 repeats of the simulation.

However, when reducing the number of spheres to 3, my system performed significantly worse. An exhibition of this is in figure 13, in which the Hough detection has incorrectly found the same circle twice. Unfortunately, this was quite a common occurrence and resulted in a TPR of 0.8 across 5 repeats.

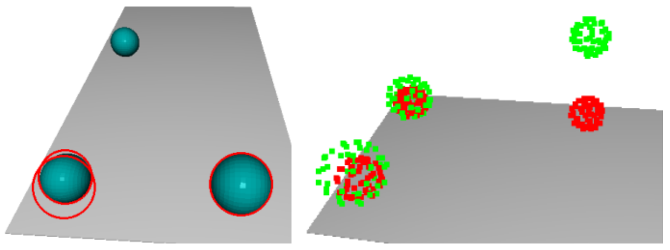

Figure 13: Hough cicles and reconstructions of 3 spheres

Next, I began to experiment with the min and max radii of the spheres. Immediately, I had low hopes for my system here as my Hough circle detection is very fixed in the radius range it searches.

Firstly, I tried making the radius range larger, to anywhere between 20 and 25. This was catastrophic for my system, with most of the spheres going undetected by the Hough detector. This can be seen in figure 13 where there are lots of smaller circles on a single sphere because it is out of its radius range.

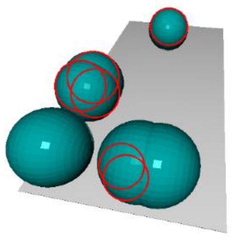

Figure 13: Hough circles of larger spheres exhibiting limit of my system.

Subsequently, I tried making my spheres smaller, specifically, in the radius range of 1 to 3. With this bound on the radius, my system did not detect any circles at all, which means it achieved a devastating TPR of 0.

Finally, I had to test where the minimum and maximum separation affected the performance of my system. After the previous failures, I had low hopes for my systems performance when moving the spheres close together. I ran my code 5 times with a separation in the range of 1-3 and I was pleasantly surprised to see my TPR remained at a strong 0.86. An example of this can be seen in figure 14 with all of my point clouds coinciding with each other.

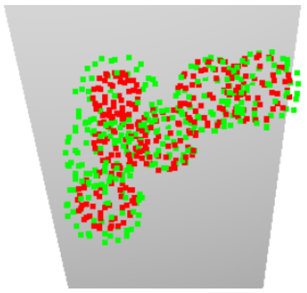

Figure 14: Point clouds of my reconstructed spheres with a lower separation

Moving onto a higher separation, I found that my code would no longer run. I believe this may be to do with the simulation being too computationally expensive with them being further apart. Furthermore, it could be to do with spheres being plotted outside the limits of the plane, causing the scene to be incredibly large.

Task 9 – Conclusion

Whilst my system for reconstruction is quite robust, it has some major downfalls – such as when the radius is too large or too small which causes my system to detect very few circles.

To improve my implementation, I would try doing multiple iterations of Hough circle detections, each with different radii bounds – and then attempt to cluster them. Hopefully by doing this it would make my system more reactive to varied radii. Alternatively, I could implement a more advanced object detection (such as SIFT) algorithm to detect the spheres so that the detections are less limited.

In addition, if I had more time, I would have liked to analyze how adding noise to the camera poses would affect the performance of my implementation. By doing this, I would be making my system more similar to real life where the camera coordinates will not be defined as precisely.

However, overall, I have made a very good implementation. In summary, I have successfully used the extrinsic and intrinsic camera matrices as well as my knowledge of epipolar geometry to accurately reconstruct the spheres in the randomly generated scene. As well as this, I performed experimental analysis on my system, from which I was able to determine the strengths and weaknesses of my system.