Project 4: Multi-camera Geometry

Param Vipulkumar Chokshi
University of South Florida
Department of Computer Science and Engineering
pvc@usf.edu

Abstract

Multi-camera geometry involves the understanding of relative positions of the cameras in the world. Geometrical approaches can be used for multiple applications like pose estimation, depth estimation, 3D geometry recovery, and in setting up multi camera setups used for various purposes like movies and sports broadcast.Epipolar geometry helps in establishing relation between two cameras or between two frames taken from a moving camera, This forms our basis for working with multiple cameras. The exercise mentioned in the report demonstrates concepts of epipolar geometry and also an application of height estimation using 3D-estimates on images taken form WILDTRACK dataset.

1. Introduction

The project aims at studying and exploring the estimation of 3D geometry using multiple-camera setups. It aims at developing a deeper understanding of epipolar geometry and constraints among images collected with multiple cameras.

1.1. Epipolar Geometry

- For every point in one of the images, its corresponding point has to be along the **epipolar line**.
- The 3D point and the centers of the camera define a **epipolar plane**.
- The epipolar plane intersects the two image planes along **epipolar lines**.
- Points on corresponding epipolar lines between the two images have to be match. This is known as the **epipolar constraint**.
- Note that the same relationship exists for any 2-frames of a motion (video) sequence from a moving camera.
- In the figure below, the three vectors from $\overline{\mathbf{c}_0\mathbf{c}_1}$, $\overline{\mathbf{c}_0\mathbf{p}}$, and $\overline{\mathbf{c}_1\mathbf{p}}$, all line on a plane. We ignore the fact that image locations are quantized in pixel units and the rays might not pass exactly through the same 3D point in practice.

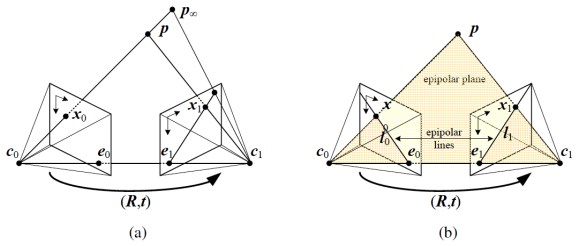


Figure 11.3 Epipolar geometry: (a) epipolar line segment corresponding to one ray; (b) corresponding set of epipolar lines and their epipolar plane.

Figure 1: Epipolar Geometry

1.2. WILDTRACK Dataset

'WILDTRACK' dataset brings multi-camera detection and tracking methods into the wild. It meets the need of the deep learning methods for a large-scale multi-camera dataset of walking pedestrians, where the cameras' fields of view in large part overlap. Being acquired by current high tech hardware it provides HD resolution data. Further, its high precision joint calibration and synchronization shall allow for development of new algorithms that go beyond what is possible with currently available data-sets.

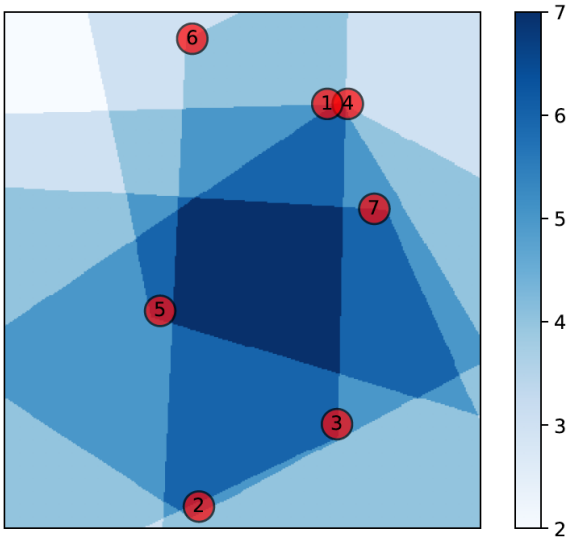


Figure 2. The overlap between the cameras' fields of view (top view). See § 3.1.

Figure 2: Camera view overlap

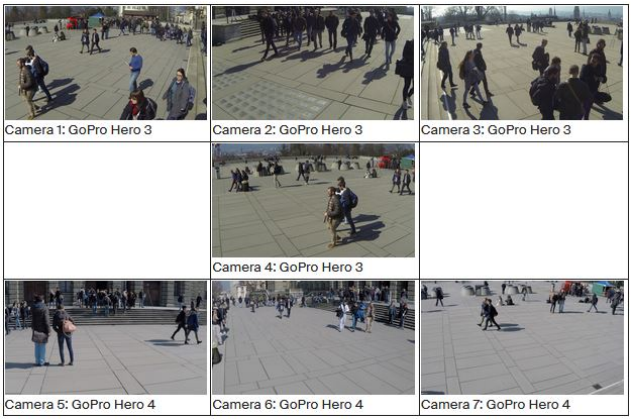


Figure 3: Camera views

2. Experimental Setup

The general setup for implementing the algorithms involved reading xml files available from the WILDTRACK data-set to initialize list of intrinsic and extrinsic parameters for each of the seven cameras. Once the the parameters are read from the files, they can be used in the the calculations stated in the later sections.

The environment used to implement these project comprised of python along with some essential libraries.

Image Processing: imageio, skimage
Linear algebra: numpy
Plotting: matplotlib
Scientific computing: SciPy

3. Functions Implemented

3.1. Camera Placement in terms of World Coordinates

The function implemented reads in the intrinsic and the extrinsic calibration information from xml files from the WILDTRACK dataset, creates a plot of camera locations in the world coordinates. The plot shown in Figure 4 shows the resulting plot. The plot contains the world coordinates, cameras and the view direction of the cameras.

3.1.1 Calculations

The camera centres w.r.t world coordinates can be calculated using following formula:

$$-\mathbf{R}^T\mathbf{t}$$

- where R is the rotation matrix, and t is the translation matrix calculated by reading the calibration information.

To plot the view arrows, a point along the z axis of the camera coordinate systems was plotted (as the tip of the arrow) in the world coordinates using the following formula:

$$\mathbf{x}\mathbf{R}^T - \mathbf{R}^T\mathbf{t}$$

- where x is the point along the z axis, R is the rotation matrix, and t is the translation matrix calculated by reading the calibration information.

To plot the world origin and axes on the images form the cameras, we need to add the intrinsic parameters of the cameras to our calculations. The world coordinate origin and axes can be converted to image coordinates using the following:

$$x_3\tilde{\mathbf{x}} \quad = \quad \mathbf{K}(\mathbf{R}\tilde{\mathbf{p}} + \mathbf{t})$$

- where $\tilde{x}$ and $\tilde{p}$ are the homogeneous image coordinates and world coordinates respectively. K is the intrinsic parameter matrix read form the calibration information.

3.1.2 Pseudo-code

The following pseudo-code illustrates how these formulas were used to achieve the desired results.

Algorithm 1 plotCameras()

for all camera in cameras **do**
  $center \leftarrow -\mathbf{R}^T\mathbf{t}$
  $z \leftarrow (0, 0, 200)$
  $view \leftarrow \mathbf{z}\mathbf{R}^T - \mathbf{R}^T\mathbf{t}$
  Plot center
  Plot a ray from center to view
end for

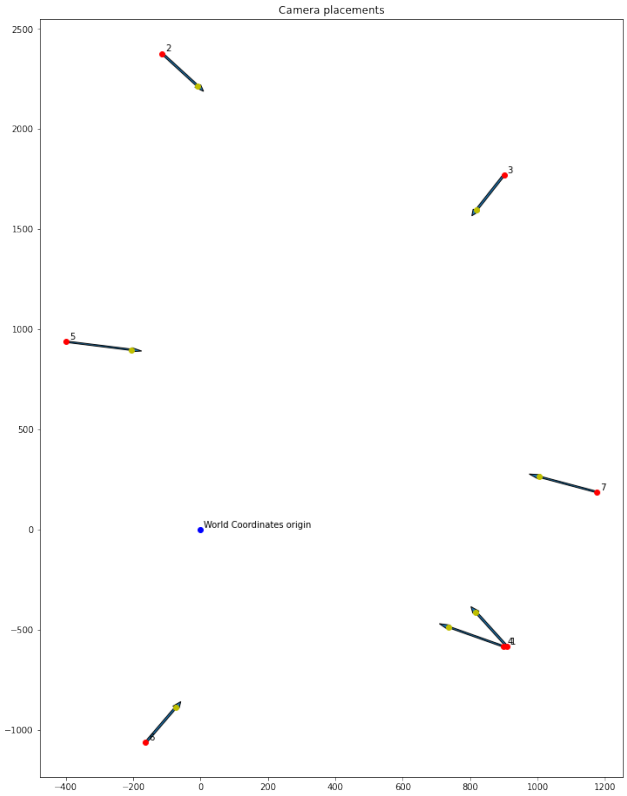


Figure 4: Camera placement in world coordinates

Algorithm 2 plotCoordinateAxes()

for all camera in cameras **do**
  $image \leftarrow frame_{15}$
  $origin \leftarrow world\_to\_image(center)$
  $x \leftarrow world\_to\_image((100, 0, 0))$
  $y \leftarrow world\_to\_image((0, 100, 0))$
  $z \leftarrow world\_to\_image((0, 0, 100))$
  Plot a ray from origin to x on image
  Plot a ray from origin to y on image
  Plot a ray from origin to z on image
end for

The results from plotting the world coordinate axes on the images gives a deeper understanding of relative orientation and placement of each of the cameras.

3.1.3 Results

The following figure shows the result form the algorithm implemented. All cameras are generally pointing towards a common view.

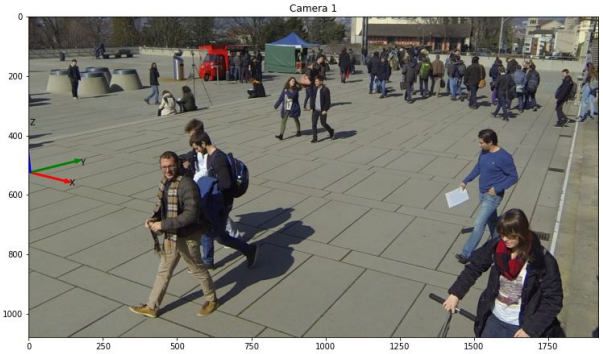


Figure 5: View from camera 1

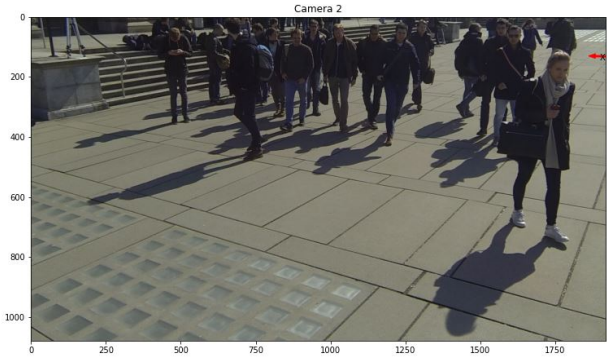


Figure 6: View from camera 2

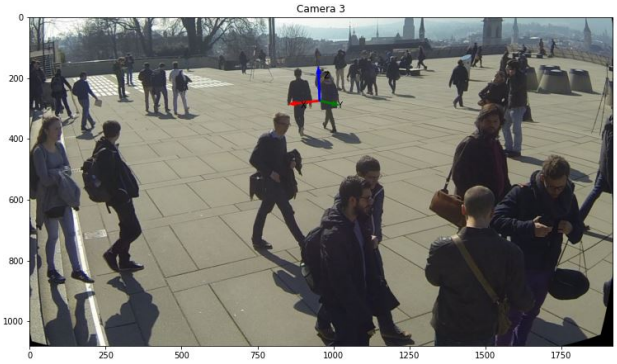


Figure 7: View from camera 3

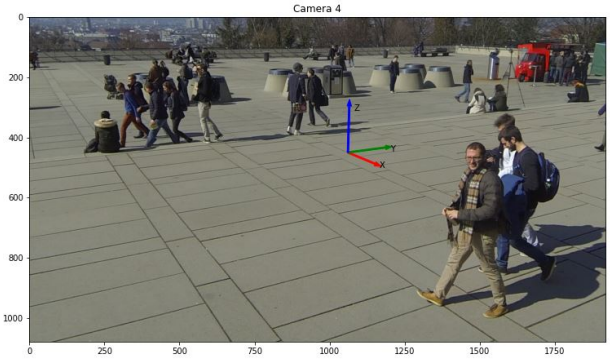


Figure 8: View from camera 4

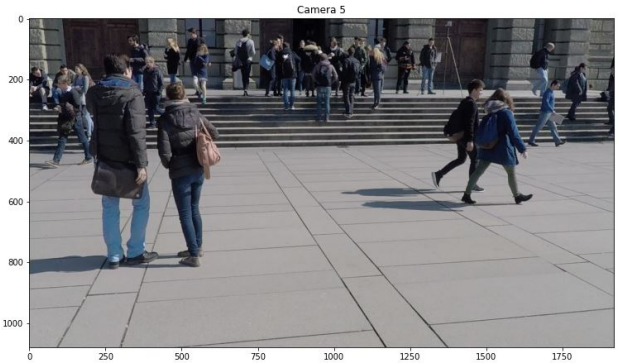


Figure 9: View from camera 5

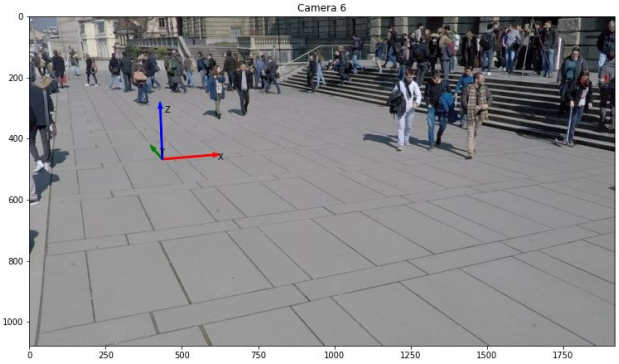


Figure 10: View form camera 6

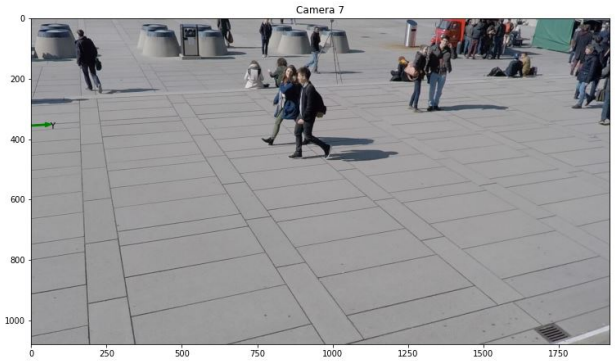


Figure 11: View from camera 7

3.2. Corresponding Epipolar lines of point

The function implemented here takes in a point in any of the seven cameras and draws the corresponding epipolar lines on images taken from the other six cameras. To approach this problem we will look into stereo setup of cameras, as for each cameras we have to find the

relation/transformation between the coordinate system of the given cameras and the camera in which the inputted points belong to.

3.2.1 Calculations

In stereo, we have two calibrated cameras. To simplify the mathematics, we will consider the world coordinate system to be one of the cameras, $\mathbf{c}_0$.The rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$ takes coordinates with respect to $\mathbf{c}_0$ and expresses it with respect to the second camera, $\mathbf{c}_1$.

The two equations that relate the image coordinates to the respective 3D coordinates for the two cameras are

$$\begin{aligned} x_3\tilde{\mathbf{x}}_\mathbf{0} &= \mathbf{K_0}\tilde{\mathbf{p}}_\mathbf{0} \\ x_3'\tilde{\mathbf{x}}_\mathbf{1} &= \mathbf{K_1}(\mathbf{R}\tilde{\mathbf{p}}_\mathbf{0} + \mathbf{t}) \end{aligned}$$

The above two equations can be combined into one equation by using the expression for $\tilde{\mathbf{p}}_\mathbf{0} = x_3\mathbf{K_0}^{-1}\tilde{\mathbf{x}}_\mathbf{0}$ from the first equation.

$$\begin{aligned} x_3'\tilde{\mathbf{x}}_\mathbf{1} &= \mathbf{K_1}(\mathbf{R}(x_3\mathbf{K_0}^{-1}\tilde{\mathbf{x}}_\mathbf{0}) + \mathbf{t}) \\ x_3'\mathbf{K_1}^{-1}\tilde{\mathbf{x}}_\mathbf{1} &= x_3\mathbf{R}\mathbf{K_0}^{-1}\tilde{\mathbf{x}}_\mathbf{0} + \mathbf{t} \end{aligned}$$

To reduce notational clutter, let $\hat{\mathbf{x}}_\mathbf{0} = \mathbf{K_0}^{-1}\tilde{\mathbf{x}}_\mathbf{0}$ and $\hat{\mathbf{x}}_\mathbf{1} = \mathbf{K_1}^{-1}\tilde{\mathbf{x}}_\mathbf{1}$. Both $\hat{\mathbf{x}}_\mathbf{0}$ and $\tilde{\mathbf{x}}_\mathbf{0}$ refer to the 2D location of the same pixel in an image. One is with respect to the optical center of the image, and the other is the array index, scaled by the focal length. They are related by the intrinsic parameter matrix $\mathbf{K_0}$

Using these intrinsic parameter normalized indices, we have the following relationship between the coordinates of the two images.

$$x_3'\hat{\mathbf{x}}_\mathbf{1} = x_3\mathbf{R}\hat{\mathbf{x}}_\mathbf{0} + \mathbf{t}$$

Take a cross product with vector $\mathbf{t}$ of both sides. Note $\mathbf{t} \times \mathbf{t} = 0$.

$$x_3'\mathbf{t} \times \hat{\mathbf{x}}_\mathbf{1} = x_3\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_\mathbf{0}$$

Take dot product with $\hat{\mathbf{x}}_\mathbf{1}$. The left-hand side will be zero. Note the scaling constants x_3, and x_3' are not part of this final relationship.

$$\hat{\mathbf{x}}_\mathbf{1}^T(\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_\mathbf{0}) = 0$$

Any cross product can be written as matrix multiplication with a skew-symmetric matrix.

$$\begin{aligned} \hat{\mathbf{t}} \times \mathbf{a} &= \begin{vmatrix} i & j & k \\ \hat{t}_x & \hat{t}_y & \hat{t}_z \\ a_0 & a_1 & a_2 \end{vmatrix} \\ \left[\hat{\mathbf{t}}\right]_\times \mathbf{a} &= \begin{bmatrix} 0 & -\hat{t}_z & \hat{t}_y \\ \hat{t}_z & 0 & -\hat{t}_x \\ -\hat{t}_y & \hat{t}_x & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \end{aligned}$$

Using this, we arrive at the following expression.

$$\hat{\mathbf{x}}_\mathbf{1}^T\left[\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_0\right] = 0$$

$$\hat{\mathbf{x}}_\mathbf{1}^T \begin{bmatrix} 0 & -\hat{t}_z & \hat{t}_y \\ \hat{t}_z & 0 & -\hat{t}_x \\ -\hat{t}_y & \hat{t}_x & 0 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} \hat{\mathbf{x}}_0 = 0$$

The 3 by 3 matrix $\mathbf{E} = \mathbf{t} \times \mathbf{R}$ is called the **essential matrix**.

$$\mathbf{E} = \begin{bmatrix} 0 & -\hat{t}_z & \hat{t}_y \\ \hat{t}_z & 0 & -\hat{t}_x \\ -\hat{t}_y & \hat{t}_x & 0 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}$$

Using this we can express the constraint between the corresponding points, specified with respect to the center of the image (camera coordinates) in the two views as

$$\begin{aligned} \begin{bmatrix} \hat{x}_1 & \hat{y}_1 & 1 \end{bmatrix} \begin{bmatrix} e_{00} & e_{01} & e_{02} \\ e_{10} & e_{11} & e_{12} \\ e_{20} & e_{21} & e_{22} \end{bmatrix} \begin{bmatrix} \hat{x}_0 \\ \hat{y}_0 \\ 1 \end{bmatrix} &= 0 \\ \hat{\mathbf{x}}_1^T\mathbf{E}\hat{\mathbf{x}}_0 &= 0 \end{aligned}$$

The **essential matrix** relates the homogenous coordinates of the corresponding points in the two images that are expressed **with respect to the center of the image.**

We can have a similar relationship between the array-based pixel coordinates. We have to use the intrinsic parameters.

$\hat{\mathbf{x}}_j = \mathbf{K}_j^{-1}\tilde{\mathbf{x}}_j$.

Plugging this relationship into the essential matrix relationship, we have the following relationship.

$$\begin{aligned} \hat{\mathbf{x}}_1^T\mathbf{E}\hat{\mathbf{x}}_0 &= 0 \\ (\mathbf{K}_1^{-1}\tilde{\mathbf{x}}_1)^T\mathbf{E}(\mathbf{K}_0^{-1}\tilde{\mathbf{x}}_0) &= 0 \\ (\tilde{\mathbf{x}}_1^T\mathbf{K}_1^{-T})\mathbf{E}(\mathbf{K}_0^{-1}\tilde{\mathbf{x}}_0) &= 0 \\ \tilde{\mathbf{x}}_1^T(\mathbf{K}_1^{-T}\mathbf{E}\mathbf{K}_0^{-1})\tilde{\mathbf{x}}_0 &= 0 \\ \tilde{\mathbf{x}}_1^T\mathbf{F}\tilde{\mathbf{x}}_0 &= 0 \end{aligned}$$

Thus, the epipolar constraint can also be expressed with the pixel coordinates, $\tilde{\mathbf{x}}_j$, using the **fundamental matrix**. The relationship between the fundamental and essential matrices is:

$$\mathbf{F} = {\mathbf{K_1}}^{-T}\mathbf{E}{\mathbf{K_0}}^{-1}$$

We can use the equation we just derived to specify the corresponding epipolar lines in the two images: $\tilde{\mathbf{x}}_1^T\mathbf{F}\tilde{\mathbf{x}}_0 = 0$

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} f_{00} & f_{01} & f_{02} \\ f_{10} & f_{11} & f_{12} \\ f_{20} & f_{21} & f_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = 0$$

For a pixel location (a, b) in the image for camera $\mathbf{c}_0$, the corresponding epiolar line in $\mathbf{c}_1$ can be calculated as follows:

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} f_{00} & f_{01} & f_{02} \\ f_{10} & f_{11} & f_{12} \\ f_{20} & f_{21} & f_{22} \end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} = 0$$

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} af_{00} & bf_{01} & f_{02} \\ af_{10} & bf_{11} & f_{12} \\ af_{20} & bf_{21} & f_{22} \end{bmatrix} = 0$$

Thus the equation of epipolar line in c1 is :

$$x_1(af_{00} + bf_{01} + f_{02}) +$$

$$y_1(af_{10} + bf_{11} + f_{12}) +$$

$$(af_{20} + bf_{21} + f_{22}) + = 0$$

3.2.2 Pseudo-code

Using the above mentioned equations we can define functions which take in a point in the image taken from a camera and draw the epipolar lines in all the other ones. Using these formulas we will construct our next 3 algorithms.

Algorithm 3 plotEpipolarLines(cameraNum, pixelX, pixelY)

```
baseCam ← camera[cameraNum]
for all camera in cameras do
    image ← frame_15
    if camera = baseCam then
        Plot a point at (pixelX,pixelY) on image
    else
        R_c, T_c, K_c ← baseCam params
        R, T, K ← camera params
        F ← compute_F_matrix(R_c, T_c, K_c, R, T, K)

        Plot epipolar_lines(F, pixelX, pixelY)
    end if
end for
```

3.2.3 Results

The following figures shows the resulting images from the algorithm. Each image shows epipolar lines corresponding to pixel [967,346] in image taken by camera 3. Also epipolar lines for pixel values in the range of +- 1 pixels is shown in the figures.

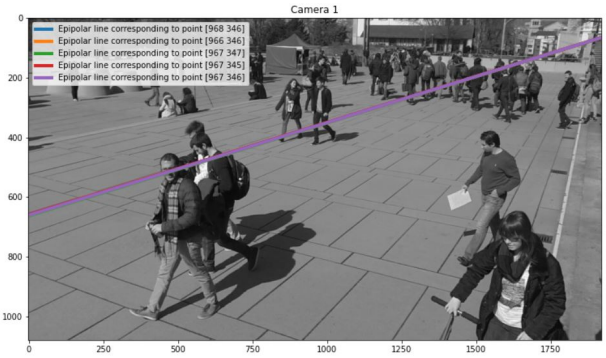


Figure 12: Epipolar lines corresponding to camera 3 image's [967,346] in camera 1 view

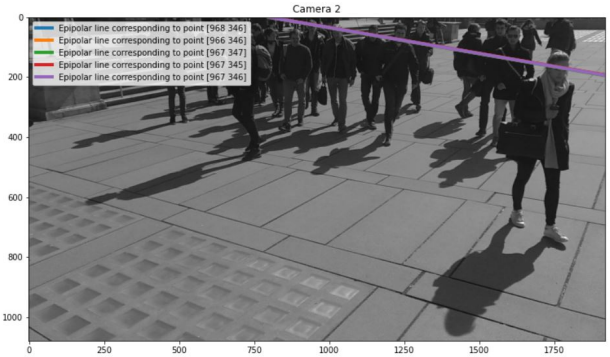


Figure 13: Epipolar lines corresponding to camera 3 image's [967,346] in camera 2 view

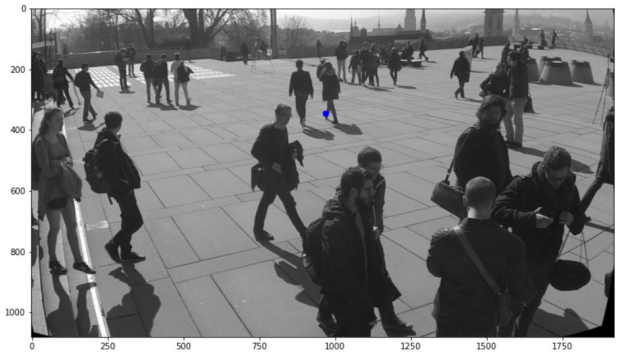


Figure 14: Camera 3 view showing [967,346]

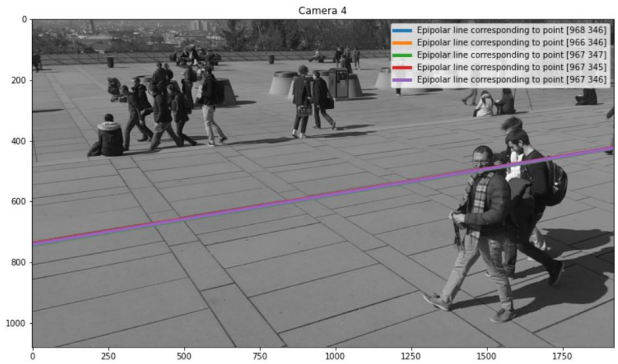


Figure 15: Epipolar lines corresponding to camera 3 image's [967,346] in camera 4 view

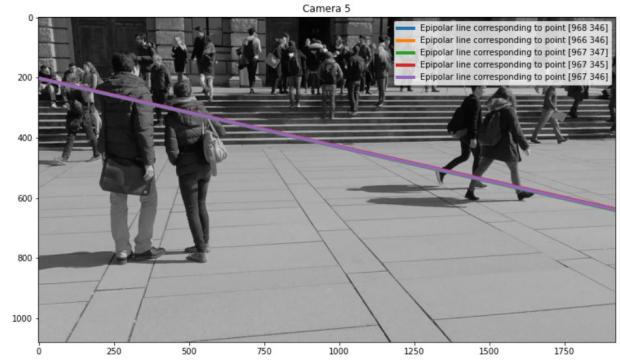


Figure 16: Epipolar lines corresponding to camera 3 image's [967,346] in camera 5 view

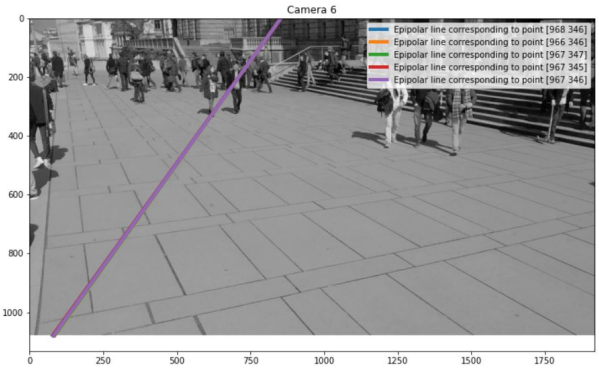


Figure 17: Epipolar lines corresponding to camera 3 image's [967,346] in camera 6 view

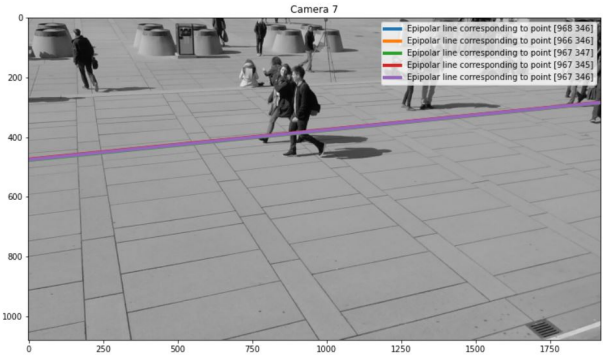


Figure 18: Epipolar lines corresponding to camera 3 image's [967,346] in camera 7 view

3.2.4 Implications of error in point locations

To understand the implications of error in measuring pixel location, refer to the following figure. The purple epipolar line corresponds to the actual point location given, while the green and red lines are drawn considering +-1 pixel neighborhood. As apparent from the figure changes in the location of the point, shifts the epipolar lines. This can eventually throw off our dense depth estimation calculations. For example, in the figure the red line corresponds to an epipolar line which is just over the shoe tip. Thus when estimating depth using multiple cameras(at least 2), the point corresponding to the shoe tip can be a point on the red line which is not the show tip but the ground with a different depth compared to the shoe tip. Here since the error is not that large (just 1 pixel) the implications are minute, but still significant. We will discuss the results of depth estimation with this error in a later section, but the error in estimation in our case reached up to 1 cm in estimating height of a person.

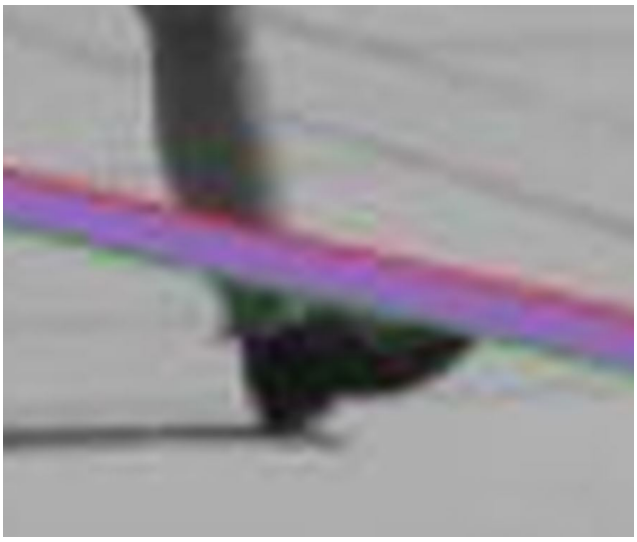


Figure 19: Implications of error in point locations

3.3. Corresponding Epipolar lines of corresponding pair of points

The function implemented here, takes in two point corresponding points (i.e. the same points in world coordinates) in images taken from different cameras, and draws the epipolar lines corresponding to them in the images taken from the rest of the cameras. The the calculation described in section 3.2 are also used here.

3.3.1 Pseudo-code

Algorithm 4 plotEpipolarLines(cameraNum1, pixelX1, pixelY1, cameraNum2, pixelX2, pixelY2)

$baseCam1 \leftarrow camera[cameraNum1]$
$baseCam1 \leftarrow camera[cameraNum2]$
for all camera in cameras **do**
  $image \leftarrow frame_{15}$
  if camera = baseCam1 **then**
    Plot a point at (pixelX1,pixelY1) on image
  else if camera = baseCam2 **then**
    Plot a point at (pixelX2,pixelY2) on image
  else
    R1_c, T1_c, K1_c ← baseCam1 params
    R2_c, T2_c, 2K_c ← baseCam2 params
    R, T, K ← camera params
    $F1 \leftarrow compute\_F\_matrix(R1\_c, T1\_c, K1\_c, R, T, K)$

    $F2 \leftarrow compute\_F\_matrix(R2\_c, T2\_c, K2\_c, R, T, K)$

    Plot epipolar_lines(F1, pixelX1, pixelY1)
    Plot epipolar_lines(F2, pixelX2, pixelY2)
  end if
end for

3.3.2 Results

The following figures shows the resulting images from the algorithm. The correspondign epipolar lines of the given pair of points intersects close to the corresponding point (show tip) in the camera view. Also, as seen in the camera two view, even though the shoe tip is not visible, the intersection accurately estimates the location of the show tip. This is due to the use of epipolar geometry, instead of visual matching. Some lines with +-1 pixel variations are also plotted, to understand the implications of having an error in point locations.

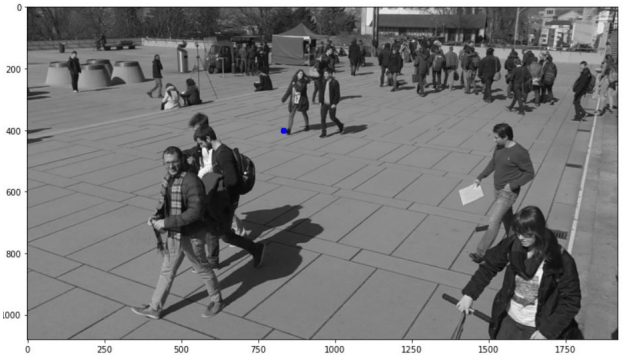


Figure 20: Camera 1 view showing [831,401]

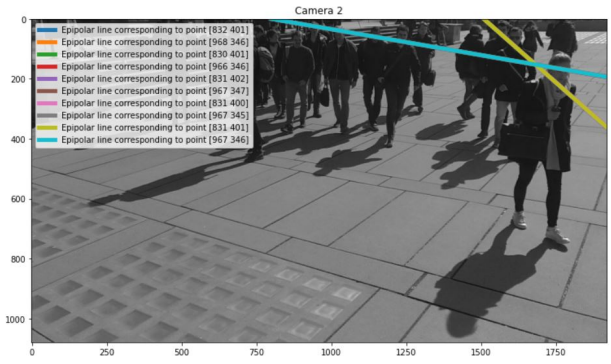


Figure 21: Epipolar lines corresponding to camera 1 image's [831,401] and camera 3 image's [967,346] in camera 2 view

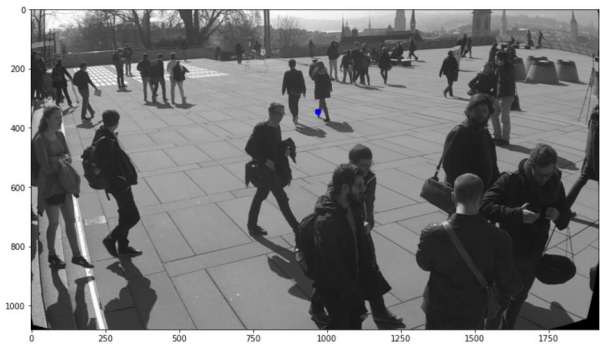


Figure 22: Camera 3 view showing [967,346]

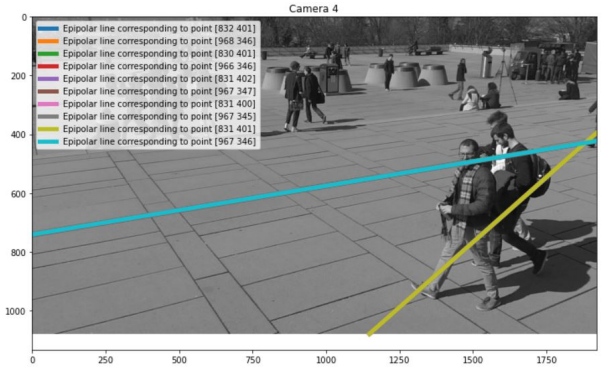


Figure 23: Epipolar lines corresponding to camera 1 image's [831,401] and camera 3 image's [967,346] in camera 4 view

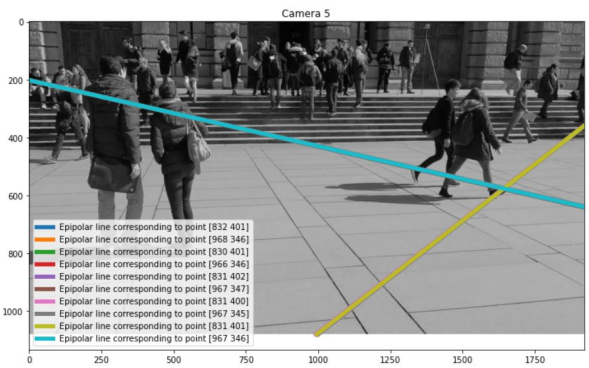


Figure 24: Epipolar lines corresponding to camera 1 image's [831,401] and camera 3 image's [967,346] in camera 5 view

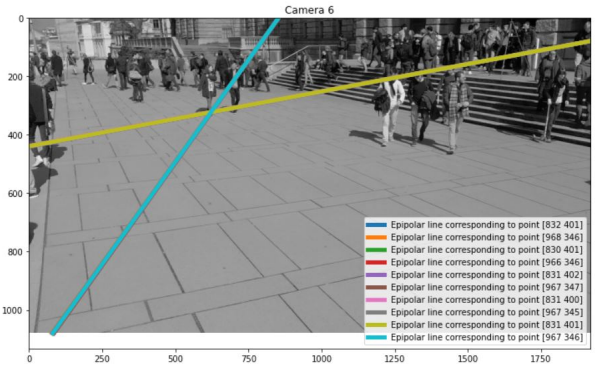


Figure 25: Epipolar lines corresponding to camera 1 image's [831,401] and camera 3 image's [967,346] in camera 6 view

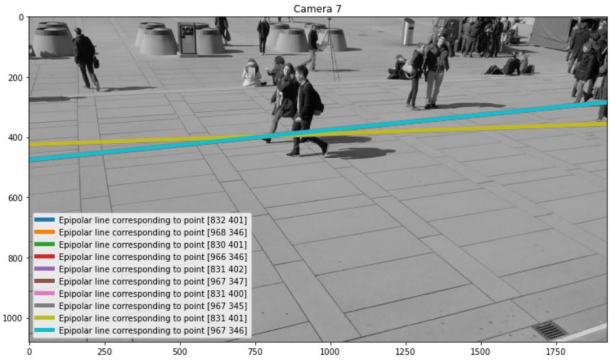
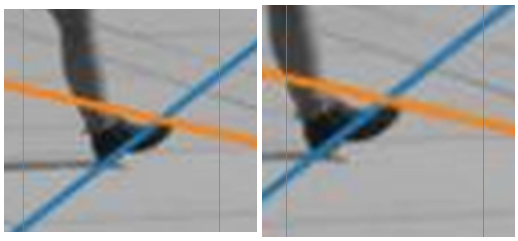


Figure 26: Epipolar lines corresponding to camera 1 image's [831,401] and camera 3 image's [967,346] in camera 7 view

3.3.3 Implications of error in point locations

The following figure compares two intersection points to elaborate the error in point location effects the process of location same object/person. Using corresponding pixel locations from two images, we can estimate the location of the corresponding object in another image by locating the intersection point between the two epipolar lines corresponding to the known pixel locations. As mentioned in the previous section, the +- 1 pixel error shifts the epipolar lines. Thus this shifting changes the point at which these two lines intersect. As shown in figure, the intersection point in (b) is higher and more towards the left side than the one in (a). Thus, this creates an uncertainty in our estimation of the location of the same object. Our intersections can lie in the neighborhood of the actual location. In the next section we will use multiple line and their intersection to locate an object which will help in increasing our confidence in the estimation.



(a) [831,401], [967,346] (b) [831,400], [967,345]

Figure 27: Implication of error in points locations

3.4. Corresponding Epipolar lines of corresponding set of points

This function is the generalized version of the two previous functions discussed in section 3.2 and 3.3. It uses the same calculations to draw epipolar lines. The functions takes in corresponding points from any number of cameras less than 7, and plots the corresponding epipolar lines in the images from the cameras not included in the input list.

3.4.1 Pseudo-code

Algorithm 5 plotEpipolarLines(cameraList, pixelList)

for all camera in cameraList **do**
  $baseCams \leftarrow baseCams \cup camera[cameraNum]$
end for
for all camera in cameras **do**
  $image \leftarrow frame_{15}$
  if camera in cameraList **then**
    Plot a point at pixelList[cameraIndex] on image
  else
    for all baseCam in cameraList **do**
      R_c, T_c, K_c ← baseCam params
      R, T, K ← camera params
      $F \leftarrow compute\_F\_matrix(R\_c, T\_c, K\_c, R, T, K)$

      Plot epipolar_lines(F, pixelList[cameraIndex])
    end for
  end if
end for

3.4.2 Results

The following figures shows the results of the algorithm. Theoretically the point at which the epipolar line intersect corresponds to the same point/object (shoe tip). But due the uncertainty in finding corresponding points from different images, there is no exact intersection, but more like small region formed by intersections between those lines.

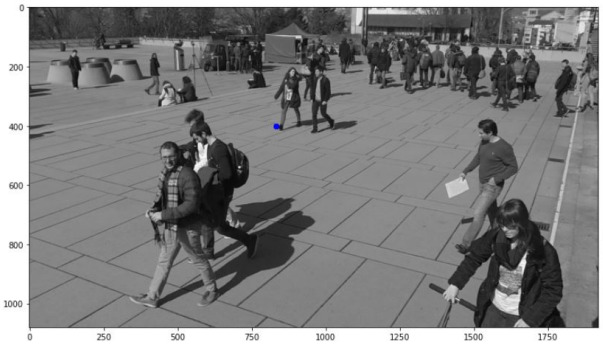


Figure 28: Camera 1 view showing [831,401]

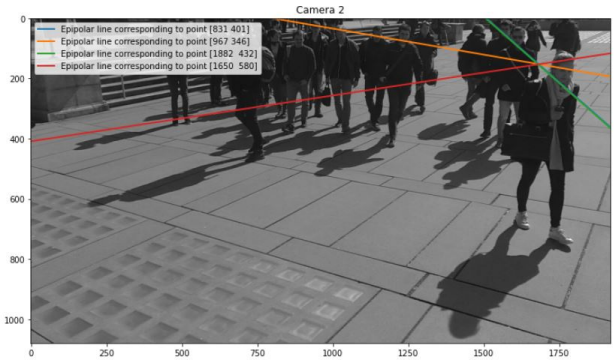


Figure 29: Epipolar lines corresponding to camera 1 image's [831,401], camera 3 image's [967,346], camera 4 image's [1882, 432], and camera 5 image's [1650,580] in camera 2 view

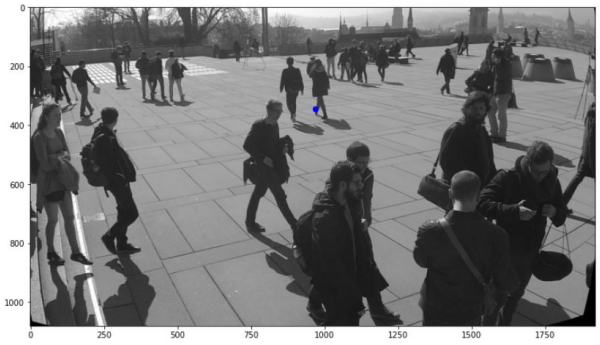


Figure 30: Camera 3 view showing [967,346]

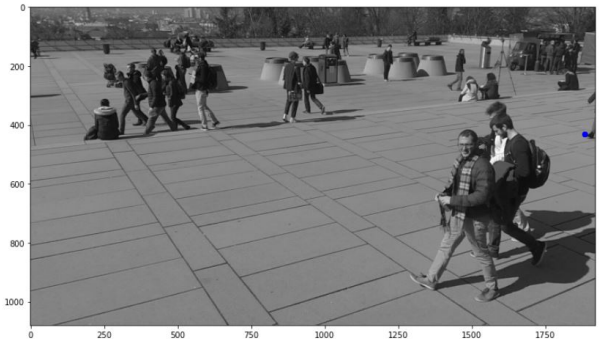


Figure 31: Camera 4 view showing [1882,432]

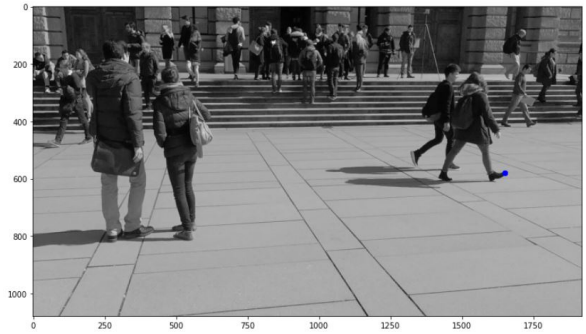


Figure 32: Camera 5 view showing [1650,581]

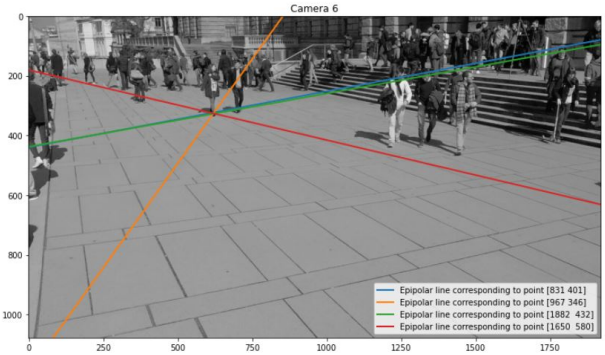


Figure 33: Epipolar lines corresponding to camera 1 image's [831,401], camera 3 image's [967,346], camera 4 image's [1882, 432], and camera 5 image's [1650,580] in camera 6 view

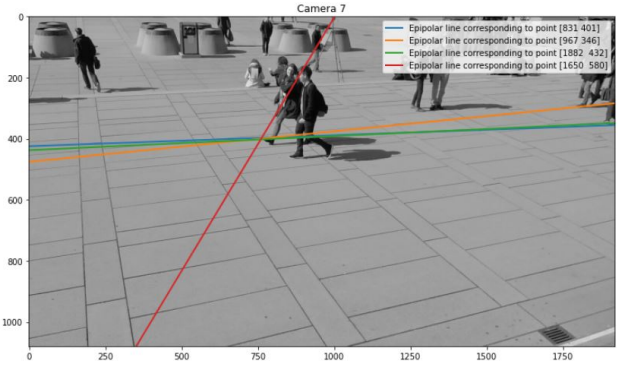
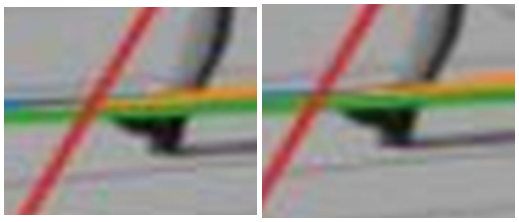


Figure 34: Epipolar lines corresponding to camera 1 image's [831,401], camera 3 image's [967,346], camera 4 image's [1882, 432], and camera 5 image's [1650,580] in camera 7 view

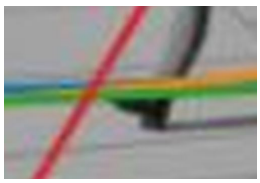
3.4.3 Implications of error in point locations

As seen from our previous discussions, error in pixel values shifts the epipolar lines. When considering the intersection of more than two epipolar lines, a small region of intersection is created where all the lines are closest to each other. The actual point (here shoe tip) lies in this region. Thus the +-1 pixel error can shift the lines and change the size of that area, increasing our uncertainty in the estimation. As compared to (a) in the figure, the region of intersection is slightly bigger in (d). Also the location at which we consider the intersection is also shifted to left ,and a bit higher in (d) compared to (a).

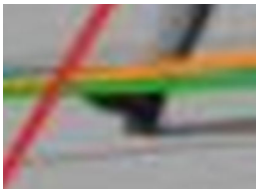


(a) [831,401], [967,346], [1882,432], [1650,580]

(b) [831,402], [967,347], [1881,432], [1651,580]



(c) [830,401], [966,346], [1883,432], [1651,580]



(d) [831,400], [967,345], [1882,432], [1650,581]

Figure 35: Implication of error in points locations

3.5. Height Estimation using 3D estimation

This function takes in and input of top (top most pixel of the person) and the bottom (Bottom most pixel of the person or the ground) points from camera images in which the person is view-able and estimates the height of that person. The height estimation is done through estimating the 3D coordinates (in world coordinates) of the top and bottom of the person. The euclidean distance between those two points gives us the estimated height in cm. The following process is used to estimate the 3D coordinates:

3.5.1 Calculations

Let the image of the 3D point $\mathbf{p} = (X_p, Y_p, Z_p)$ be denoted by $\mathbf{x_j}$ in the j-th camera. The intrinsic and extrinsic parameters of the j-th camera are denoted by $\mathbf{K}_j$, $\mathbf{R}_j$, and $\mathbf{t}_j$,

$$
\begin{aligned}
\tilde{\mathbf{x}}_\mathbf{j} &= \mathbf{K}_j \begin{bmatrix} \mathbf{R}_j & \mathbf{t}_j \end{bmatrix} \tilde{\mathbf{p}} \\
\mathbf{K}_j^{-1}\tilde{\mathbf{x}}_\mathbf{j} &= \begin{bmatrix} \mathbf{R}_j & \mathbf{t}_j \end{bmatrix} \tilde{\mathbf{p}} \\
\mathbf{R}_j^T\mathbf{K}_j^{-1}\tilde{\mathbf{x}}_\mathbf{j} &= \begin{bmatrix} \mathbf{I} & \mathbf{R}_j^T\mathbf{t}_j \end{bmatrix} \tilde{\tilde{\mathbf{p}}} \\
&= \begin{bmatrix} 1 & 0 & 0 & (\mathbf{R}_j^T\mathbf{t}_j)[0] \\ 0 & 1 & 0 & (\mathbf{R}_j^T\mathbf{t}_j)[1] \\ 0 & 0 & 1 & (\mathbf{R}_j^T\mathbf{t}_j)[2] \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ Z_p \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} X_p + (\mathbf{R}_j^T\mathbf{t}_j)[0] \\ Y_p + (\mathbf{R}_j^T\mathbf{t}_j)[1] \\ Z_p + (\mathbf{R}_j^T\mathbf{t}_j)[2] \end{bmatrix} \\
\mathbf{R}_j^T\mathbf{K}_j^{-1}\tilde{\mathbf{x}}_\mathbf{j} &= \mathbf{p} + \mathbf{R}_j^T\mathbf{t}_j
\end{aligned}
$$

Using the above, we can rewrite the world coordinates as

$$\mathbf{p} = \mathbf{R}_j^T\mathbf{K}_j^{-1}\tilde{\mathbf{x}}_\mathbf{j} + \mathbf{c}_j$$

Please keep in mind that $\mathbf{p}$ is specified with respect to the world coordinates.

It appears that we have a closed form solution for the 3D location of the image point just from one equation, but that is not so! In the above equation, we have to specify the homogeneous coordinates of the image pixel location, $\tilde{\mathbf{x}}_\mathbf{j} = \begin{bmatrix} w_j x_j \\ w_j y_j \\ w_j \end{bmatrix}^T$. We know (x_j, y_j), the pixel location, but we do not know w_j. The point could be anywhere along the vector that start from the image coordinate origin and go through the image pixel (x_j, y_j).

Re-writing the above equation using a compact notation.

$$\mathbf{p} = \mathbf{R}_j^T\mathbf{K}_j^{-1} \begin{bmatrix} w_j x_j \\ w_j y_j \\ w_j \end{bmatrix} + \mathbf{c}_j$$

$$\mathbf{p} = w_j \mathbf{R}_j^T\mathbf{K}_j^{-1} \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix} + \mathbf{c}_j$$

$$\mathbf{p} = d_j \hat{\mathbf{v}}_j \ + \mathbf{c}_j$$

where $\hat{\mathbf{v}}_j$ is the normalized unit vector, along $\mathbf{R}_j^T\mathbf{K}_j^{-1} \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix}$, representing the vector through the image pixel, with respect to the world coordinates. We denote it by

$$\hat{\mathbf{v}}_j = \mathcal{N}(\mathbf{R}_j^T\mathbf{K}_j^{-1}\mathbf{x}_j)$$

The residual vector $\mathbf{r}_j$ is the given by

$$\begin{aligned}\mathbf{r}_j &= (d_j\hat{\mathbf{v}}_j - \mathbf{c}_j) - \mathbf{p} \\ &= d_j\hat{\mathbf{v}}_j - (\mathbf{p} - \mathbf{c}_j)\end{aligned}$$

This is the vector connecting the lines through the image pixel and the vector to the 3D world coordinate, with respect to the **camera** coordinates, which is $(\mathbf{p} - \mathbf{c}_j)$. Use Figure 7.2 to illustrate this.

The total residual over N-cameras is given by

$$\sum_{j=1}^{N} ||\mathbf{r}_j||^2 = \sum_{j=1}^{N} (d_j\hat{\mathbf{v}}_j - (\mathbf{p} - \mathbf{c}_j))^T (d_j\hat{\mathbf{v}}_j - (\mathbf{p} - \mathbf{c}_j))$$

There are (N+3) unknowns: distances with respect to the N cameras d_j's, and the three world coordinates in $\mathbf{p}$. The unknowns are the depths from each of the cameras, d_j's, and the coordinate of the 3D point, $\mathbf{p}$. We will derive the minimization conditions in two steps.

1. First, we will take derivative of the total residual with respect to d_j and set it to zero.

$$\begin{aligned}\frac{\partial}{\partial d_j}(d_j\hat{\mathbf{v}}_j + \mathbf{c}_j - \mathbf{p})^T(d_j\hat{\mathbf{v}}_j + \mathbf{c}_j - \mathbf{p}) &= 0 \\ 2(\hat{\mathbf{v}}_j)^T(d_j\hat{\mathbf{v}}_j + \mathbf{c}_j - \mathbf{p}) &= 0 \\ d_j + \hat{\mathbf{v}}_j^T(\mathbf{c}_j - \mathbf{p}) &= 0 \\ d_j &= \hat{\mathbf{v}}_j^T(\mathbf{p} - \mathbf{c}_j)\end{aligned}$$

Plugging this back into the residual equation, we have a new expression of the residual that does not involve d_j.

$$\sum_{j=1}^{N} ||\mathbf{r}_j||^2 = \sum_{j=1}^{N} ||(\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T(\mathbf{p} - \mathbf{c}_j) - (\mathbf{p} - \mathbf{c}_j))||^2$$

$$= \sum_{j=1}^{N} ((\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j))^T((\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j))$$

2. Second, we take the derivative of the above residual expression with respect to $\mathbf{p}$ and set it to zero.

$$\begin{aligned}\sum_{j=1}^{N}(\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)^T((\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j)) &= 0 \\ \sum_{j=1}^{N}(\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j) &= 0 \\ \sum_{j=1}^{N}(\mathbf{I} - 2\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T + \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j) &= 0 \\ \sum_{j=1}^{N}(\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j) &= 0 \\ \left(\sum_{j=1}^{N}(\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)\right)\mathbf{p} - \sum_{j=1}^{N}(\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)\mathbf{c}_j &= 0\end{aligned}$$

The least square estimate of the 3D location of the point is given by

$$\mathbf{p} = \left(\sum_{j=1}^{N}(\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)\right)^{-1}\sum_{j=1}^{N}(\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)\mathbf{c}_j$$

This equation is valid only when we have more than one camera. For one camera, we get a degenerate solution where $\mathbf{p} = \mathbf{c}_j$, i.e. the point is located at the camera origin!

3.5.2 Pseudo-code

Algorithm 6 estimateHeight(top, bottom)

$selected\_list \leftarrow listofcamerasinthethepersonisintheview$

for all camera in selected_list **do**
$K\_list \leftarrow K\_list \cup camera\_K$
$R\_list \leftarrow R\_list \cup camera\_R$
$T\_list \leftarrow T\_list \cup camera\_T$
end for

$top\_3D \leftarrow estimate\_3D\_pt(top[:, selected\_list, K\_list, R\_list, T\_list)$

$bottom\_3D \leftarrow estimate\_3D\_pt(bottom[:, selected\_list, K\_list, R\_list, T\_list)$

$height \leftarrow euclidean(top\_3D, bottom\_3D)$
return height

3.5.3 Results

The following figures shows the cameras and the points used for estimating the height of the person.

Figure 36: Camera 1 view showing the person and the points selected

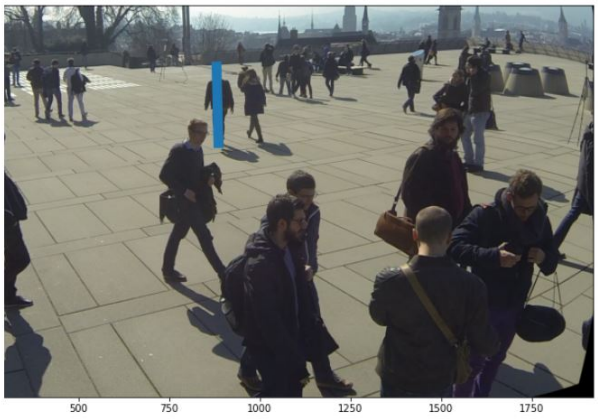


Figure 37: Camera 3 view showing the person and the points selected

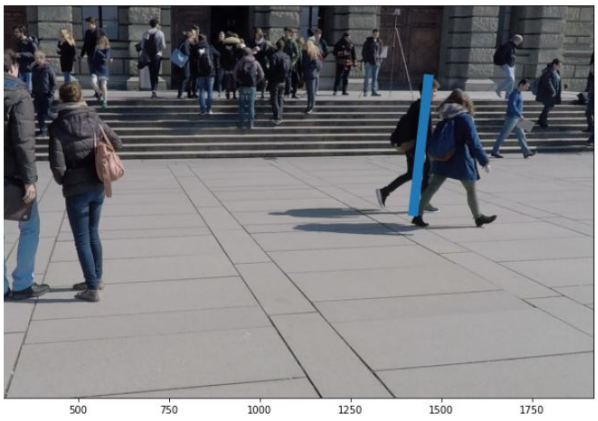


Figure 38: Camera 5 view showing the person and the points selected

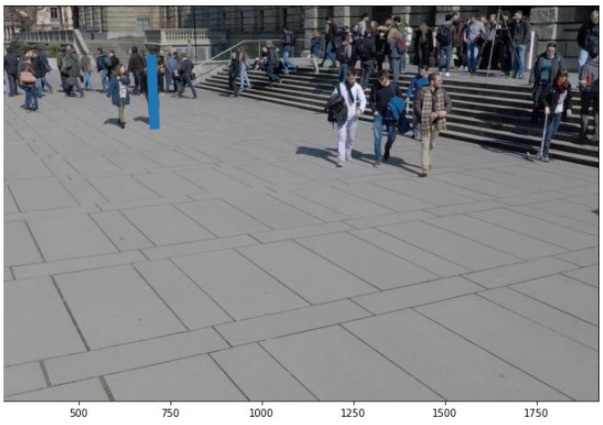


Figure 39: Camera 6 view showing the person and the points selected

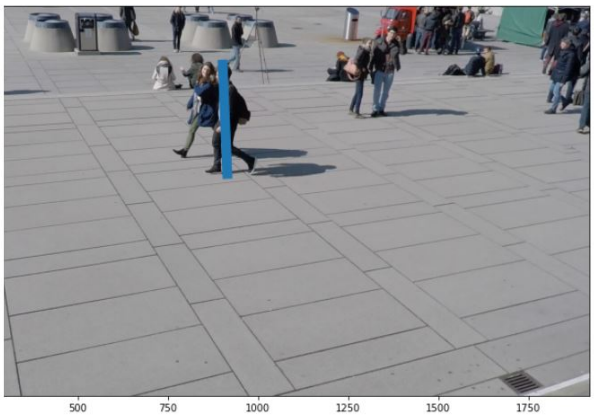


Figure 40: Camera 7 view showing the person and the points selected

3.5.4 Implications of error in point locations

Continuing our discussion of implications of error in point locations, considering pixel errors here in the 3D estimation functions shows the effect of implications discussed in the previous sections. Looking at the variations in height estimate proves the observations made in the previous sections. Errors in measuring the corresponding points moves the top and the bottom points which creates taller and shorter height estimates. The following estimates of the height (cm) were produced by the algorithm considering +- 1 pixel variations:

$[166.4365661844584, 167.16419348088917, 167.89223273173616, 165.75055364678659, 166.47778349708892, 167.20543034300263, 165.06482970449144, 165.79165684707175, 166.51890611018877]$

Here:
$\mu_{height} = 166.478cm$
$std_{height} = 0.816cm$
$max_{height} = 167.892cm$
$min_{height} = 165.065cm$

4. Conclusions: factors affecting 3D estimation

So one the factors which affect our height estimation is the pixel location accuracy. Since in our implementation the pixel location from top and bottom points were manually annotated, there can be inaccuracies in locating them correctly. Through the sections we discussed that this creates an uncertainty in our estimation. One way to mitigate this is to take the mean of heights estimated using multiple pair of top and bottom locations (As shown in the previous section). Another factor that can affect the estimation is the placement and orientations of the cameras used to estimate the height. This factor affects our triangulation calculations. Cameras placed very close to each other results into bad estimates. Thus to mitigate this issue, the camera have to setup all around (spread out) the person in way that they create good overlap in camera views. The person's posture can also affect the height estimation. We used WILDTRACK data set in our experiment, which provides still frames of moving people. Thus posture and body angle variations can create inaccurate estimates. To mitigate this, it is essential to use the 3D estimation process on multiple frames taken at different instances, and calculate a mean of those values.

5. References

The report and the code uses elements from Dr. Sudeep Sarkar's (Department of Computer Science and Engineering, University of South Florida) Colab Notebooks, from the class curriculum of CAP 4410 Computer Vision, Fall 2021:

https://github.com/SudeepSarkar/Undergraduate-Computer-Vision/blob/main/CAP_4410_Lecture_21_2D_to_3D_camera_calibration_and_pose_estimation.ipynb

https://github.com/SudeepSarkar/Undergraduate-Computer-Vision/blob/main/CAP_4410_Lecture_23_Stereo.ipynb

https://github.com/SudeepSarkar/Undergraduate-Computer-Vision/blob/main/CAP_4410_Lecture_25_3D_by_triangulation.ipynb

Dataset Used: WILDTRACK Dataset

"WILDTRACK: A Multicamera HD Dataset for Dense Unscripted Pedestrian Detection," T. Chavdarova, P. Baqué, S. Bouquet, A. Maksai, C. Jose, T.Bagautdinov, L. Lettry, Pascal Fua, Luc Van Gool, François Fleuret;Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 5030-5039, https://www.epfl.ch/labs/cvlab/data/data-wildtrack/