A. SFM Algorithm

Data: Image Matches, K
Result: X, C, R
for *all possible pair of images* **do**
  `// Reject outlier correspondences`
  [x1, x2] = GetInliersRANSAC(x1, x2);
end
`// For first two images`
F = EstimateFundamentalMatrix(x1, x2);
E = EssentialMatrixFromFundamentalMatrix(F, K);
[Cset, Rset] = ExtractCameraPose(E);
`// Perform linear triangulation`
for *i = 1:4* **do**
  Xseti = LinearTriangulation(K, zeros(3,1), eye(3), Cseti, Rseti, x1, x2);
end
`// Check cheirality condition`
[C R] = DisambiguateCameraPose(Cset, Rset, Xset);
`// Perform Non-linear triangulation`
X = NonlinearTriangulation(K, zeros(3,1), eye(3), C, R, x1, x2, X0));
Cset = C, Rset = R;
`// Register camera and add 3D points for the rest of images`
for *i=3:I* **do**
  `// Register the` i^{th} `image using PnP.`
  [Cnew Rnew] = PnPRANSAC(X, x, K);
  [Cnew Rnew] = NonlinearPnP(X, x, K, Cnew, Rnew);
  Cset = Cset∪Cnew, Rset = Rset∪Rnew;
  `// Add new 3D points.`
  Xnew = LinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2);
  Xnew = NonlinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2, X0);
  X = X∪Xnew;
  `// Build Visibility Matrix.`
  V = BuildVisibilityMatrix(traj);
  `// Perform Bundle Adjustment.`
  [Cset Rset X] = BundleAdjustment(Cset, Rset, X, K, traj, V);
end

Loop Closure

B. Depth Algorithm

Data: Image Matches, K
Result: X, C, R
for *all possible pair of images* **do**
    // Reject outlier correspondences
    [x1, x2] = GetInliersRANSAC(x1, x2);
end
// For first two images
F = EstimateFundamentalMatrix(x1, x2);

// Make epilines parallel

Rectified_images = rectification(image_matches, F, images)

H1, H2 = cv2.stereoRectifyUncalibrated(left_pts, right_pts, F)

Rectified_images = cv2.warpPerspective(image, H)

// Extract Depth Map

Disparity, depth_map = find_depthmap(warpedimages, F, Baseline, window_size)

Disparity_map = disparity((warpedimages, F, Baseline, window_size)

for *all windows along x* do:

Distance = Sum of Squared Distances or Normalized cross correlation

Disparity_map = min_x - x

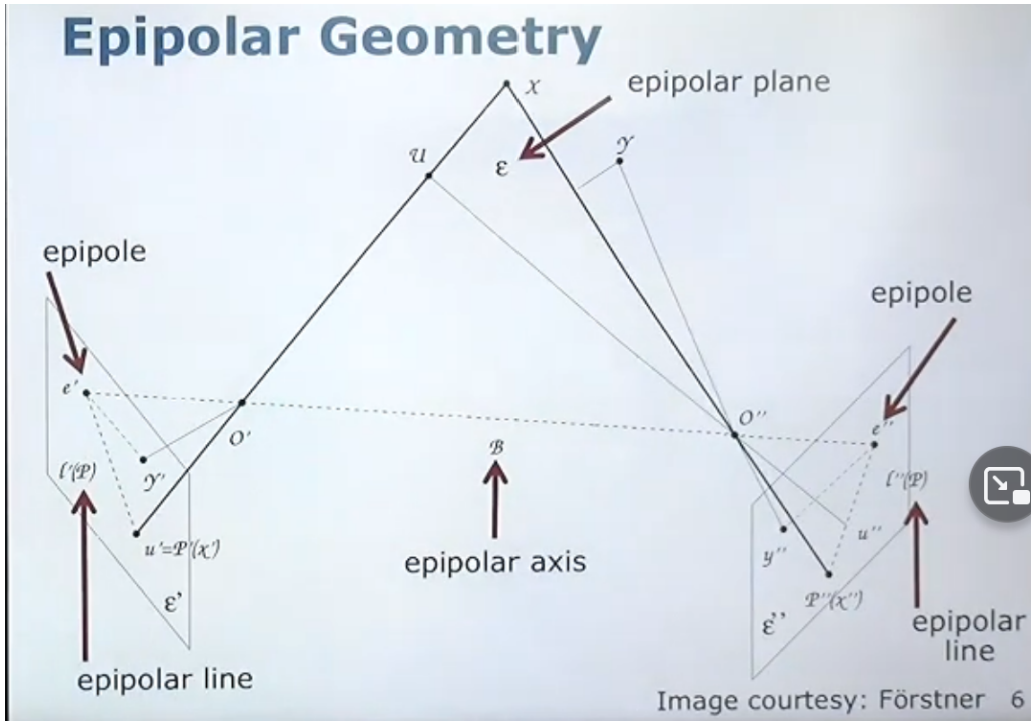
Depth_map = baseline*F/disparity_map

1. Keypoint detection

- ORB/SIFT
- Track the matches to determine where the camera has moved in each frame
- ORB:
    - Version1: Image Scales the imagesto extract better features
    - Version2: Fast9: Center Pixel and surrounding 16 pixels to form circle, 9 or mode pixel in the circle darker than the center
    - Version3: Non max suppression: retain the most dominant corner pixel
    - 4Rotation variance: characterize the corners.
    - 5BRIEF: Binary Robust Independent Elementary Features
    - 6Rotation invariace: descriptor matching, hamming distance b/w two brief descriptors

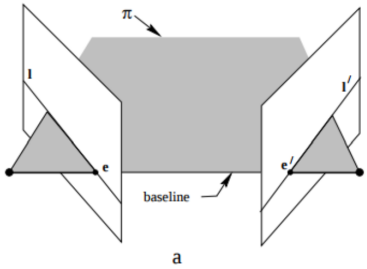
C. Epipolar Geometry

- reference Cyrill



- Epipolar geometry is used to describe geometric relations in image pairs.
- Enables efficient search and prediction of corresponding points
- Given a straight-line preserving mapping, the search space reduces to 2D to a 1D line
- Image 1: ε', Image 2: ε''
- **Projection centers**: O' and O''
- Real world point: x
- Mapping of x in image 1: $u' = P'x'$
- Mapping of x in image 2: $u'' = P''x''$
- Point y mapped to a different location through same method
- B = $(O'O'')$: **Epipolar axis**: line connecting the two projection centers.
- $(O'\ O''$ x) form an **epipolar plane** ε, different points in space will lead to different epipolar planes. (coplanrity constraint)
- $e' = (O'')' = P'O''$ and $e'' = (O')'' = P''O'$: **Epipoles**, projection of the other camera's projection center into the image. They are the intersection of the epipolar axis and the corresponding images.
- e' is the projection of O'' on image 1 and vice versa

- $l'(x)$ = $(O''x)'$ and $l''(x)$= $(O'x)''$ **epipolar line**: intersection of the epipolar plane with the image plane. This will be used for getting correspondences.



a

  The lines are the images of the rays $(O''x)$ and $(O'x)$ in the other image respectively.
- The **epipolar line** reduces the corresponding point search to a 1D search just along the epipolar line.
- No matter where the 3D point is in space, the projection centers will always be part of the epipolar plane, for eg, plane made from point y also has the two centers. Just by rotating the plane along the epipolar axis from x to reach point y
- Image points lie on the epipolar lines such that $x'^{T}l' = 0$ (**when a point lies on a line**) constraint.
- Coplanarity constrained: vector defined through x' and the vector from x'' must form a plane(epipolar plane) $x'^{T}Fx'' = 0$, Fx'' = l' the **epipolar line** in image 1 can be derived using fundamental matrix and the corresponding point in image 2 and vice versa $F^{T}x''$ = l'
- **All epipolar lines pass through the epipoles. For all epipolar lines,** $e'^{T}l'$ **= 0**
- $e'^{T}Fx'' = 0$, Hence epipole is the null space of F^{T}. e' and e'' are **eigenvectors** of F
- $e'^{T}F = 0$ null(F^{T}) = e' ,
- $Fe'' = 0$ null(F) =e'' these correspond to an **eigenvalues** of **zero**
- Hence since F has a null space apart from a zero vector, it is rank deficient.

D. Fundamental Matrix

- Projects a line(epipolar line) from a point.
- Singular matrix, Rank 2, because it maps a point as a line. 7 DoF
- 7 Dof: 5 degrees of freedom for the epiplane and 1 each to specify the two epipoles on
- the epiplane.
- **Coplanarity constraint:** x2.T*F*x1=0, all the lines will pass through the epipole
- The most convenient way to do this is to correct the matrix F found by the SVD solution from A. Matrix F is replaced by the matrix F' that minimizes the Frobenius norm ||F − F'|| subject to the condition det F' = 0.
- The F matrix is calculated as:
  1. Normalize the 8 correspondences selected

2. By using SVD, extract the fundamental matrix from the last vector of the V.T matrix. The A matrix format for the fundamental matrix is:

$$\begin{bmatrix} u_l^{(1)}u_r^{(1)} & u_l^{(1)}v_r^{(1)} & u_l^{(1)} & v_l^{(1)}u_r^{(1)} & v_l^{(1)}v_r^{(1)} & v_l^{(1)} & u_r^{(1)} & v_r^{(1)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_l^{(i)}u_r^{(i)} & u_l^{(i)}v_r^{(i)} & u_l^{(i)} & v_l^{(i)}u_r^{(i)} & v_l^{(i)}v_r^{(i)} & v_l^{(i)} & u_l^{(i)} & u_r^{(i)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_l^{(m)}u_r^{(m)} & u_l^{(m)}v_r^{(m)} & u_l^{(m)} & v_l^{(m)}u_r^{(m)} & v_l^{(m)}v_r^{(m)} & v_l^{(m)} & u_l^{(m)} & u_r^{(m)} & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

3. The F matrix extracted from this is of rank 3 but the fundamental matrix is always of rank 2, hence it needs to be put through a constraint where, the last element of the S matrix in SVD(F)= U, S, V.T, is made to be 1
S = diagonal[s11, s22, 1] and generate a new F by combining the 3 matrices again.

Objective

Given $n \geq 8$ image point correspondences $\{\mathbf{x}_i \leftrightarrow \mathbf{x}'_i\}$, determine the fundamental matrix F such that $\mathbf{x}'^{\mathsf{T}}_i \mathrm{F} \mathbf{x}_i = 0$.

Algorithm

(i) **Normalization:** Transform the image coordinates according to $\hat{\mathbf{x}}_i = \mathrm{T}\mathbf{x}_i$ and $\hat{\mathbf{x}}'_i = \mathrm{T}'\mathbf{x}'_i$, where T and T′ are normalizing transformations consisting of a translation and scaling.

(ii) Find the fundamental matrix $\hat{\mathrm{F}}'$ corresponding to the matches $\hat{\mathbf{x}}_i \leftrightarrow \hat{\mathbf{x}}'_i$ by

  (a) **Linear solution:** Determine $\hat{\mathrm{F}}$ from the singular vector corresponding to the smallest singular value of $\hat{\mathrm{A}}$, where $\hat{\mathrm{A}}$ is composed from the matches $\hat{\mathbf{x}}_i \leftrightarrow \hat{\mathbf{x}}'_i$ as defined in (11.3).

  (b) **Constraint enforcement:** Replace $\hat{\mathrm{F}}$ by $\hat{\mathrm{F}}'$ such that $\det \hat{\mathrm{F}}' = 0$ using the SVD (see section 11.1.1).

(iii) **Denormalization:** Set $\mathrm{F} = \mathrm{T}'^{\mathsf{T}} \hat{\mathrm{F}}' \mathrm{T}$. Matrix F is the fundamental matrix corresponding to the original data $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$.

E. Essential Matrix

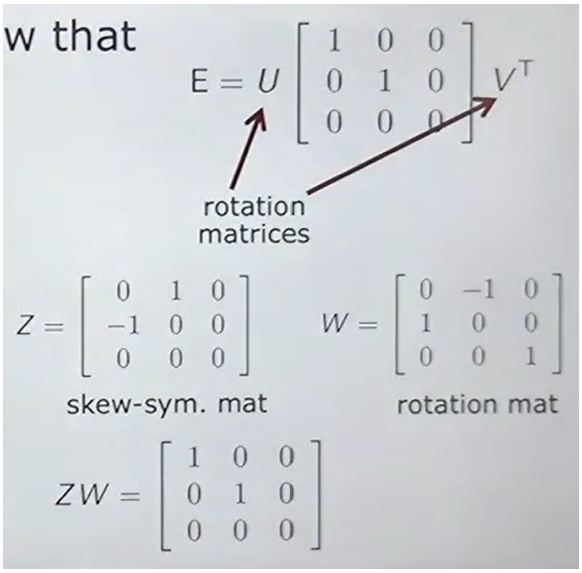
- 5 DoF: 3 rot, 3 trans but due to scale ambiguity 2trans
  An essential matrix can be extracted from a fundamental matrix given the camera intrinsic parameter, K.

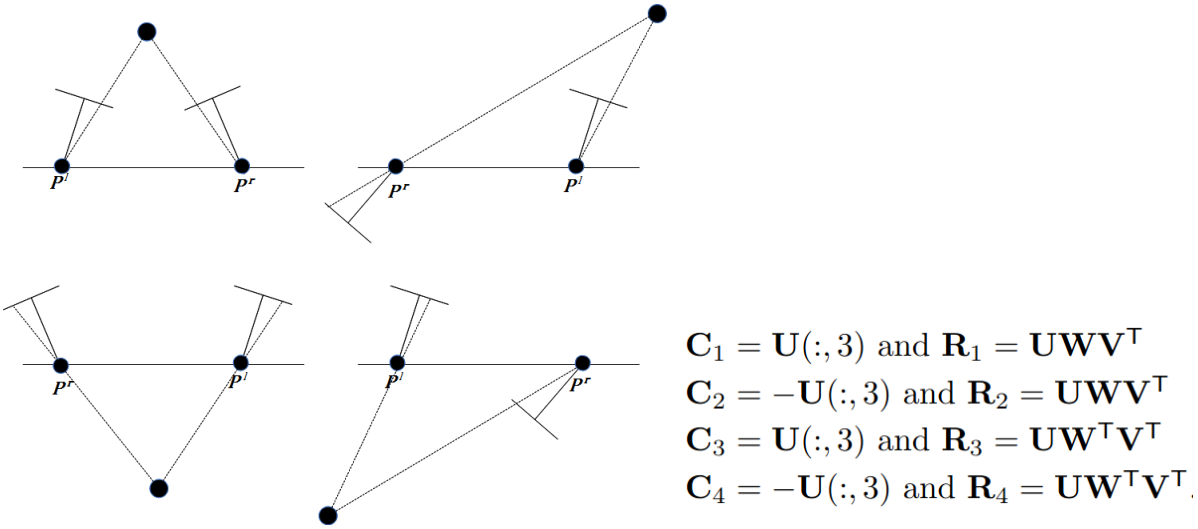
$$\mathbf{E} = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

- $E = k_2^T F k_1$
- **Estimate Camera Pose from Essential Matrix:**
  E = [t]×R

- Recreate the Essential matrix with these constraints

$$\mathbf{E} = UDV^T \text{ and } W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ and D = [1,1,0]

- W is orthogonal(Rotational matrix)
- 

- The camera matrices may be retrieved from the essential matrix up to scale and a four-fold ambiguity. Hence 4 different positions of pixels can be generated using 4 sets of rotation and translation configurations.
- 

$$\mathbf{C}_1 = \mathbf{U}(:,3) \text{ and } \mathbf{R}_1 = \mathbf{U}\mathbf{W}\mathbf{V}^\mathsf{T}$$
$$\mathbf{C}_2 = -\mathbf{U}(:,3) \text{ and } \mathbf{R}_2 = \mathbf{U}\mathbf{W}\mathbf{V}^\mathsf{T}$$
$$\mathbf{C}_3 = \mathbf{U}(:,3) \text{ and } \mathbf{R}_3 = \mathbf{U}\mathbf{W}^\mathsf{T}\mathbf{V}^\mathsf{T}$$
$$\mathbf{C}_4 = -\mathbf{U}(:,3) \text{ and } \mathbf{R}_4 = \mathbf{U}\mathbf{W}^\mathsf{T}\mathbf{V}^\mathsf{T}$$

This creates 4 sets (R1, C1), (R2, C2), (R3, C3), (R4, C4)

- Out of the 4 combinations, only one is the correct combination that projects the image in front of the camera and that is the set we should select.

- To check which combination generates the 3D points on the front side of the camera, we will make use of the epipolar plane. I triangulated 3D points using both the camera's poses. And checked which set of poses gives out the highest number of 3D points that lie in the positive half of the z plane.
    - Calculate Projection matrices for both the cameras, For the left camera, we assume, P = [I | 0] For the right camera, P = KR[I3×3−C]
    - Obtain 3D points for each pose set using the inbuild OpenCV function triangulatePoints().
    - For the set of 3D points in which most points satisfy the below condition, called the cheirality condition, their corresponding pose set will be chosen.
      R3[3d_pt - C] > 0 and z > 0

Problems:

- The number matches for further computation was very low. So had to force sift to churn out atleast 2000 features to choose from.
- Normalization: https://www.cc.gatech.edu/classes/AY2016/cs4476_fall/results/proj3/html/sdai30/index.html
- To check if F matrix was correct, checked if the determinant of all 4 R matrices is 1

F. Triangulation

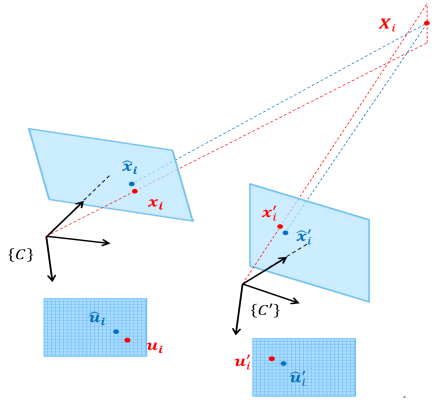
https://www.uio.no/studier/emner/matnat/its/nedlagte-emner/UNIK4690/v16/forelesninger/lecture_7_2-triangulation.pdf

Estimate a 3D point X for a noisy 2D correspondence under the assumption that camera matrices P and P' are known

$PX = u$

$P'X = u$

1. Geometric approach:
   In order to determine the 3D point X we back-project the two image points and determine their intersection. But due to noise, the two rays in 3D will "never" truly intersect, so we need to estimate a best solution to the problem



   One natural estimate for X is the midpoint on the shortest line between to two back-projected rays. This minimize the 3D error, but not the reprojection error. Cant be used with multiple cameras since it gives out higher error

2. Linear Triangulation approach:(algebraic error minimize)
   This algorithm uses the two equations for perspective projection to solve for the 3D point that are optimal in a least squares sense. P = KR[I | -C]

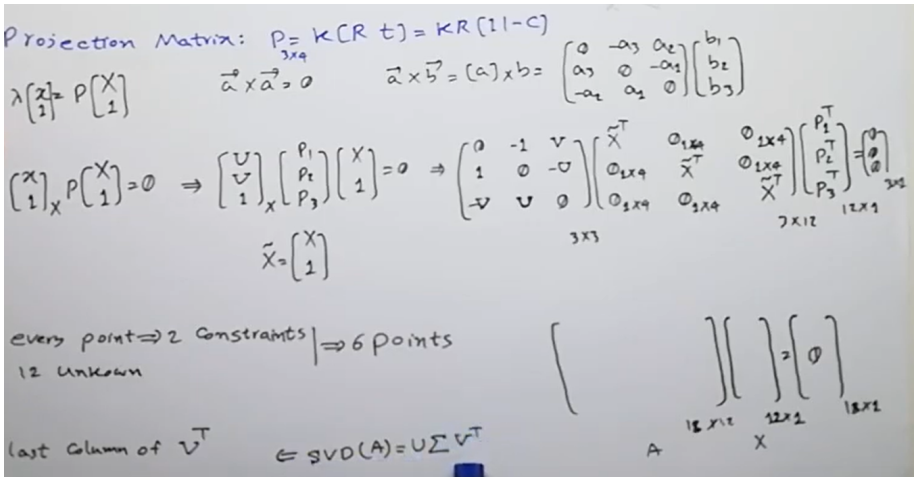
$$\tilde{\boldsymbol{u}}_i = P\tilde{\boldsymbol{X}}_i \qquad \tilde{\boldsymbol{u}}'_i = P'\tilde{\boldsymbol{X}}_i$$
$$\Downarrow \qquad \Downarrow$$
$$\tilde{\boldsymbol{u}}_i \times P\tilde{\boldsymbol{X}}_i = \boldsymbol{0} \qquad \tilde{\boldsymbol{u}}'_i \times P'\tilde{\boldsymbol{X}}_i = \boldsymbol{0}$$
$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{p}^{1T} \\ \boldsymbol{p}^{2T} \\ \boldsymbol{p}^{3T} \end{bmatrix} \tilde{\boldsymbol{X}}_i = \boldsymbol{0} \qquad \begin{bmatrix} u'_i \\ v'_i \\ 1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{p}'^{1T} \\ \boldsymbol{p}'^{2T} \\ \boldsymbol{p}'^{3T} \end{bmatrix} \tilde{\boldsymbol{X}}_i = \boldsymbol{0}$$
$$\begin{bmatrix} v_i\boldsymbol{p}^{3T} - \boldsymbol{p}^{2T} \\ \boldsymbol{p}^{1T} - u_i\boldsymbol{p}^{3T} \\ u_i\boldsymbol{p}^{2T} - v_i\boldsymbol{p}^{1T} \end{bmatrix} \tilde{\boldsymbol{X}}_i = \boldsymbol{0} \qquad \begin{bmatrix} v'_i\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{2T} \\ \boldsymbol{p}'^{1T} - u'_i\boldsymbol{p}'^{3T} \\ u'_i\boldsymbol{p}'^{2T} - v'_i\boldsymbol{p}'^{1T} \end{bmatrix} \tilde{\boldsymbol{X}}_i = \boldsymbol{0}$$
$$\Updownarrow \qquad \Updownarrow$$
$$\begin{bmatrix} v_i\boldsymbol{p}^{3T} - \boldsymbol{p}^{2T} \\ u_i\boldsymbol{p}^{3T} - \boldsymbol{p}^{1T} \end{bmatrix} \tilde{\boldsymbol{X}}_i = \boldsymbol{0} \qquad \begin{bmatrix} v'_i\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{2T} \\ u'_i\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{1T} \end{bmatrix} \tilde{\boldsymbol{X}}_i = \boldsymbol{0}$$

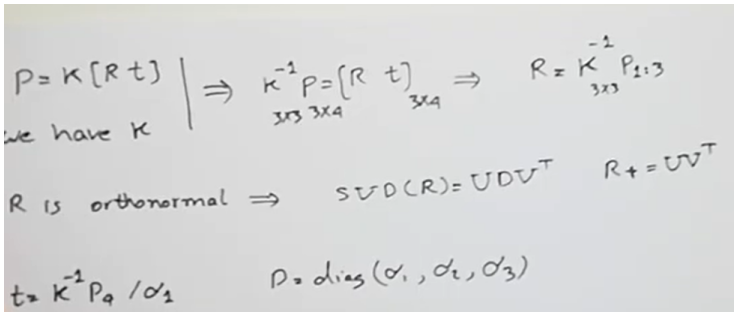
$$\begin{bmatrix} y\boldsymbol{p}_3^\top - \boldsymbol{p}_2^\top \\ \boldsymbol{p}_1^\top - x\boldsymbol{p}_3^\top \\ y'\boldsymbol{p}_3'^\top - \boldsymbol{p}_2'^\top \\ \boldsymbol{p}_1'^\top - x'\boldsymbol{p}_3'^\top \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

cross product of two vectors in the same direction is zero.
Third line is a linear combination of the first and second lines
This is useful for when there are multiple cameras
AX=0
SVD = U,S,V
From V Take eigenvector corresponding to the smallest eigenvalue.
***Given two camera poses and linearly triangulated points, X, refine the locations of the 3D points that minimizes reprojection error

PnP

https://www.youtube.com/watch?v=RR8WXL-kMzA
Perspective n point: Estimate pose of a calibrated camera, given the 3D points and their correspondences.





if det of R is neg, negate C and R signs

G. Bundle Adjustment

https://www.youtube.com/watch?v=RR8WXL-kMzA

Fine tuning parameters: Rotation, Translation(position of camera center), 3D points
Position of point in 3D as well as the amera's relative position
U = KR[I | -C]X
E = actual_image_point - U
Minimize reprojection error: min|| e ||**2 = min|| U - f(R,C,X) ||**2
Non linear optimization: X_new = X + delta_x*e
Delta_x = (J_trans*J)*(-1)*J_trans
By calculating the jacobian, for each output a differentialtion is done wrt the variables. Huge matrix but Its a sparse matrix

Kalman Filter:
- Prediction based on last estimate
- error b/w prediction and actual measurement
- update prediction

H. Loop Closure

- Global Pose Optimization, loop detection+loop closure
- reduce the drift in both the map and pose correction
- Real time loop closing based on t**he optimization of a pose graph called the Essential Graph**. It is built from a spanning tree maintained by the system, loop closure links and strong edges from the **covisibility graph**.
- As we continue to analyze features in frames, we start encountering cases wherein the features seen in the current frame are almost identical to the one seen in the starting frames.
- At this stage we say that loop closure is detected and start the loop closure tuning.
- The estimated trajectory and the true trajectory are significantly different due to the accumulated errors
- The objective of loop closure is to fuse the last frame position with the starting frame position by correcting the errors in the estimated pose for all the frames.

The estimated fundamental matrix is derived from an erroneous pose and will thus show significant deviation from the fundamental matrix equation. Thus the error function is -

$$E_{N,0} = \sum \left(\begin{bmatrix} x'_N & y'_N & 1 \end{bmatrix} * F_{N,0} * \begin{bmatrix} x'_0 \\ y'_0 \\ 1 \end{bmatrix} \right)^2$$

Where (x'_N, y'_N) and (x'_0, y'_0) are the matching point 2D co-ordinates of key points common for Fr_0, and Fr_N.

We minimize $E_{N,0}$ by tuning the estimated translation magnitudes as well as the rotation and translation for the pose of individual keyframes.

I. Fish Eye

- Fish eye calibration: https://www.mathworks.com/help/vision/ug/fisheye-calibration-basics.html
- Fish eye calibration paper
- The fish eye camera model takes into account the **radial fisheye distortion**. the polynomial describing the radial distortion is a function of an anglular distance from the centre of perspective, rather than a linear distance in the image

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \frac{\theta}{r}\left[1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8\right] \begin{pmatrix} x_n \\ y_n \end{pmatrix},$$

where $r = \sqrt{x_n^2 + y_n^2}$ and $\theta = \tan^{-1}(r)$.

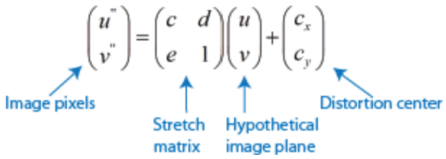
What I did: Well I was working on the fish eye stereo model two weeks back but due to some disadvantages of the entire module of the raspberryPi, I shifted to a much more powerful module with a normal stereo camera.

The work that I did on the fish eye stereo camera was that, I implemented depth estimation and a deep learning model for object detection to obtain depth of certain objects.
And using the distortion matrix acquired during the calibration process of the fisheye model I undistorted the image before going through with the depth estimation and feeding the image to the object detection model.

So given the FOV and resolution of the image, I was able to extract the depth information of the image quite accurately but running the YOLOv3 on stereoPi gave out some latency issues due to which the output wasnt real time. And that was before applying any other computations on the processor. So I decided to move to a different technology which would give me better performance so now I am the Jetson TX2 model with a normal stereo camera.

Fish eye Calibration: Well I havent written the fish eye calibration algorithm myself, I used the API provided by StereoPi. But I have a high level overview of the entire calibration process which is slightly different from the normal Zhang's calibration approach.

Calibration Explaination: These cameras use a complex series of lenses to enlarge the camera's field of view, enabling it to capture wide panoramic images. The lenses achieve this extremely wide angle view by distorting the lines of perspective in the images
The intrinsic parameters need to be slightly modified in order to take in to account the **stretching** and **radial distortion** in the image. The stretch matrix compensates for the sensor-to-lens misalignment, and the distortion vector adjusts the origin location of the image plane.
The following equation relates the real distorted coordinates (u'',v'') to the ideal distorted coordinates (u,v).



I am not entirely familiar with the math behind it but K includes the polynomial mapping coefficients of the projection function. The alignment coefficients are related to sensor alignment and the transformation from the sensor plane to a pixel location in the camera image plane.

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \lambda \begin{pmatrix} u \\ v \\ a_0 + a_2\rho^2 + a_3\rho^3 + a_4\rho^4 \end{pmatrix}$$

- (u, v) are the ideal image projections of the real-world points.
- λ represents a scalar factor.
- a_0, a_2, a_3, a_4 are polynomial coefficents described by the Scaramuzza model, where $a_1 = 0$.
- ρ is a function of (u,v) and depends only on the distance of a point from the image center: $\rho = \sqrt{u^2 + v^2}$.

Steps:

Checkerboard points: initialized the dimensions and the known 3D coordinates in world frame.
Checker board corners detection in the image plane.

- Polynomial **mapping coefficients** for the projection function
- Center of **distortion** in pixels, specified as a [cx cy] vector.
- **Stretch** matrix: Transformation from the sensor plane to a pixel in the camera image planeThis misalignment is caused by the lens not being parallel to the sensor and by the digitization process.

Calibrate stereo: get K, D, Rot, trans, E, F
Rectify
Undistort

J. Deep Learning: Depth

https://arxiv.org/pdf/1812.11941.pdf
Encoder-Decoder approach. Pixel in Pixel Out. Similar to semantic segmentation

Classical methods: Issues such as lack of scene coverage, scale ambiguities, or reflective ma-

terials all contribute to ambiguous cases where geometry cannot be derived from appearance.
CNN models produce reasonable depth maps from a single or couple of RGB input images in real time speeds.
Single image depth estimation
Transfer learning: encoders: image classifiers
such encoders that do not aggressively downsample the spatial resolution of the input tend to produce sharper depth estimations especially with the presence of skip connections.
Encoder: densenet pretrained on imagenet
Decoder: transpose cnn with skip connections
Augmentations: reduce overfitting. learning of expected statistical properties: no vertical flips, only horizontal. Change colour channels of the image while keeping the ground truth to be the same
Loss: depth regression problems considers the difference between the ground-truth depth map y and the prediction of the depth regression network ˆy
-Intersection over Union(IOU)
-structural similarity index measure (SSIM)(Takes into account the distribution and covariance)
https://en.wikipedia.org/wiki/Structural_similarity
-level weighted *structural similarity* (LWSSIM)

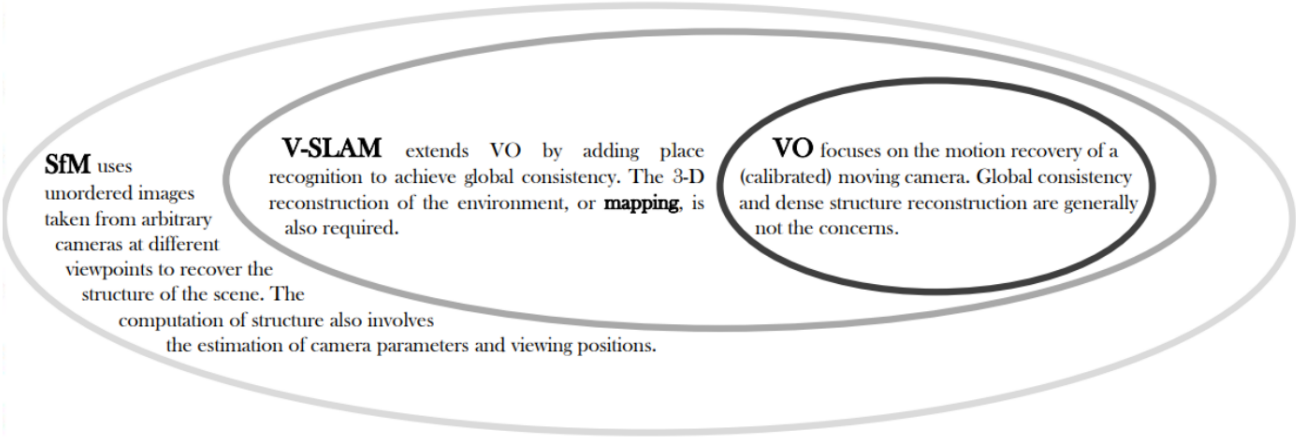
K. V-SLAM, VO, SFM

Visual odometry: Local trajectory
local map is used to obtain a more accurate estimate of the local trajectory
Using the last n poses, windowed bundle adjustment
Figuring out it's own pose regardless of the environment map.

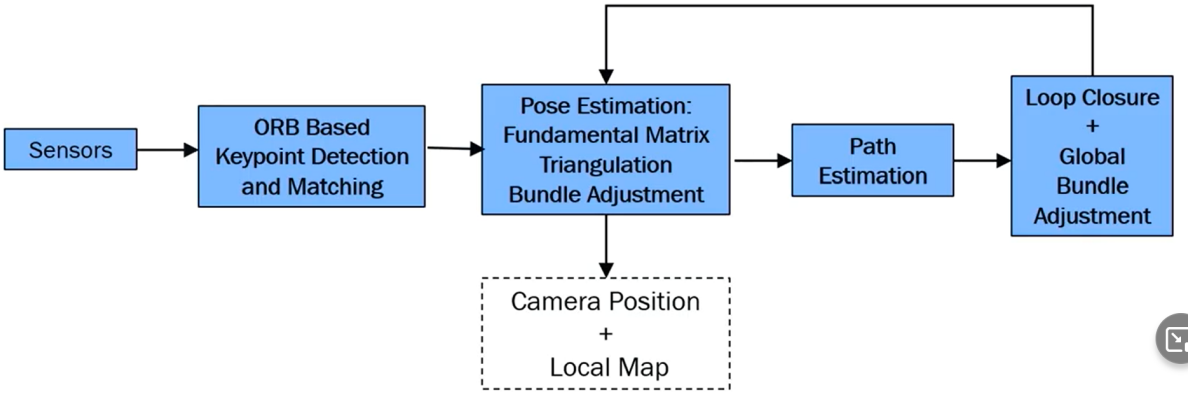
Visual SLAM: Simultaneous localization and mapping
https://arxiv.org/pdf/1502.00956.pdf
Global map
global, consistent estimate of the robot path
Global map building using loop detections and loop closures
reduce the drift in both the map and pose correction
Image sequences need to be **ordered** and from the **same** camera

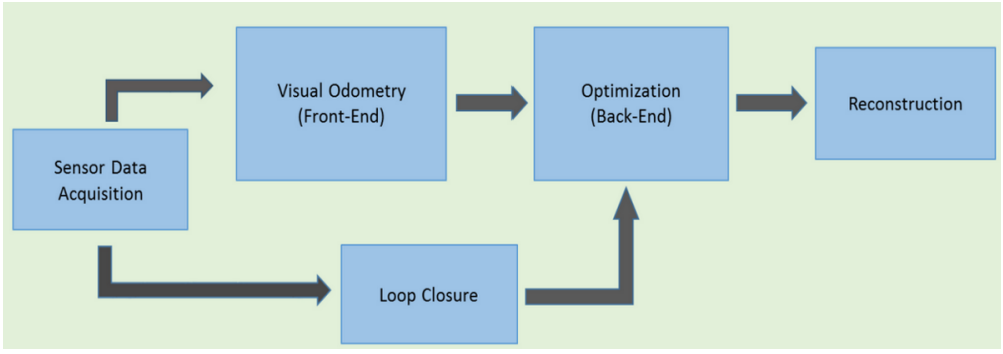
Resconstruction = dense representation of environment for navigation or obstacle avoidance

Structure from motion: SfM is usually performed offline using **unordered** sequences of images. SfM is mostly concerned with creating a map of the environment using several images taken from different perspectives. Can be taken from different camera



Slam:





L. Zhang's Calibration

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr98-71.pdf

1.Checkerboard points: initialized the dimensions and the known 3D coordinates in world frame.

2.Checker board corners detection in the image plane.

3.Detect the feature points in the images

4.Estimate the five intrinsic parameters and all the extrinsic parameters

5.Estimate the coefficients of the radial distortion by solving the linear least-squares

6.Refine all parameters by minimizing

3D points and corresponding 2d points are known, extract the projection matrix
Checkerboard points: initialized the dimensions and the known 3D coordinates in world frame.
Checker board corners detection in the image plane.
All points are on one plane(checkerboard), z axis is removed from the rotation matix
H = K[r1 r2 t]
Extract H
[-X -Y 1 0   0 0 xX xY x]
[0   0 0 -X -Y 1 yX yY y]
Using H and orthonormal properties of the rotation matrix (r1.T*r2=0, ||r1||=||r2||=1)
R1 = k_inv*h1

B = K_inv_trans*K_inv
B matrix is symmentric
Create a V matrix using these 2 equations, apply SVD on it to get the elements of B
Using the cholesky decomposition= chol(B)=A*A_trans
Equations of the principle points, pixel resolution, skew
Extract K
Calculate r1, r2, r3, t
Radial distortion:
Keep zero and then apply least square to update the loss to improve reprojection error by changing the intrinsic param and the distortion. And using that get the better extrinsic mat

$$\breve{u} = u + (u - u_0)[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2]$$
$$\breve{v} = v + (v - v_0)[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2]$$

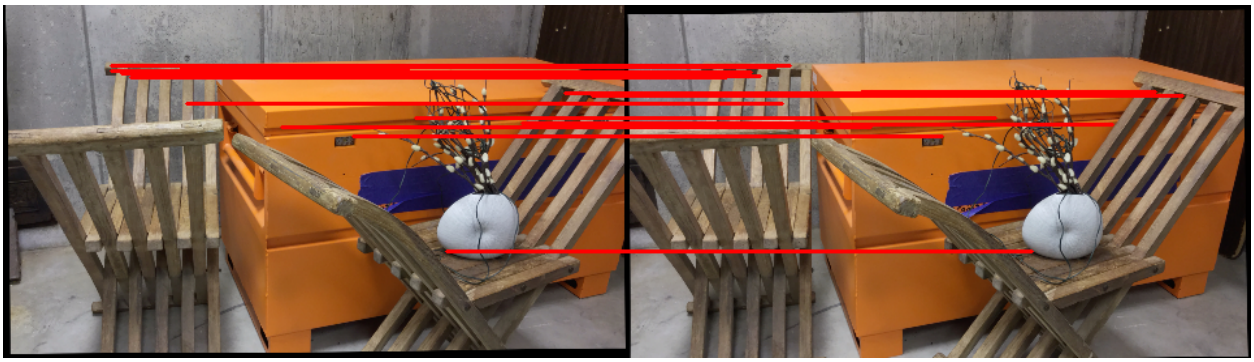
x=distortion free points
k=distortion coefficient

1. Find Matches and Extract F matrix

- **Finding matches and the Fundamental Matrix**:
- Keypoint matches between the two images are extracted using SIFT.

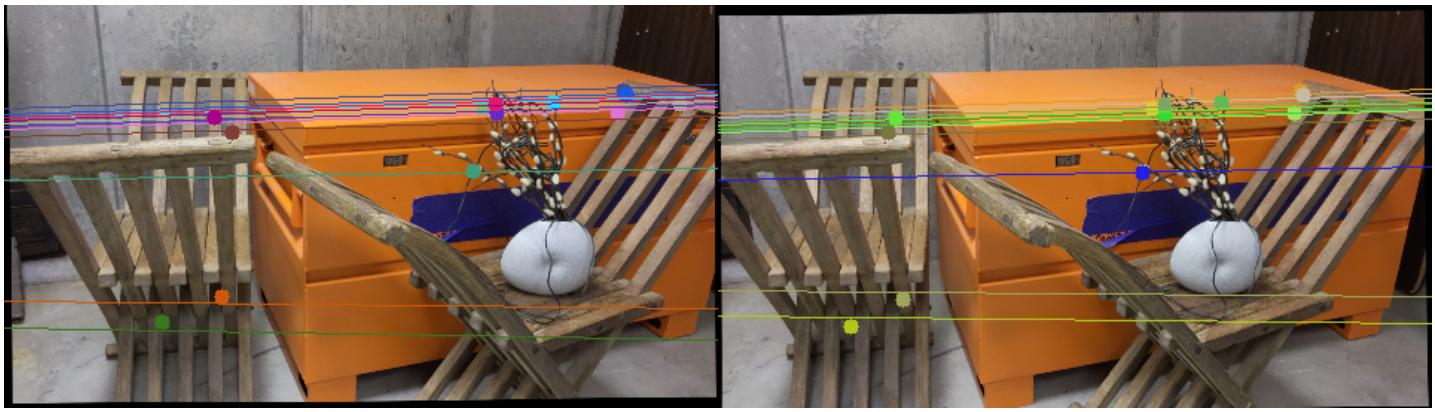
- These matches are then filtered out to get the best matches by rejecting outline correspondence using RANSAC by following the algorithm below:

$n \leftarrow 0$
for $i = 1 : M$ **do**
  Choose 8 correspondences, $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$, randomly
  `F = EstimateFundamentalMatrix(`$\hat{\mathbf{x}}_1$`,` $\hat{\mathbf{x}}_2$`)`
  $\mathcal{S} \leftarrow \emptyset$
  for $j = 1 : N$ **do**
    if $|\mathbf{x}_{2j}^{\mathsf{T}}\mathbf{F}\mathbf{x}_{1j}| < \epsilon$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$
    end if
  end for
  if $n < |\mathcal{S}|$ **then**
    $n \leftarrow |\mathcal{S}|$
    $\mathcal{S}_{in} \leftarrow \mathcal{S}$
  end if
end for

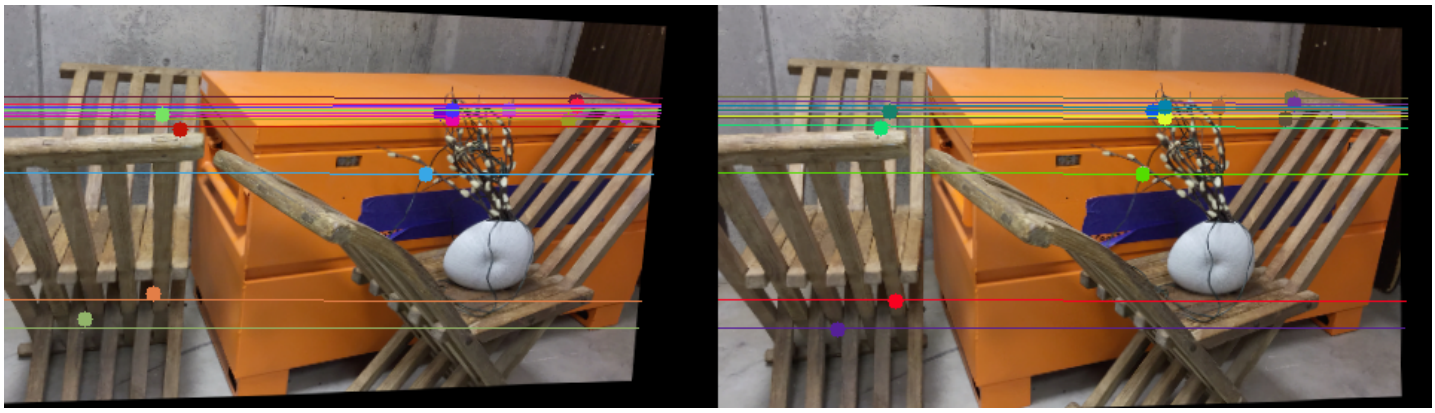


2. Rectification

Using the fundamental matrix and the inbuilt OpenCV function computeCorrespondEpilines(), derive epilines for the right and the left image. Line on image 2 = F*pts_of_img1



When these epilines are plotted, it can be observed that the lines are not parallel. And in order to obtain depth, the epipolar lines need to be parallel.
Using F, matched points from left and right images and the inbuilt function stereoRectifyUncalibrated() the homography matrices of both the images are obtained which can be used to rectify the images.
Using these homography matrices, derive the corresponding warped points from one image to the other one for both.
The epilines now derived from these transformed points will come out to be parallel.



Problems:
Understanding warping and drawing the epilines took a lot of time.

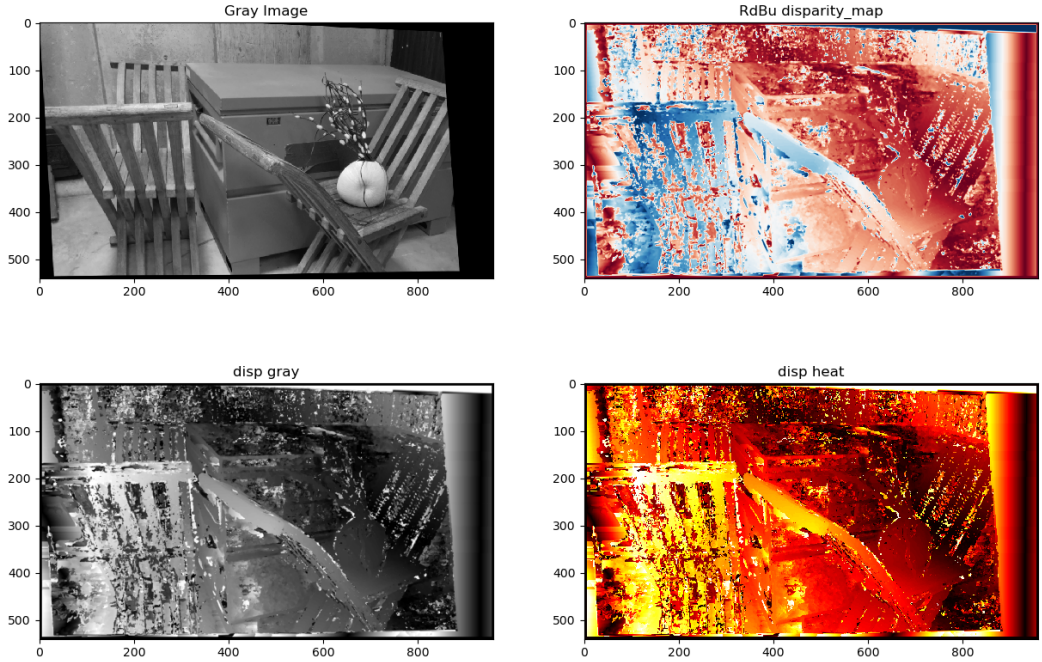
3. Correspondence

The warped images will now be used to calculate the disparity between each of the pairs of pixels in the image. This is done using the Sum of Squared Differences (SSD) as a patch similarity measure with a fixed window.
For each patch of frame in the left image, we will go through its corresponding y-axis in the right image along the x-axis. This is possible because we have now warped the image and the epipolar lines are parallel. This guarantees that the matching patch of frame in the right is on the say y coordinate as the left one.

Using SSD(sum of squared differences) we select the patch which is the most similar to the left patch and calculator the patches disparity.

$$SSD(A, B) = \sum_{i,j}(A_{ij} - B_{ij})^2$$

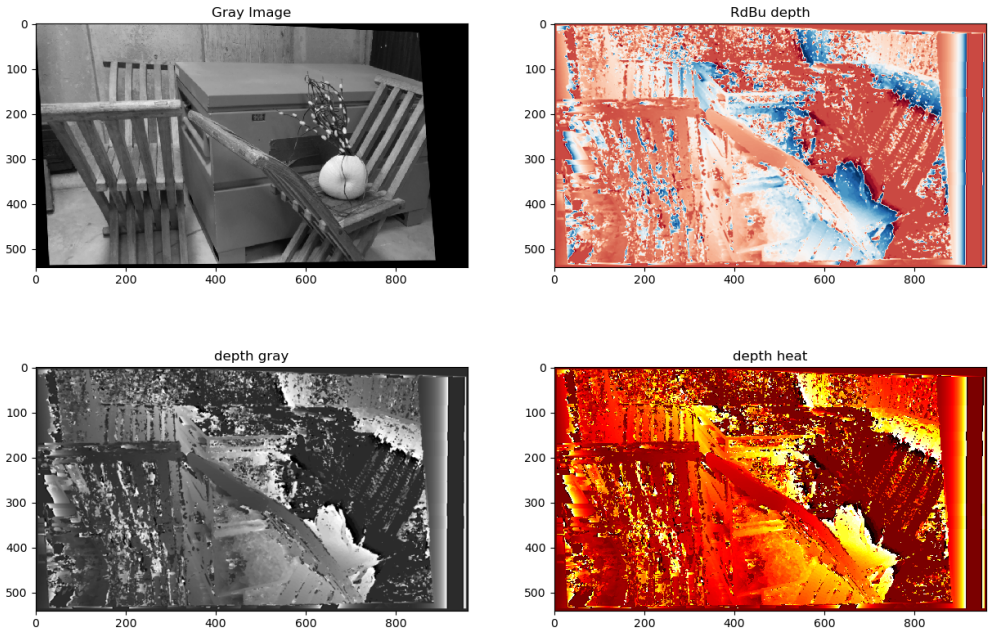
The difference between the placement of a particular pixel is both images will tell us how far the pixel is.



Problem:
Multiple combinations of disparity range and window size had to be done to get a proper Disparity map.
Had to resize image to decrease the time of computation drastically

4. Compute Depth Image

Once the disparity map is generated, we can use this $z = f(\frac{baseline}{disparity})$ to calculate depth for each point in the image. At time the max distance can become exponentially high. This results in the reduction of significance of closer objects. Hence the depth values need to the thresholded at a certain point.



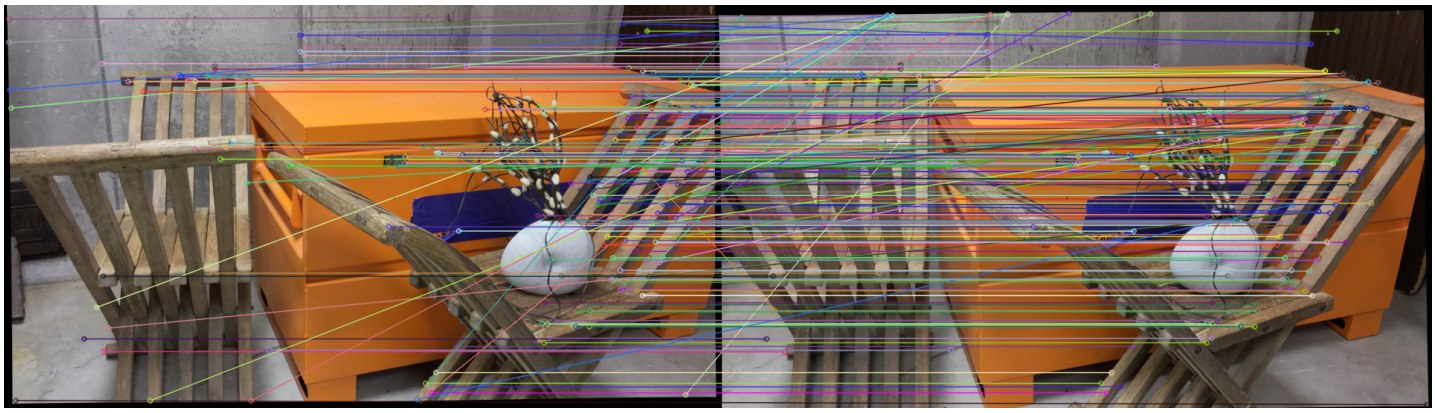
Problems:
The max distances were very high for all the datasets. Implemented a dynamic thresholding algorithm to determine the threshold point for distance.

Results can be found: here

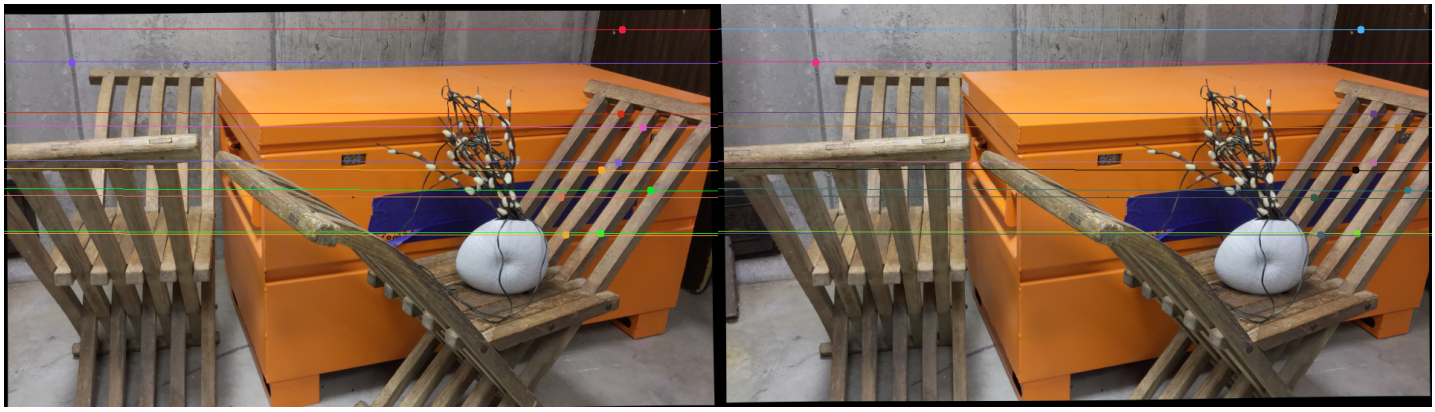
Set 1:

Fundamental Matrix:
[[3.22699441e-11 -3.18672381e-08 -7.81216444e-07]
[2.28852268e-08 -1.24835714e-08 3.14997429e-03]
[3.72424434e-06 -3.14098871e-03 -2.13460370e-03]]
Number of good matches found: 162 out of 200
Essential Matrix:
[[1.69085987e-05 -1.76400644e-02 -5.85095513e-03]
[1.26936433e-02 -3.73895481e-03 9.99895706e-01]
[5.11504173e-03 -9.99824084e-01 -3.70143529e-03]]
Final poses:

R:
[[-0.99947986 -0.01101169 0.03031079]
[-0.01089909 0.99993309 0.00387746]
[0.03035145 -0.00354508 0.999533]]
C:
[-0.99982728 -0.00578517 0.01766176]
All matches through SIFT:

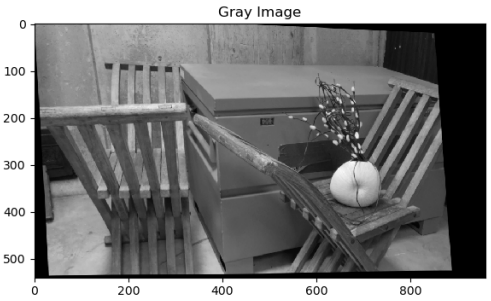


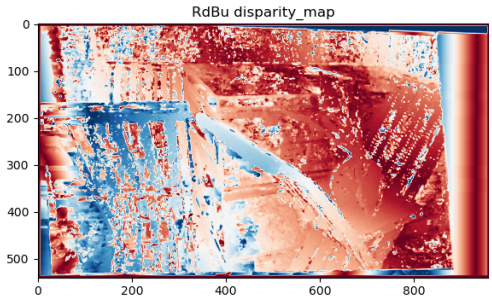
Non-parallel epilines:

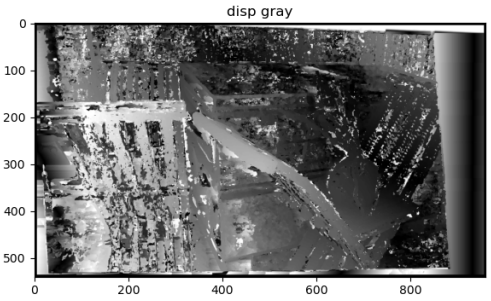


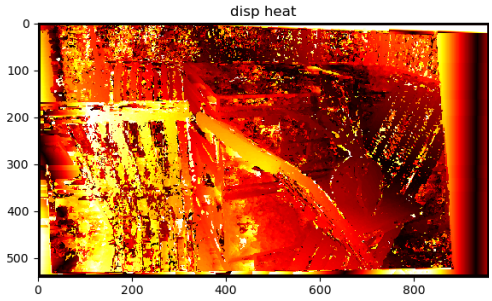
Rectified images:

Disparity:

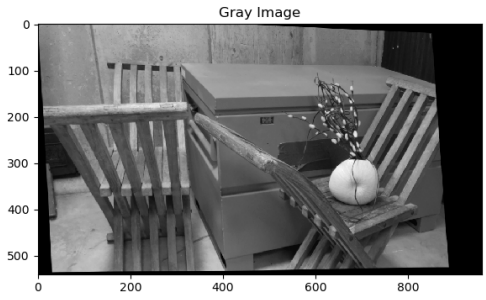

Gray Image

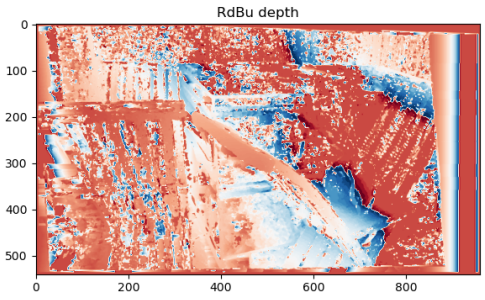

RdBu disparity_map

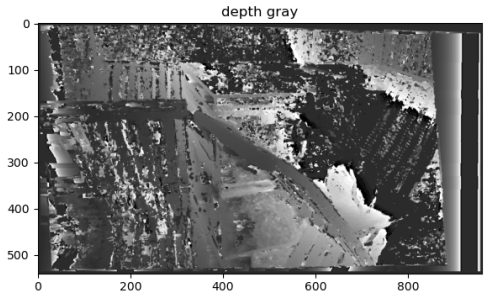

disp gray

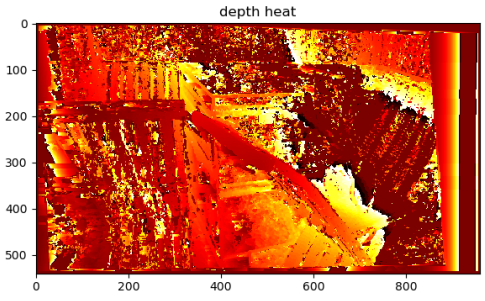

disp heat

Depth:

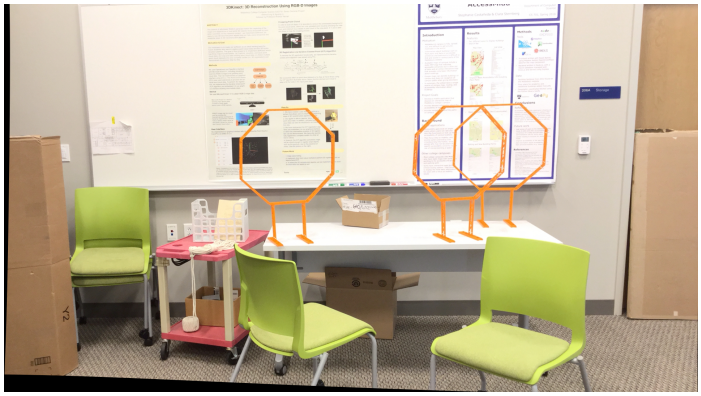

Gray Image


RdBu depth


depth gray


depth heat

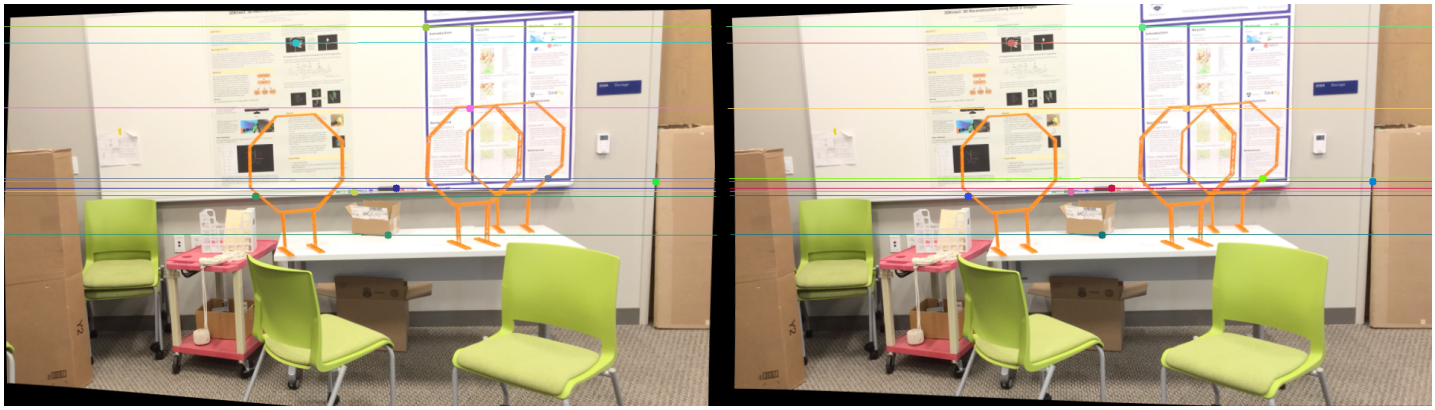
Set 2:

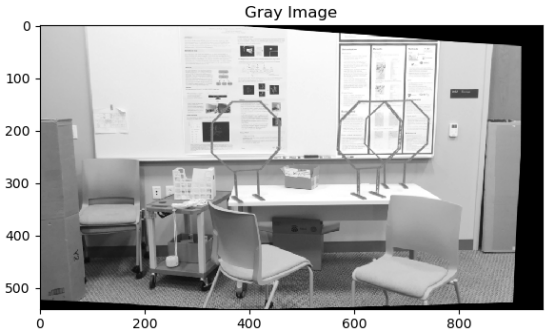
Fundamental Matrix:
[[-1.64231140e-09 1.97439670e-09 -1.14217018e-04]
 [-5.79102471e-08 6.93382572e-09 -3.89058569e-03]
 [1.27752729e-04 3.92216270e-03 -1.42650573e-02]]
Number of good matches found: 199 out of 200
Essential Matrix:
[[-7.28117881e-04 8.18906103e-04 -2.90783433e-02]
 [-2.56759557e-02 1.12931140e-03 -9.99246384e-01]
 [2.42538267e-02 9.99705384e-01 5.29499284e-04]]

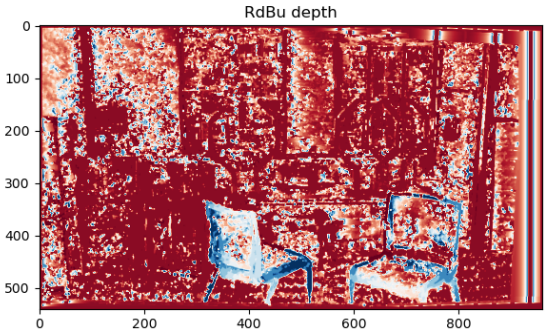
Final poses:
R:
[[-9.98226904e-01 5.33008630e-02 2.64965322e-02]
 [5.33131937e-02 9.98577811e-01 -2.41343777e-04]
 [2.64717130e-02 -1.17169891e-03 9.99648876e-01]]
C:
[-9.99576535e-01 2.90883672e-02 7.85941052e-04]
All Matches:

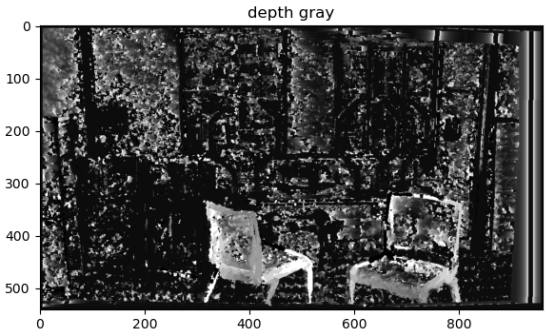
Rectified image:

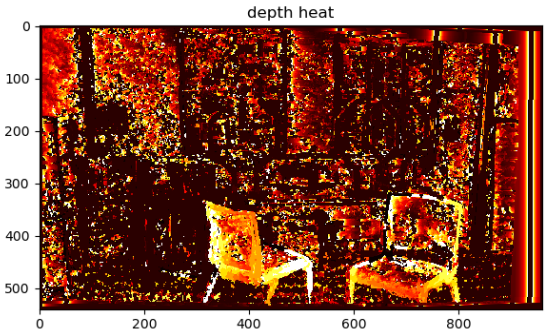


Disparity:

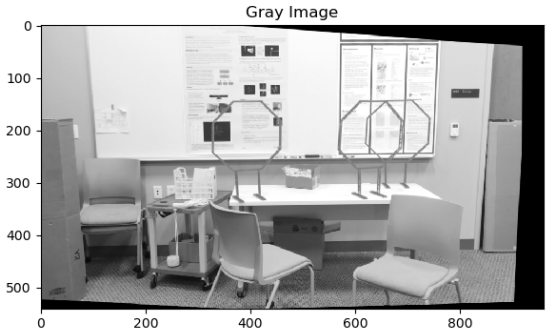


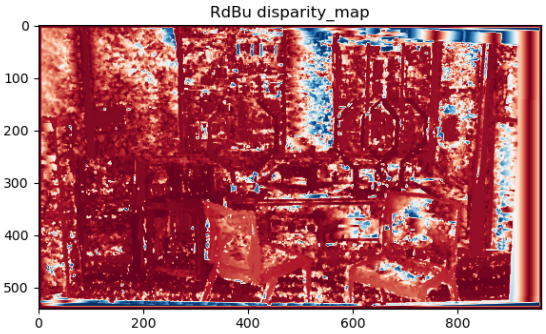


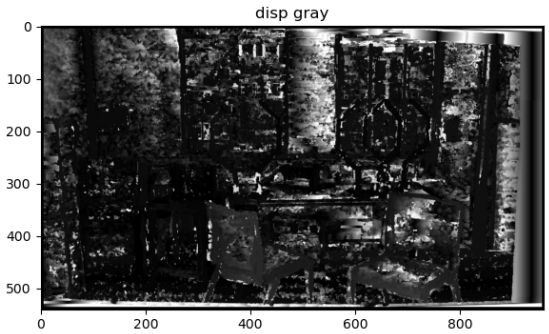


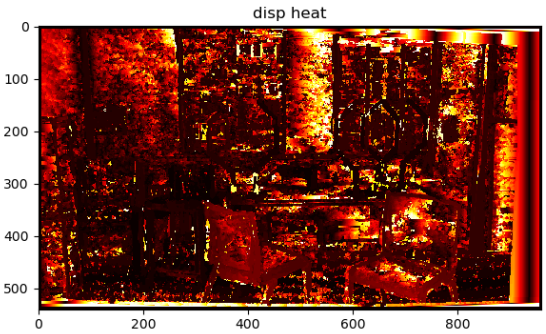


Depth:





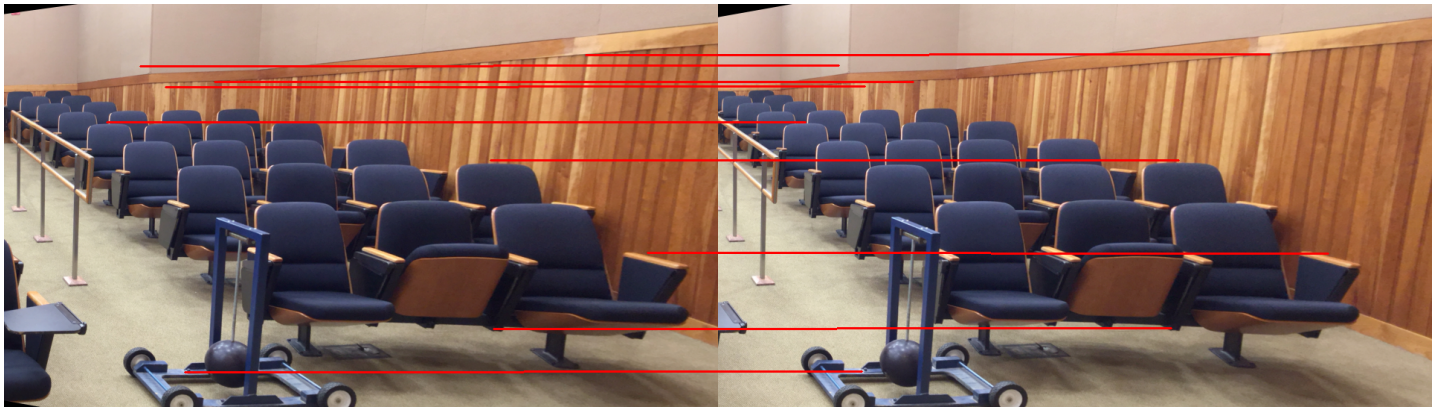




Set 3:

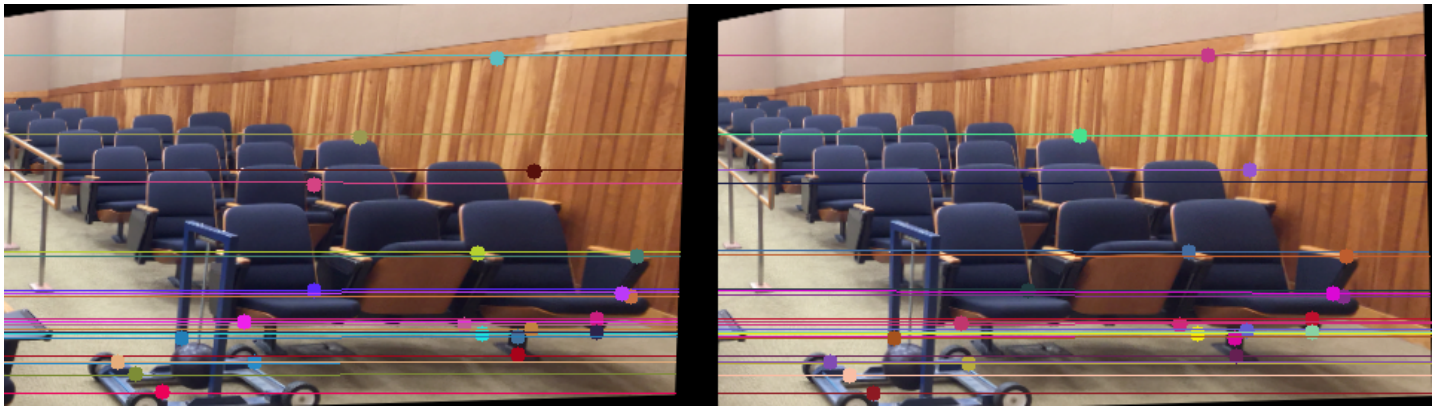
Fundamental Matrix:
[[-1.19508029e-10 1.97133027e-07 -2.84977259e-05]
[-2.01639128e-07 2.26741327e-08 3.10247895e-03]
[3.09625035e-05 -3.09161145e-03 -1.13804438e-03]]
Number of good matches found: 185 out of 200
Essential Matrix:
[[2.37844774e-05 1.07382982e-01 2.45209014e-02]
[-1.08785379e-01 7.22092117e-03 9.93717181e-01]
[-2.58872738e-02 -9.93874532e-01 7.11680768e-03]]
R:
[[9.99998555e-01 -5.85906478e-04 1.59586065e-03]

[5.97383819e-04 9.99973894e-01 -7.20098854e-03]
[-1.59159988e-03 7.20193148e-03 9.99972799e-01]]
C:
[0.9939153 -0.02529356 0.10720362]
Matches:

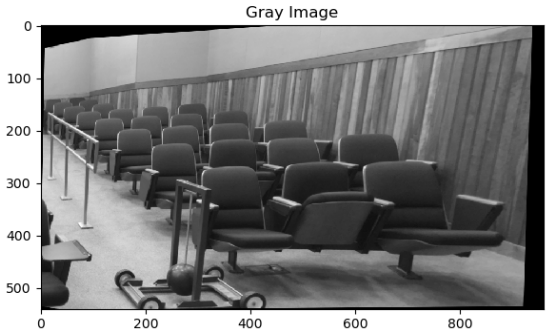


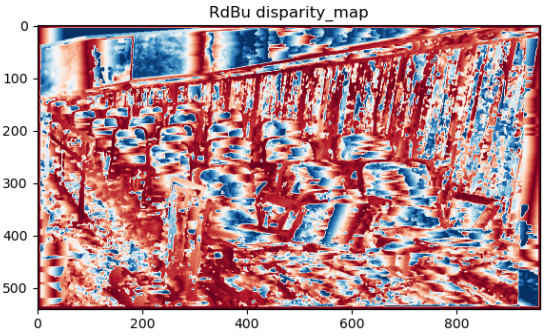
Epilines before rectification

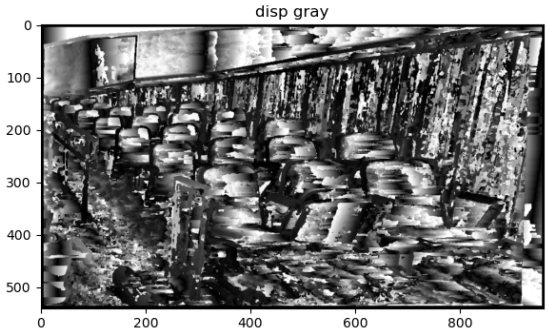
Epilines after rectification

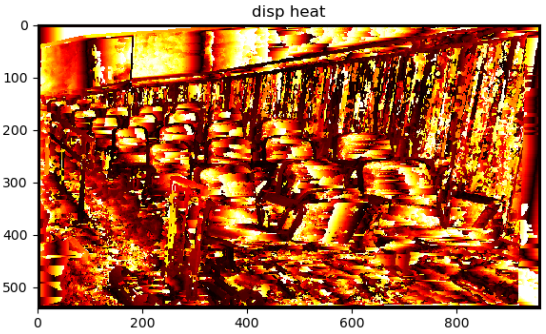


Disparity:

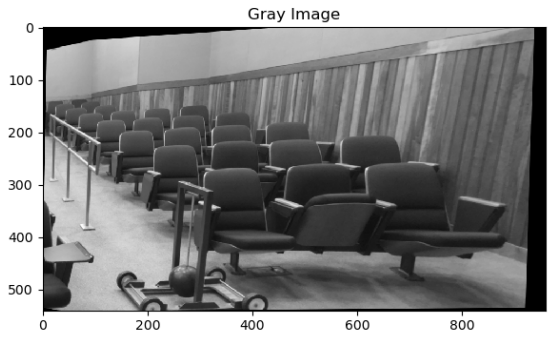


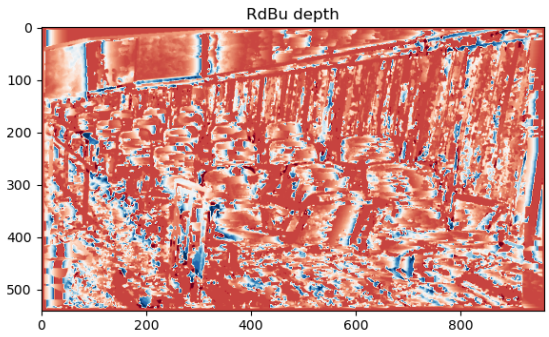


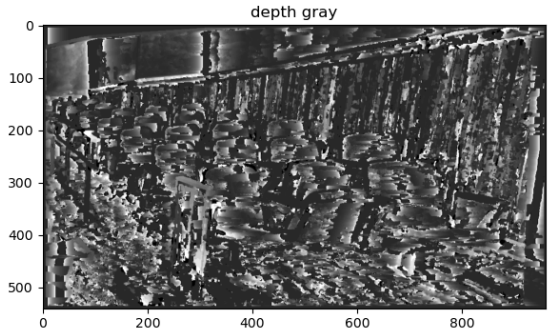


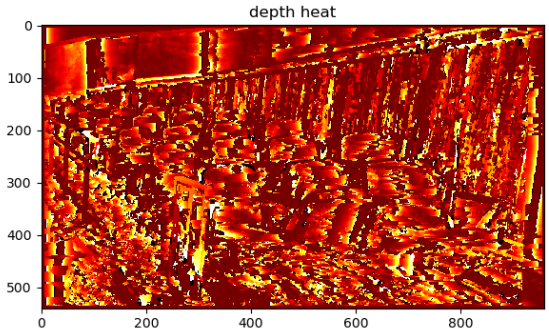


Depth:









References

- http://www.r-5.org/files/books/computers/algo-list/image-processing/vision/Richard_Hartley_Andrew_Zisserman-Multiple_View_Geometry_in_Computer_Vision-EN.pdf
- https://www.cis.upenn.edu/~cis580/Spring2015/Projects/proj2/proj2.pdf
- https://johnwlambert.github.io/stereo/
- https://cmsc733.github.io/2022/proj/p3/#featmatch