Final Exam Prep

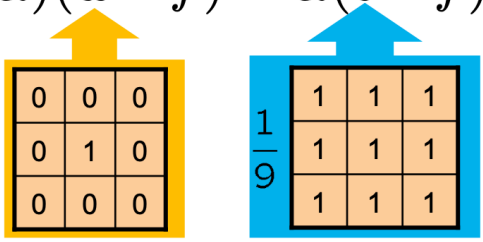
- Before Friday: modules 1-3
- Friday: modules 4-6
- Saturday: modules 7-10
- Sunday: quizzes

Week 1:

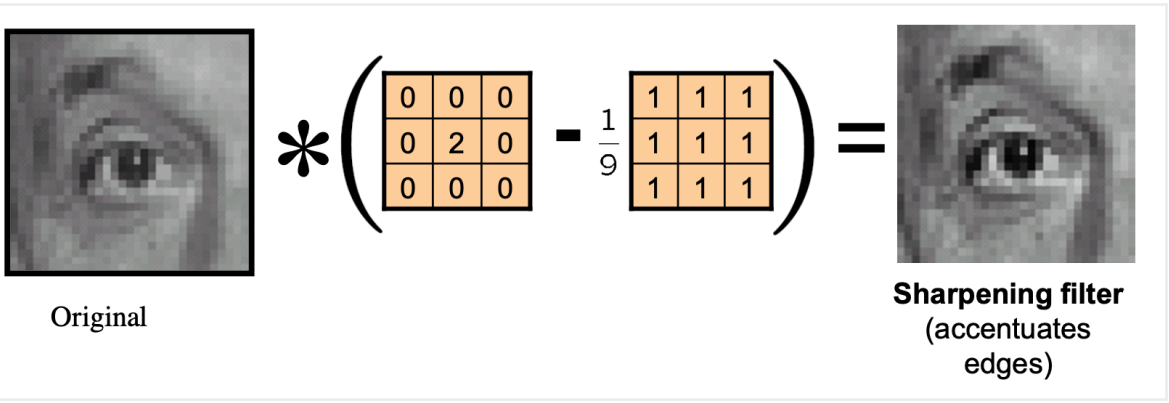
- Colors: RGB, HSI/HSV/HSB (hue-saturation-intensity/value/brightness), etc
- Image Filtering cv2 functions
- Linear Filtering: used for noise deduction by averaging neighboring pixels since nearby pixels are likely to belong to same object and therefore have similar color (mean filtering)
- Another option is kernel filter with cross correlation, convolution)
  - Kernel examples: all 1's with a factor scaler, like 3x3 1's * 1/9 gives blur; 3x3 kernel with zeros and 1 in middle gives identical images.

Sharpening

$$
\begin{aligned}
f_{sharp} &= f + \alpha(f - f_{blur}) \\
&= (1+\alpha)f - \alpha f_{blur} \\
&= (1+\alpha)(w * f) - \alpha(v * f)
\end{aligned}
$$



$$= ((1+\alpha)w - \alpha v) * f$$



- Non-Linear filters: thresholding, rectification, median of neighbors
- Morphology: most common binary (or greyscale) image operations
  - Structuring element: **cv2.getStructuringElement()**
  - Erosion, dilation, opening, closing (all rely on convulsion with a structuring element like a disk, rectangle, or any other shape)
  - Opening is just another name of **erosion followed by dilation** ex cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel).
  - Closing is reverse of Opening, **Dilation followed by Erosion** ex cv2.morphologyEx(imp, cv2.MORPH_CLOSE, kernel).
  - Morphological Gradient: the difference between dilation and erosion of an image (result looks like the outline of the object) ex cv2.morphologyEx(imp, cv2.MORPH_GRADIENT, kernel).
- Connected Components Analysis
  - Connectivity if two neighboring pixels share the same or similar intensity/color value
  - Works for binary, grayscale, color

Extra: contours

Qs:

*Exercises_Wk_1, exercise 2: how to remove red or green ball? (Look at weekly project ex)

Week 2:

- Image Features
- Feature Detection: find points/areas in an image likely to be detected in other images too
  - Ex. Harris Corner Detector (can distinguish bet. flat, edge, or corner) - rotational invariance but corners are not very unique (desk and building have corners, but desk is not similar to a building)
  - Ex. Shi-Tomasi feature detection (GoodFeaturesToTrack)
  - Ex. Difference of Gaussians (DoG)
- Feature Description: create a unique descriptor fingerprint for each feature point
- Feature Matching: find correspondences among diff images. Allows us

to match multiple images and “piece together” a larger image.
- Scale-Invariant Feature Transform (SIFT) contains all: detection, description, matching
  - Robust in: change of translation, change in scale, change in rotation, change in 3D view point, change in illumination
  - Uses DoG on scale space for blob detection (only the max or min in a neighborhood are considered, all octaves (variances/stdevs) are investigated)
  - description: Creates a histogram of gradient orientations (every 10 deg -> 36 bins)
  - matching: all-all Euclidean distance can give us 1000 features for a 640x640 img (with 128 values each feature!) Instead, use Kd-Trees.
- SIFT, SURF (Speeded-Up Robust Features), BRISK, FREAK, MSER, ORB, Patches
- Applications: motion estimation, localization, mapping, photogrammetry, image retrieval, object detection & recognition, autonomous driving, etc.

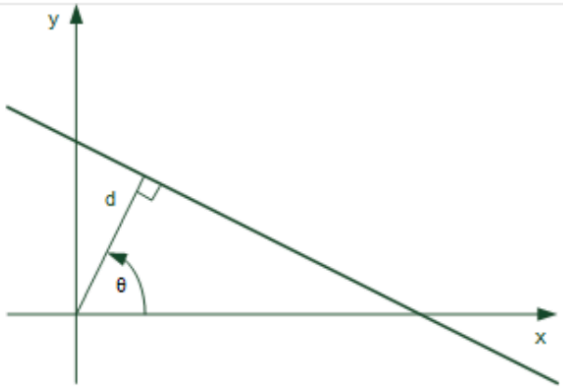
- Edge Detection
- Gradient is a vector that points in the direction of most rapid change in intensity.
  - Like all vectors, defined by its orientation and magnitude: theta=arctan(df/dy / df/dx), ||del_f|| = sqrt((df/dx)^2+(df/dy)^2)
  - Positive gradients go from black towards white.
- Derivative Filters:
  - Sobel, Scharr, Prewitt, Roberts
  - Use kernel and convolution to smooth signals.
- Canny Edge Detection: noise reduction, gradient calculation, non-maximum suppression, double threshold, edge tracking by hysteresis (history)

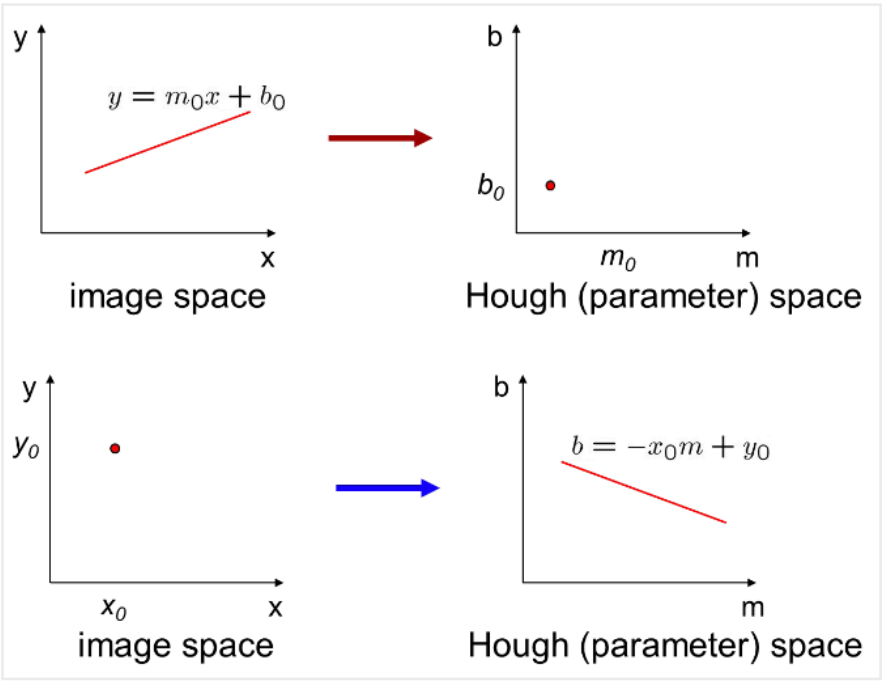
- Fitting (handling outliers)
  - Hough Transform: ax+by+c=0 is unbounded, so instead we use polar coordinates, initialize the grid using (theta, d), populate the grid by passing through the image and adding votes (see vid)
  - Identify maxima and track lines back to img
  - For non-linear features (ex circles), same approach is used but complexity grows exponentially (for circles, 3 params x,y,r are required. Not advices to go beyond 4.)

$$x\cos\theta - y\sin\theta = d$$

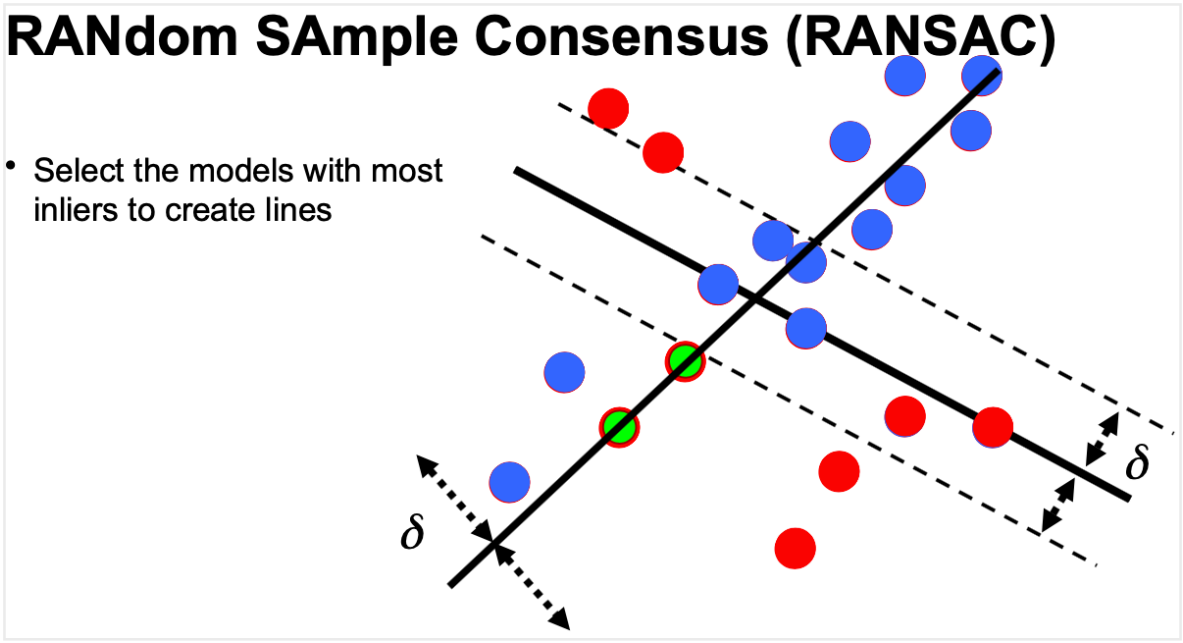
d : perpendicular distance from line to origin

θ : angle the perpendicular makes with the x-axis





- RANdom Sample Consensus (RANSAC) algorithm:
  - 1. Samples (random) # points required to fit model
  - 2. Solve for model params using samples
  - 3. Score by the fraction of inliers within a preset threshold of model
  - Repeat 1-3 until best model is found (high confidence)
  - (Like SVM's, but used for fitting, not classification like SVMs)



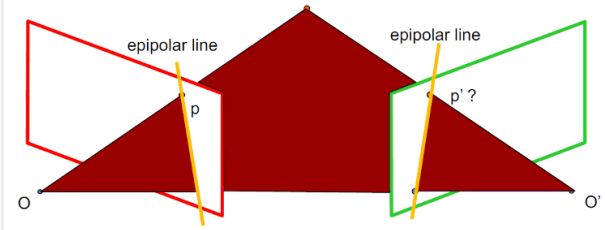
Week 3:
Stereo matching - the process of taking 2 or more images and estimatting a 3d model of the scene by finding matching pixels in the images and converting their 2d positions into 3d depths.

Epipolar Geometry

- **Baseline**: the line connecting the 2 camera centers
- **Epipole**: point of intersection of *baseline* with the image plane
- **Epipolar Plane**: the plane that contains the two camera centers and a 3D point in the world
- **Epipolar Line**: intersection of the *epipolar plane* with each image plane

Searching for feature matches along epipolar lines becomes therefore a 1D problem!

- **All epipolar lines intersect at the epipoles.**

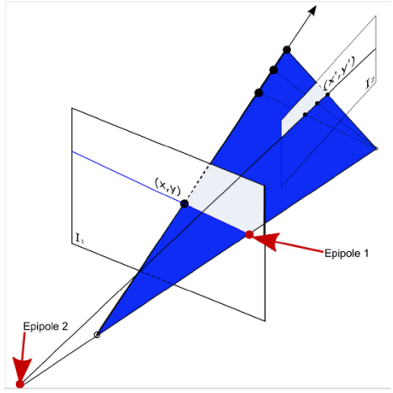


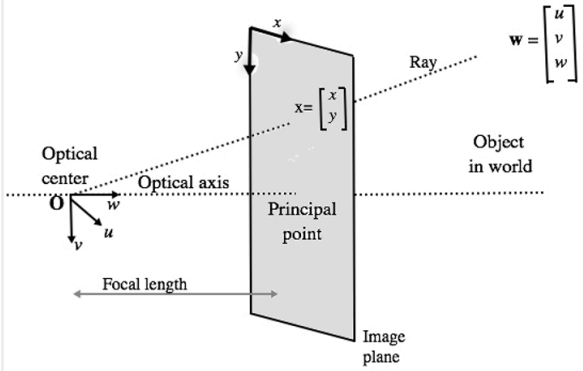
Baseline: the line connecting the two camera centers

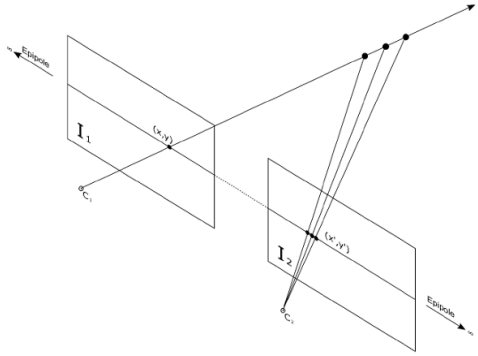
Epipole: point of intersection of *baseline* with the image plane

Epipolar plane: the plane that contains the two camera centers and a 3D point in the world

Epipolar line: intersection of the *epipolar plane* with each image plane







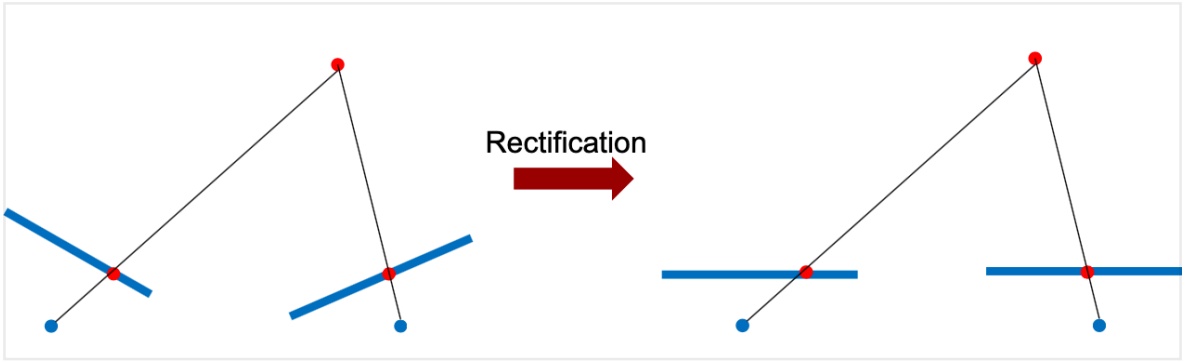
Consider a stereo vision system.
Choose all the statements below that are true.

- ✓ ☐ All epipolar lines are parallel to the optical axis
- ✓ ☐ In certain cases, there can exist just one epipole
- ➡ ✗ ☐ The epipoles can be outside the images
- ✓ ☐ The baseline intersects the epipolar plane at the epipoles
- ✓ ☐ All epipolar lines meet at the optical center
- ➡ ✗ ☐ The epipoles lie on the baseline-containing line
- ➡ ✗ ☐ All epipolar lines intersect at the epipoles

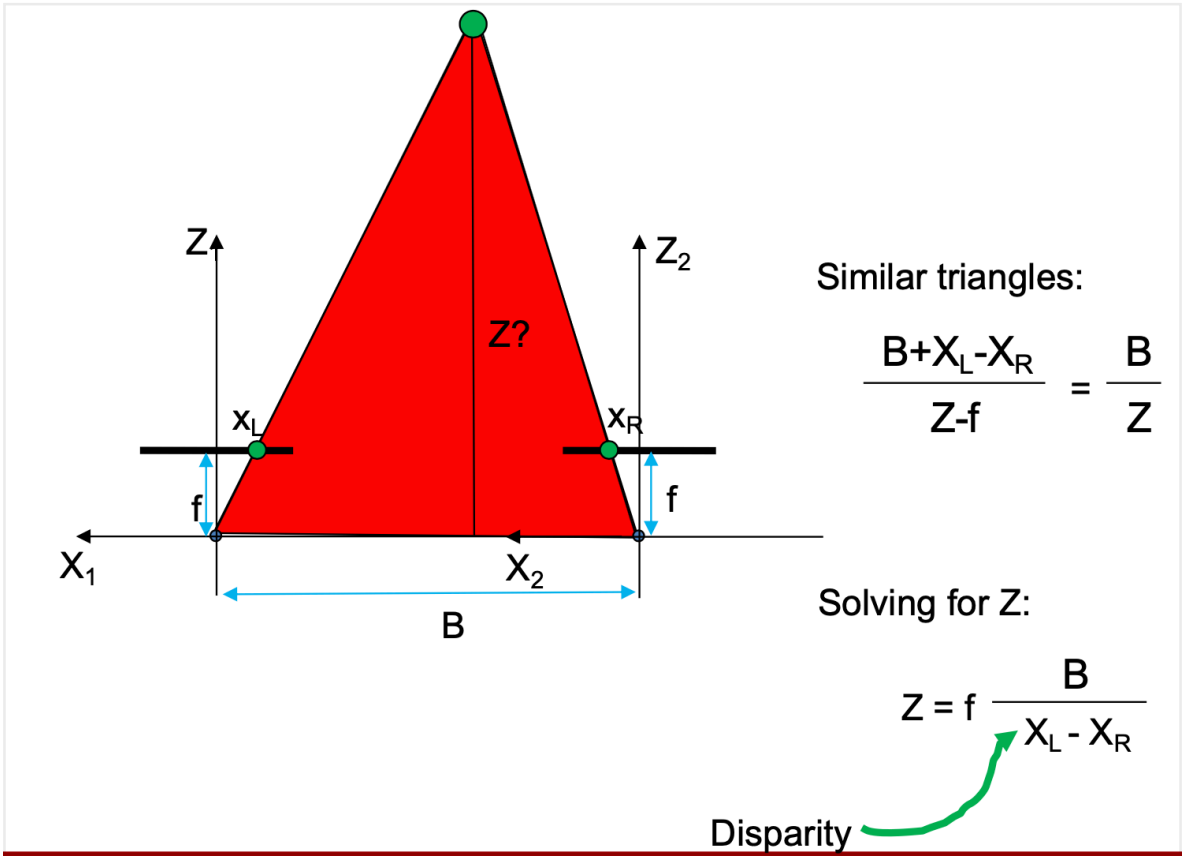
1. False. The optical axis is the line going from O to the point.
2. False. There will always be two epipoles for a stereocamera.
3. True. If the cameras are rectified.
4. False. The epilos are where the basline intersects the image plane.
5. False. The optical center has nothing to do with epipolar lines.
6. True. The epilos are where the basline intersects the image plane.
7. True. They all cross the epipoles.

Rectification

- Initial images reprojected on common plane parallel to baseline of initial images



Disparity to find depth: large disparity means objects are close to the camera; small disparity means objects are far away (inverse relation)



Correspondence Problem

- There are also “soft” constraints beyond epipolar geometry that help identify corresponding points:
  - Similarity, uniqueness, ordering, disparity gradient is limited
- Spare stereo matching:
  - Extract features (using e.g. SIFT, SURF, Harris, etc)
- Dense stereo matching:
  - Local methods (area-based)
    - Disparity of each pixel determined only by the information of the pixel itself and its neighborhood (hence “local”)

    - For each epipolar line, compare each pixel in left img with every pixel on same epipolar line in right img and choose pixel that maximizes a similarity metric (or minimizes a dissimilarity metric, like minimizing a difference in e.g. SAD or SSD)
    - Improvement: match windows instead of pixels
    - Metric ex's: SAD, SSD, NCC (see pic below)
- Global methods (energy-function minimization problem)
    - Find better solutions in exchange for more computations
    - Optimize jointly the disparity values of all pixels of each scanline (e.g. Dynamic Programming) and of the image (e.g. graph cuts)
- Other methods

Stereo Correspondence Metrics

- Sum of Absolute Differences (SAD)

$$SAD(x,y,d) = \sum_{x,y \in W} |I_l(x,y) - I_r(x,y-d)|$$

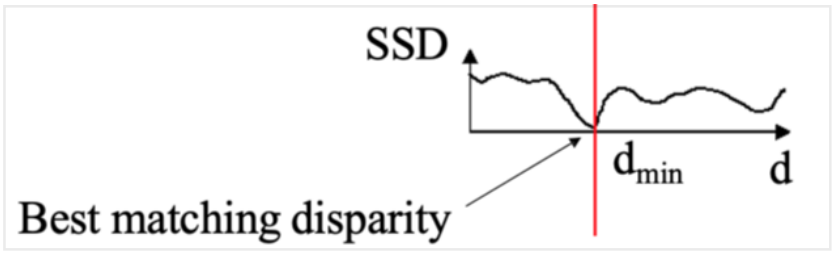
- Sum of Squared Differences (SSD)

$$SSD(x,y,d) = \sum_{x,y \in W} (I_l(x,y) - I_r(x,y-d))^2$$

- Normalized Cross-Correlation

$$NCC(x,y,d) = \frac{\sum_{x,y \in W} I_l(x,y) \cdot I_r(x,y-d)}{\sqrt{\sum_{x,y \in W} I_l^2(x,y) \cdot \sum_{x,y \in W} I_r^2(x,y-d)}}$$

- …many many more!!!



Q: exercise 2 last part - can't get to work or export
Q: project - which file correct? and did we ever get last part working? Takes too long..

Week 4:
In quaternion, w (the 4th dim) is a scalar that stores the rotation around the vector.
Projecting 3D points on the sensor:

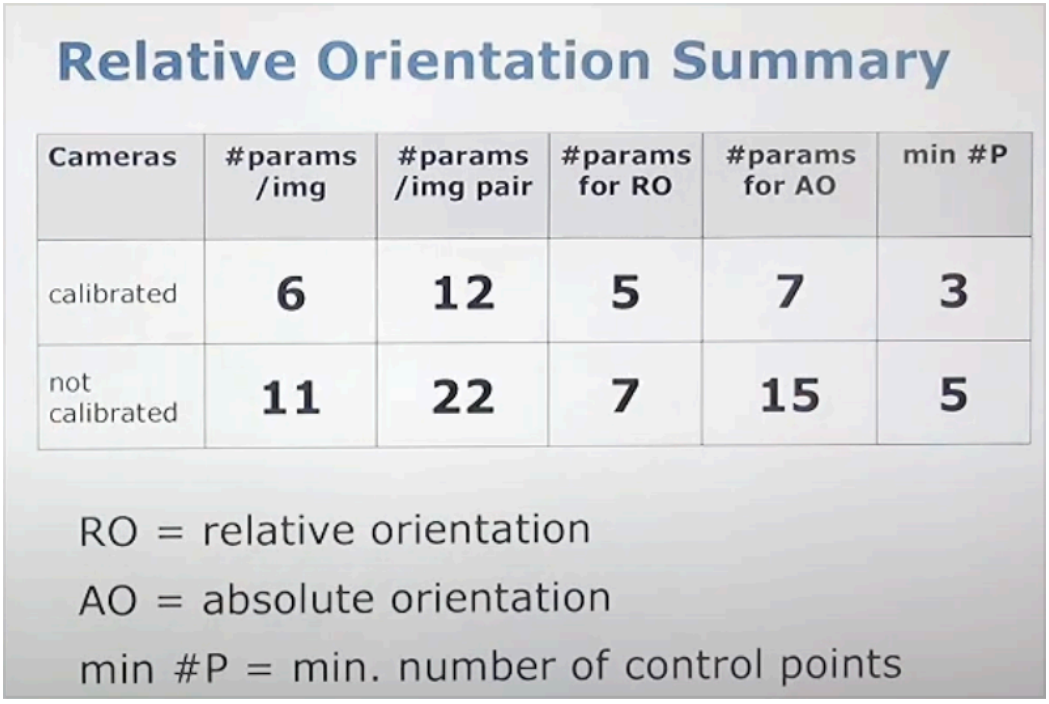
- Camera **Extrinsics** - orientation of camera wrt world coordinates
- Camera **Intrinsics** - rest of parameters

Calculating relative orientation between 2 cameras (link):
Calibrated camera

- angle preserving mapping
- Free parameters: orientation and position (3 for translation, 3 for rotation for each camera, so 12 total params)
- We can estimate 5 of the params:
  - Rotation R of 2nd camera wrt first (3 params), direction B of line connecting 2 centers (2 params)
- We cannot estimate scale or translation or rotation of first camera
- Gives us photogrammetric model

Uncalibrated camera

- straight-line preserving mapping
- Free parameters 2 cameras (each with 5 additional params) so 22 params needed in order to describe mapping
- We can estimate 7 parameters given two images



The cameras project 3d points into our image: $x' = P'x$ for camera #1, $x'' = P''x$ for camera #2

Calibration steps:

- 1. Get min 6 3d-2d correspondences

- 2. Form matrix A (AP=0)
- 3. Calculate SVD
- 4. Get lask column of V which corresponds to the values of P (the projection matrix)
- In the real world, we use predefined setup that allows us to know relative position of 3d points (the chessboard setup)

So now we have a description for the 3D point in camera frame

$$\begin{bmatrix} {}^{k}x_p & {}^{k}y_p & {}^{k}z_p & 1 \end{bmatrix}^T = \begin{bmatrix} \boldsymbol{R}_{3x3} & \boldsymbol{0}_{3x1} \\ \boldsymbol{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_{3x3} & \boldsymbol{t}_{3x1} \\ \boldsymbol{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} x_p & y_p & z_p & 1 \end{bmatrix}^T$$

rotation — translation — Homogeneous Point In World Coordinates

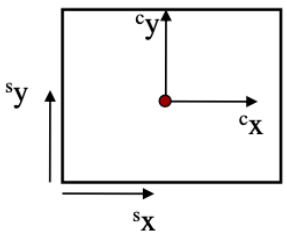
- We get to the following

$$[{}^{s}x, {}^{s}y, w]^T = \underbrace{\begin{bmatrix} -fs_x & 0 & x'_c \\ 0 & -fs_y & y'_c \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Camera Matrix / Intrinsics}} \underbrace{\begin{bmatrix} \boldsymbol{R}_{3x3} & \boldsymbol{t}_{3x1} \end{bmatrix}}_{\text{exttrinsics}} \underbrace{\begin{bmatrix} x_p & y_p & z_p & 1 \end{bmatrix}^T}_{\text{Homogeneous Point in World Coordinates}}$$

- Where W is the scale
- **This is the analytic version of the projection matrix**

- We can define the Projection Matrix as follows

$$\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$



- **This is great news for Calibration!**
- It means that our problem is a system of linear equations
- How many unknowns?
- 11: (5 + 6) Intrinsic+Extrinsic

- So for every 3D to 2D correspondence we get 2 equations:

$$(P_{31}X + P_{32}Y + P_{33}Z + P_{34})x = P_{11}X + P_{12}Y + P_{13}Z + P_{14}$$
$$(P_{31}X + P_{32}Y + P_{33}Z + P_{34})y = P_{21}X + P_{22}Y + P_{23}Z + P_{24}$$

- How many points do we need?
- 6 Points x 2 equations

– Calculate the Singular Value decomposition (SVD)

SVD : $A = UDV^T$

– Get the last column of V

$P = V_{smallest}$ *(column of V corr. to smallest singular value)*

-Reshape into the P matrix

Calibration using Homography (H):

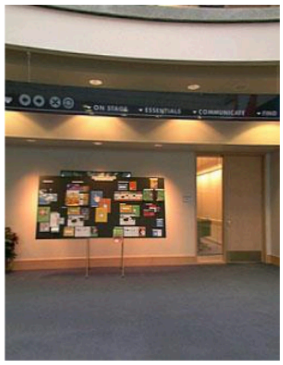
- Set z=0 so projection matrix used to project points from one plane to another loses a DOF
- 1. using a static camera, move pattern and get images
- 2. Calculate correspondence using pattern
- 3. Solve using normalized DLT (direct linear transformation)
- However, DLT known to be prone to outliers..
- Other options: Nonlinear Least Squares, RANSAC
  - 1. Choose # samples N
  - 2. Choose 4 random potential matches
  - 3. Compute H using normalized DLT
  - 4. Project points from x to x′ for each potentially matching pair
  - 5. Count points with projected distance < t
  - 6. Repeat steps 2-5 N times
  - Choose H with most inliers

Lens Distortion

- Lenses introduce distortion to the image.
- Barrel, pincushion, fisheye
- Use quartic (biquadratic) polynomial or higher order
- 1. First do DLT for projection matrix
- 2. Then form non-linear least square prob including both linear projection and non-linear distortion using Levenberg Marquardt algorithm.

$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$
$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$

Barrel



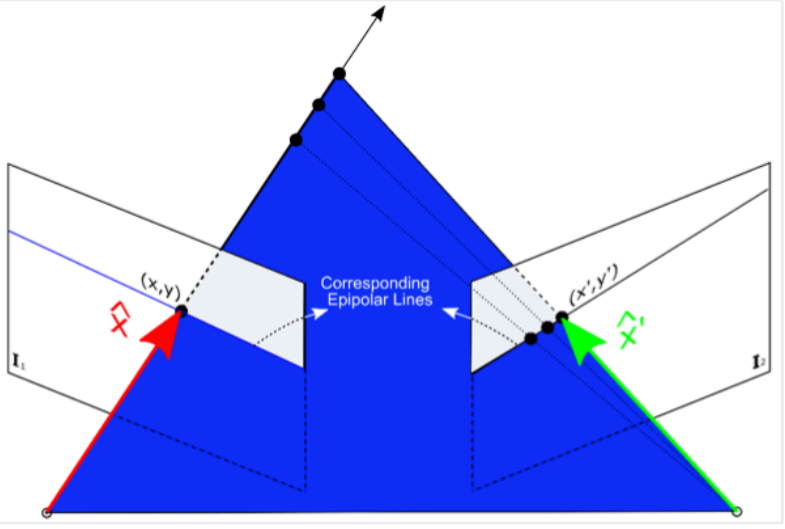
Pincushion

Fisheye

More on epipolar geometry:

- Essential Matrix:
- Assuming 2 calibrated stereo pairs, we can express x,y from the img plane to homogeneous coordinates using the inverse of the camera matrix
- The essential matrix includes the pose of the cameras WRT each other.

$$\hat{x} = K^{-1}x = X$$
$$\hat{x}' = K'^{-1}x' = X'$$



The essential matrix $E = \lfloor t \rfloor_\times R$ is a 3x3 matrix, for which:

- $E\,x'$ is the epipolar line associated with x' ($l = E\,x'$)
- $E^T x$ is the epipolar line associated with x ($l' = E^T x$)
- $E\,e' = 0$ and $E^T e = 0$
- E is singular (rank two)
- E has five degrees of freedom

- Fundamental Matrix:

- We know how to get from a homogeneous point in one camera to another but how do we get directly from one image to another?
- See slides for how to compute fundamental matrix.
- Homography (no translation) vs fundamental matrix (translation)
- The fundamental matrix projects a point in the right image frame to a point in the left image.

$$\hat{x}^T E \hat{x}' = 0$$

$$\hat{x} = K^{-1} x \quad \Rightarrow \quad x^T F x' = 0 \quad \text{with} \quad F = K^{-T} E K'^{-1}$$

$$\hat{x}' = K'^{-1} x'$$

Which is the fundamental matrix

The fundamental matrix $F = K^{-T} E K'^{-1}$ is a 3x3 matrix, for which:

- $F\,x'$ is the epipolar line associated with x'
- $F^T x$ is the epipolar line associated with x
- $F\,e' = 0$ and $F^T e = 0$
- F is singular (rank two): det(F)=0
- F has seven degrees of freedom:

To rectify an image:

1. Match some points/features
2. Calculate fundamental matrix (using coordinates of matched points)
3. Calculate the rectified image

Thus both the **Essential** and **Fundamental matrices** completely describe the geometric relationship between corresponding points of a stereo pair of cameras. The only difference between the two is that the former deals with calibrated cameras, while the latter deals with uncalibrated cameras.

Week 5:

3D Point Clouds

- **Pose**: the transformation (translation + rotation) needed to map one point cloud (model) to another point cloud (model) of the same (fully or partially) object or scene. i.e. “determining a camera’s position relative to a known 3D object or scene” (Szelinski)

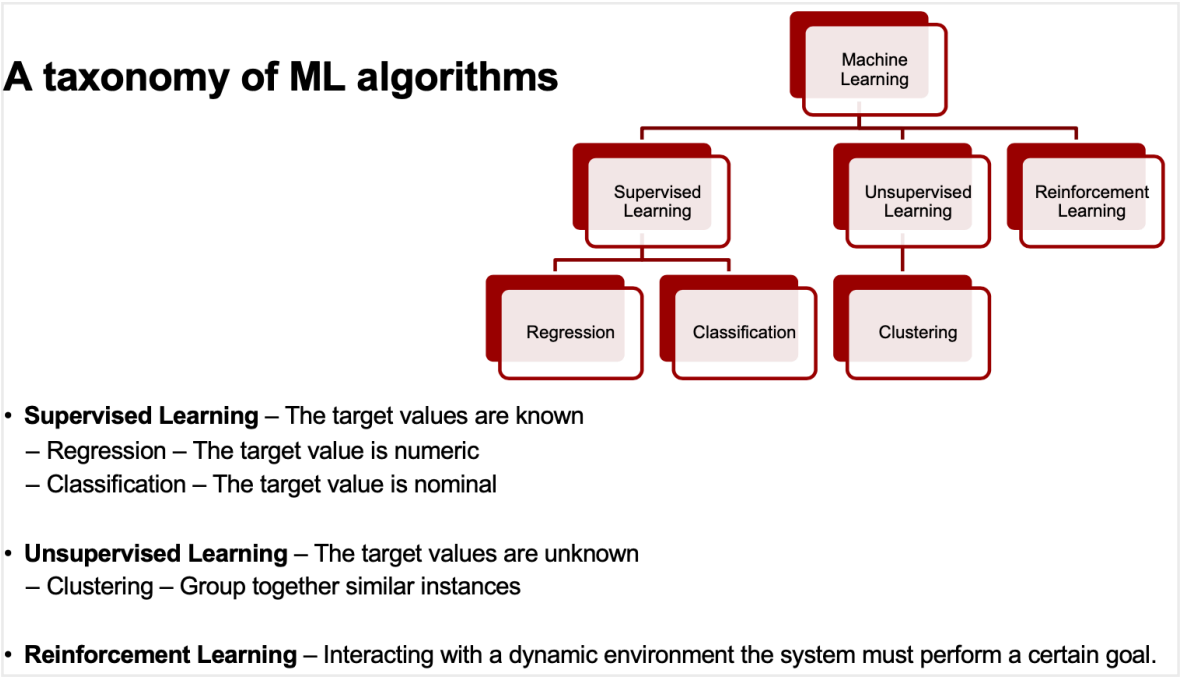
Point Cloud Registration

- Global alignment - the 2 models are roughly aligned
- Local alignment - starting from a rough initial alignment, find the exact precise alignment
- **Local alignment**:
  - Generally 2 steps are needed to register 2 given point clouds of the same object (fully or partially)

- Iterative Closest Point (**ICP**) algo:
  - Consider 2 models (point clouds) of the same object (fully or partially) that are almost aligned. What is the difference of their pose? i.e. What is the transformation that can fully align them?
  - 1. For each object point p in model 1, find the nearest point q in model 2.
    - If model 1 has X points, model 2 has Y points, then calculate XY distances.
    - Use k-d trees (k-dimensional) to speed up search.
  - 2. Use all pairs (p, q) to estimate the transformation from model 1 to model 2.
    - Kabsch algo / Procrustes Analysis: translate centroids of both models to origin, compute centroids (c_p, c_q), subtract from each point coords the coords of its corresponding centroid (p′ = p-c_p, same for q), compute com matrix (Cpq = sum(p′*trans(q′))), compute optimal rotation (calc SVD of Cpq and rot matrix R=U*trans(V)), compute optimal translation T=c_q-R*c_p.
  - 3. Apply the transformation to the points of model 1.
  - 4. Repeat steps 1-3 until convergence or stop criterion is met.
- **Global Alignment:**
  - Cannot use ICP if initial poses are too different.
  - Instead, find 3D feature descriptors and match them.
    - **Spin Images**: “spin” a discretized 2D grid around the surface normal of a point, accumulate neighboring pixels in the grid bin while spinning, the descriptor is the 2D grid (img) where each elem (pixel) contains the # accumulated points.
    - Point Feature Histograms (**PFH**): calculate 3 angle values for all pairs of points within a radius r from considered point (maybe also distance) and bin the set of all triplets/ quadruplets in a histogram.
    - Fast Point Feature Histograms (**FPFH**): same as PFH algo but within k-d tree (so reduces computational complexity)
  - Matching 3D features generally results in many false-matches.
  - Sol’n: RANSAC!
    - Robust to outliers.
    - 1. Randomly choose 3 pairs matched points
    - 2. Estimate the relative pose (kabsch / procrustes)
    - 3. Apply the transformation and assess its “validity”
    - 4. Keep transformation with most inliers.
    - 5. Repeat random sampling.

Week 6:

Machine Learning for 3D point clouds



Regression:

- Supervise learning
- Target value is numeric
- Tries to assign correct significance (weights) to input variables and make a weighted linear combination of them to predict output variables.
- Finds an eqn f that models the linear relationship bet input and output: y = f + e
- Performance evaluated by error function
  - Most used it Mean Squared Error (**MSE**) = (1/N) sum(t-y)^2 where t is true value of learned output and y is predicted value.
- Goal is to minimize the MSE.
- Polynomial regression for when mapping is not linear.

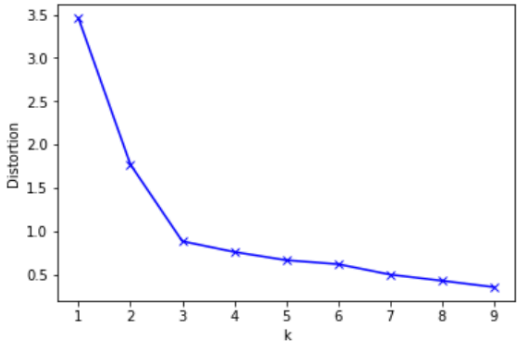
Clustering:

- Unsupervised learning
- Target values are unknown
- Group together similar instances
- Finds underlying structure/partitions of points, ie clusters
- K-means
  - Minimizes sum of squared euclidean distances (distortion) between points and their randomly assigned cluster centers, then updates centers
  - Required to know number of clusters k (use elbow method to determine best k, or silhouette analysis)
  - Iterative process

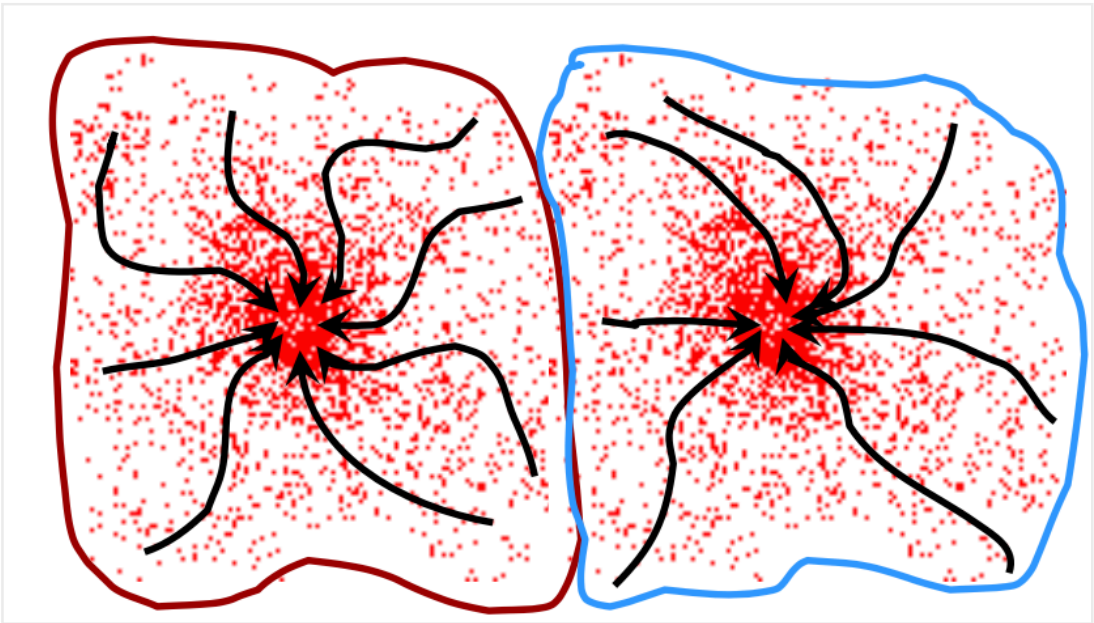
How to define *k* ?

– **Elbow method**

- Run k-means for several *k* and calculate distortion for each *k*
  - » Distortion: sum of squared distances of each point to the center of the closest cluster
- Plot distortion vs *k*
- Look for k where the curve stops decreasing rapidly



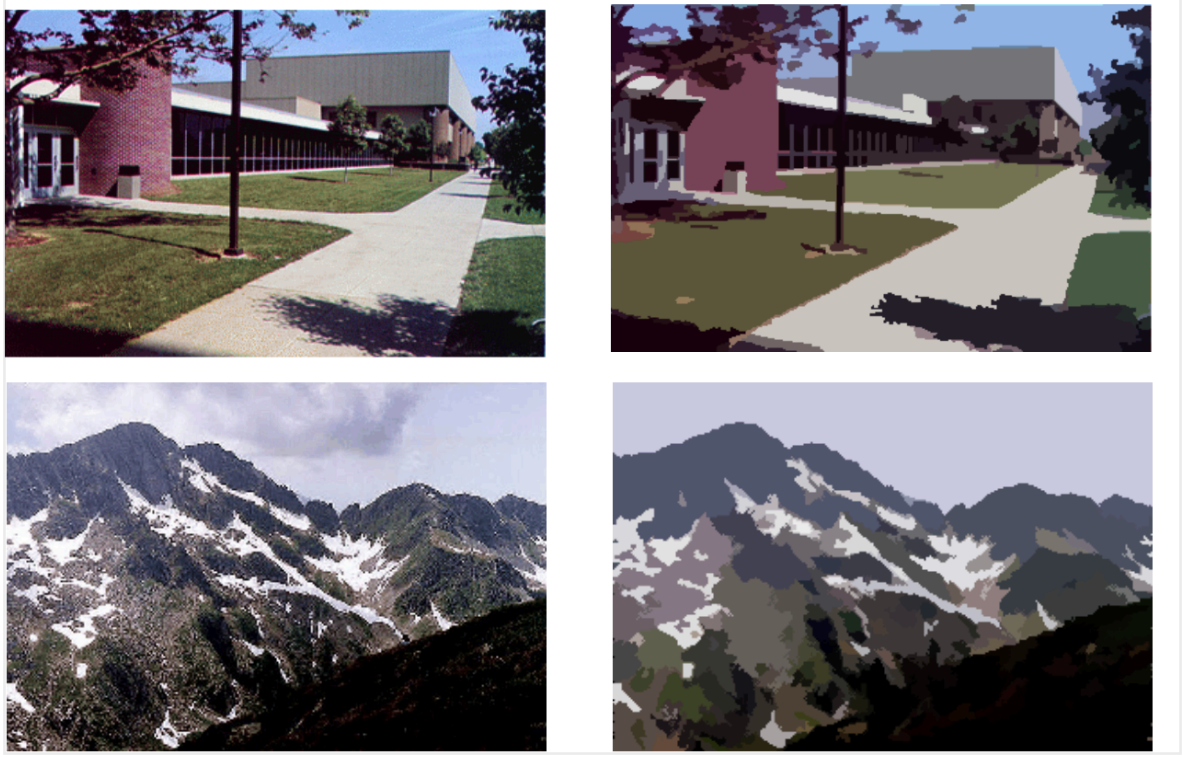
- Mean Shift
  - Finds the maxima (modes) of a density function given discrete data sampled from that function.
  - Each cluster defined by the radius ("bandwidth" h) of its region and kernel function K.
  - Only unique clusters considered (so we don't need to set number of clusters).
  - **Attraction basin**: region for which all trajectories lead to same mode
  - **Cluster**: all data points in attraction basin of a mode



The mean shift algorithm seeks *modes* of the given set of points

1. Choose kernel and bandwidth
2. For each point:
   a) Center a window on that point
   b) Compute the mean of the data in the search window
   c) Center the search window at the new mean location
   d) Repeat (b,c) until convergence
3. Assign points that lead to nearby modes to the same cluster

Mean shift segmentation results

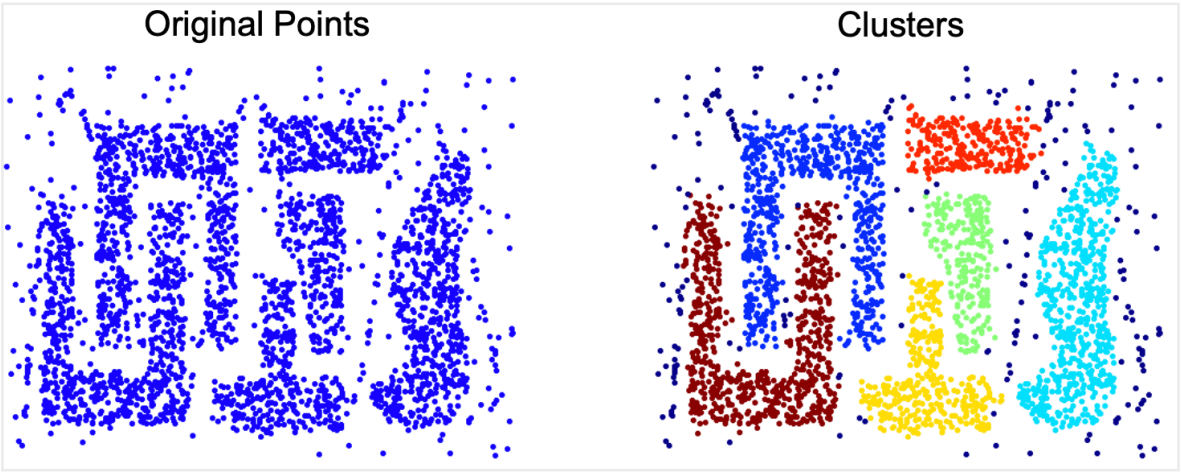


- Density-based spatial clustering of applications with noise (**DBSCAN**)
  - Partition points into dense regions separated by not-so-dense regions.
  - A cluster is defined as a maximal set of density-connected points (of arbitrary shape)
  - Density at point p is the #points within a circle/sphere of radius e (dense if contains at least MinPts points)
  - **Core point** has more than MinPts within e
  - **Border point** has fewer than MinPts within e but is in the neighborhood of a core point
  - **Noise point** otherwise
  - **Density edge** is a line between 2 core points if they are within distance e
  - A point p is **density-connected** to a point q if there's a path of

edges from p to q.
- A cluster contains all points that can be reached by following a sequence of density-connected core points and border points that are closest to one of the above-clustered core points.

DBSCAN algorithm

1. Label points as core, border and noise
2. Eliminate noise points
3. For every core point p that has not been assigned to a cluster
   – Create a new cluster with the point p and all the points that are density-connected to p.
4. Assign border points to the cluster of the closest core point.



- Hierarchical Clustering
  - Builds a hierarchy of clusters (clusters that consist of other smaller clusters)
  - **Agglomerative** = bottom-up approach (each point starts in its own cluster and pairs of clusters are merged as one moves up the hierarchy)
    - Based only on distance (similarity) between points

Agglomerative / Bottom-up hierarchical clustering
is based only on distance (similarity) between the points.

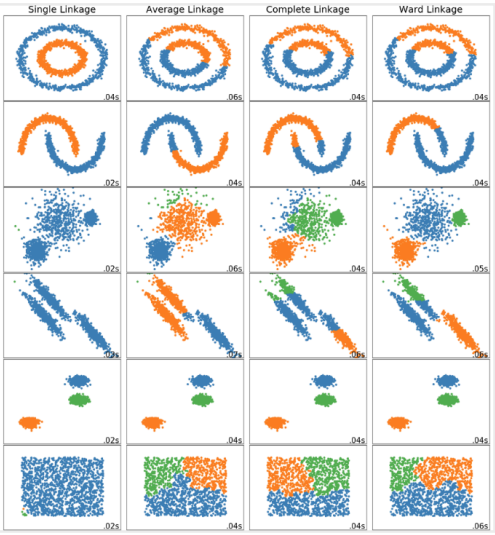
Algorithm steps:

1. Start by assigning each point to its own cluster, obtaining as many clusters as points.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster.
3. Compute distances (similarities) between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster.

How is distance (similarity) between clusters assessed?

Linkage methods:

- **Single-link**: the distance between two clusters is equal to the minimum distance from any member of one cluster to any member of the other cluster.
- **Complete-link**: the distance between two clusters is equal to the maximum distance from any member of one cluster to any member of the other cluster.
- **Average link**: the distance between two clusters is equal to the average distance from any member of one cluster to any member of the other cluster.
- **Ward-link:** the distance between two clusters is equal to the sum of squared differences within any member of one cluster to any member of the other cluster.



  - **Divisive** = top-down approach (all points start in one cluster and split recursively as one moves down the hierarchy)

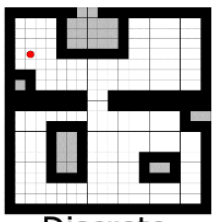
Week 7:
Histogram Filter
Kalman Filter

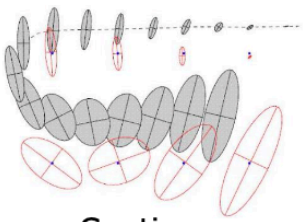
State Estimation

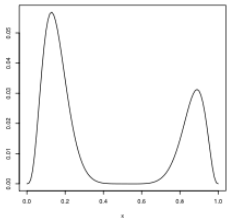
- Given a state vector of a system, estimate over time the state using input of external sensors

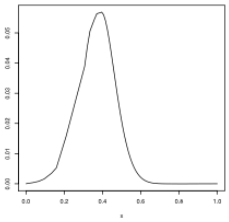
Continuous *vs* Discrete State
Unimodal *vs* Multimodal Distribution

- State:
- Distribution


Discrete

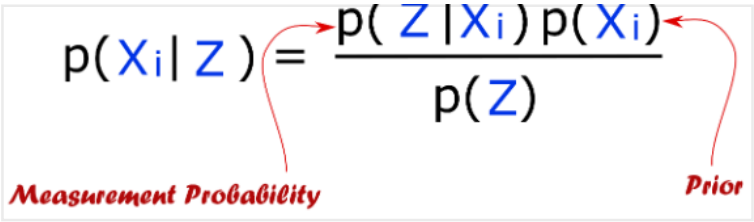

Continuous


Multimodal


Unimodal

- Localization is a sense/move cycle
- Global localization:
  - Belief -> Probability

  - Measurements -> Multiplication followed by normalization
  - Moving -> Convolution
- Bayes Rule



- Total Probability

$$\Pr(A) = \sum_{n} \Pr(A \mid B_n)\,\Pr(B_n)$$

Kalman uses gaussian probabilities and variance is now a covariance matrix.

Prediction (Ingredients):
- X: State Vector (Including our Prior Info)
- P: Uncertainty Covariance (Incl. Prior Info)
- F: State Transition Matrix (we just discussed it)
- u: External Motion (E.g. Deceleration from car)

Measurement Update (Ingredients): :
- Z: Measurement
- H: Measurement Matrix
- R: Measurement Noise
- y: Error
- K: Gain
- I : Identity Matrix

Recipe

$$X' = F X + u$$
$$P' = F \cdot P \cdot F^T$$
$$y = Z - H \cdot X$$
$$S = H \cdot P \cdot H^T + R$$
$$K = P \cdot H^T \cdot S^{-1}$$
$$X' = X + K \cdot Y$$
$$P' = (I - K \cdot H) \cdot P$$

Week 8:

Classification
- Supervised learning
- Output is usually decision boundaries that separate the classes

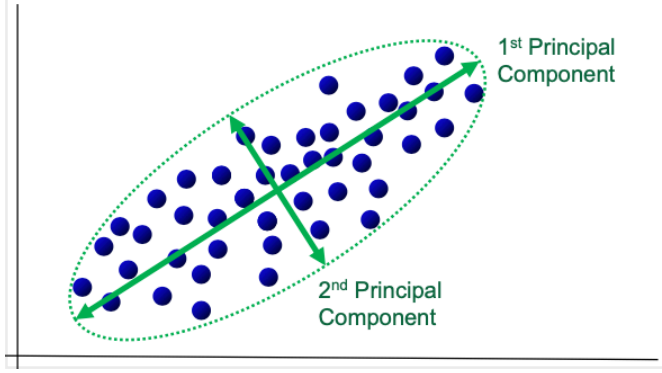
Dimensionality Reduction
- Principal Component Analysis (**PCA**)
  - Reduces high dimensionality.
  - Creates new features that are linear combinations of original features.
  - New features are orthogonal to each other.
  - Keep new features that account for large amount of variance in original dataset.
  - Re-base the dataset's coordinate system in a new space defined by its lines of greatest variance.

- Ex: detecting faces (facial recognition)

- **Principal Component Analysis (PCA)**

1. Center the input data
2. Calculate Covariance Matrix
   $C = \frac{1}{N-1} X^T X$
3. Compute eigenvectors & eigenvalues of the Covariance Matrix
   - The first principal component is the eigenvector of the covariance matrix that has the largest eigenvalue
     - » This vector points towards the direction of the largest variance of the data
     - » The corresponding eigenvalue defines the magnitude of this vector
   - The second largest eigenvector is orthogonal to the largest eigenvector, and points into the direction of the second largest spread of the data.
4. Sort the eigenvectors according to their eigenvalues
5. Calculate the variance score (significance).
6. Keep eigenvectors that explain most (e.g. 95%) variance / remove the rest! (← dimensionality reduction)
7. Project instances to eigenvectors y.
   $y = W^t x$

- **Eigenvector:** points to the direction of variance.
- **Eigenvalues:** shows how much of the total variance is explained by the corresponding eigenvector.
- Keep the eigenvectors that explain most (e.g. 95%) of the variance.



Eigenfaces

- Then an image of our dataset becomes:

  Image = Mean Image + c_1*(1st Eigenface) + c_2*(2nd Eigenface) + …

- So, these coefficients c_1, c_2, … c_{64} represent the face image!!

- We can perform Face Recognition
  - For a new face image
    - We can calculate its 64 coefficients
    - Find the closest training face (most similar coefficients) in the 64-dimensional space (the most similar face of my training dataset)
    - **Classify** the new face as this most similar training face.

Classifiers

- K-Nearest Neighbors (**KNN**)
  - A geometric method for classification based on the assumption

that neighbor instances belong to the same class.

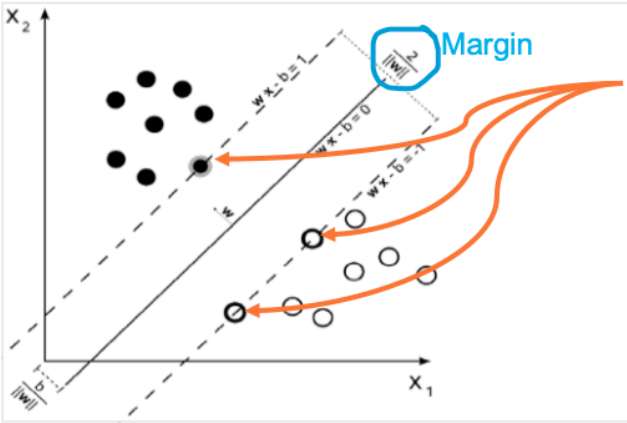
For classifying a new instance x_* :

- we calculate its distance from all the other instances.
- Then we select the k-nearest instances.
- The probability of the class is given by:

$$p(C_j|x_*)=\frac{k_j}{K}$$

where k_j is the number of nearest neighbors that belong to class j, and K is the total number of nearest neighbors.

- Support Vector Machines (**SVM**)
  - A geometric method for classification that finds a line/surface in feature space that separates the classes
  - The linear decision boundary satisfies 2 criteria:
    - 1. The distance between the instances and the boundary is as large as possible (max margin)
    - 2. The instances are classified as good as possible
  - Thus, the decision boundary is a hyperplane h(x) defined as **h(x) = trans(w) * x + b** and the goal is to find the w that satisfy 1 and 2.
  - Maximize the margin 2/||w|| while correctly classifying all training data points.



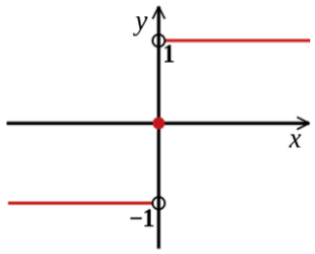
It is only these 3 points that define our separation boundary!

These 3 points are called: Support Vectors

The classification rule is expressed as: $f(x_*)=sgn(h(x_*))$

The sign function is defined as:

$$\operatorname{sgn}(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$



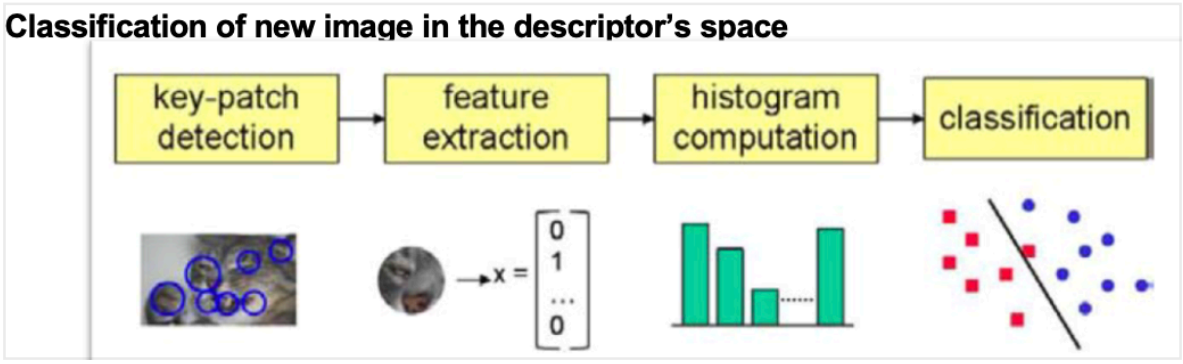
Thus, a new instance is labeled with 1 if it is located above the boundary and with -1 otherwise.

- If data is above 2D or not linearly separable, we can use Multi-class SVMs and kernels (linear, polynomial, gaussian like radial

basis function RBF)

Bag of Words

- A vector of occurrence counts of a vocabulary of local image features.



Object Detection:

- when we know what we are looking for

Instance Recognition:

- when we have a specific rigid object we are trying to recognize

Category/Class Recognition:

- when we want to recognize instances of extremely varied classes (e.g. animals or furniture)

Latest methods rely on deep neural networks

- e.g. YOLO and Faster R-CNN for object detection
- e.g. PointNet for classification in 3D point clouds

Week 9:

Visual Odometry (VO)

- Has been proposed as an alternative to wheel odometry.
- Concerns the use of cameras to estimate the Pose (position and orientation) of a mobile system by observing the apparent motion of the "static" world.
- Assumes a static world where the only moving object is the mobile system.
- Does not provide a map of the environment.
- It uses previous states of the world to improve its accuracy (SLAM simultaneous localization and mapping).
- Estimates are usually combined with other sensors (GPS, IMU, Laser, Wheel odometry; VINS, VIO visual inertial odometry)
- 3 main variants: 3D-3D, 2D-2D, 3D-2D
  - 2D-2D and 3D-2D are better than 3D-3D
- Accumulates transformations T_k from frame f_k-1 to frame f_k over time, providing full trajectory C_0:n
  - Can optimize over multiple frames using **bundle adjustment**
    - Similar to pose-optimization but also optimizes 3D points
    - Use levenberg-marquadt

        - "Given a set of images depicting a number of 3D points from different viewpoints, bundle adjustment can be defined as the problem of simultaneously refining the 3D coordinates describing the scene geometry, the parameters of the relative motion, and the optical characteristics of the camera(s)." (Wikipedia)
    - Overall: uses consecutive camera frames (frame-to-frame approach) to calculate the relative pose (position and orientation) of the cameras.

Levenberg-Marquadt

- Mixture of Gauss-Newton and Gradient descent.
- Acts like Gauss-Newton when close to the minimum (quadratic region)
- Gradient descent when improvement is difficult.
- Depends on a parameter λ which
  1. Controls the mixture of Gauss-Newton and Gradient Descent
  2. Controls the step-length.

Visual SLAM

- Uses state estimation to exploit additional constraints and re-observations of the same areas (loop-closures) to optimize the localization.

Motion Tracking

- Two main approaches:
    - 1. Feature based
        - Calculate feature on both images, match among features or do block matching around initial point (for small motion)
    - 2. Optic Flow
        - estimate apparent motion
        - Lukas Kanade

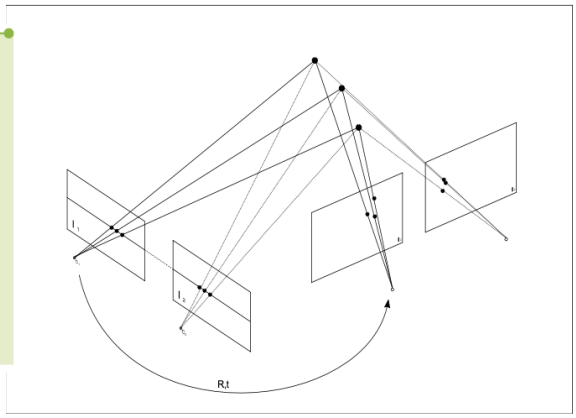
Orientation

- Rotation Matrix
  - Positives (The king of Orientation)
    - Unique, no gimbal lock
  - Negatives:
    - No perturbation, interpolation, unintuitive
- Euler anglers
  - Positives
    - Minimal representation, intuitive
  - Negatives
    - Gimbal Lock, non commutative
- Axis Angle:
  - Positives
    - No gimbal lock, minimal representation, nice for perturbation, linear mapping to rotation matrix
  - Negative
    - Not linear “scaling” wrt magnitude
  - Exponential coordinates
- Quaternions
  - Positives
    - all the axis angle ones, smooth trajectory
  - Negatives
    - No direct geometric representation

Visual Odometry 3D – to 3D

Algorithm 2. VO from 3-D-to-3-D correspondences.

1) Capture two stereo image pairs $I_{l,k-1}, I_{r,k-1}$ and $I_{l,k}, I_{r,k}$
2) Extract and match features between $I_{l,k-1}$ and $I_{l,k}$
3) Triangulate matched features for each stereo pair
4) Compute T_k from 3-D features X_{k-1} and X_k
5) Concatenate transformation by computing $C_k = C_{k-1}T_k$
6) Repeat from 1).

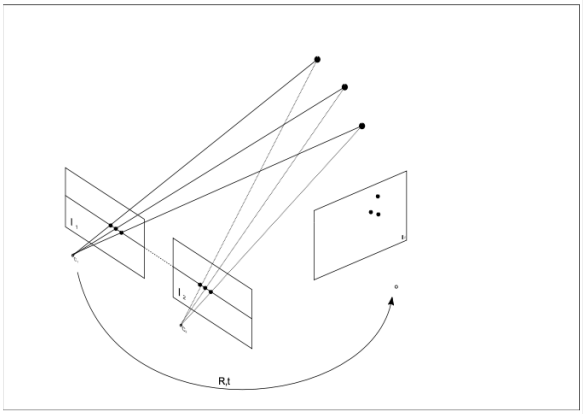


3D-3D

Visual Odometry 3D – to 2D

Algorithm 3. VO from 3-D-to-2-D Correspondences.

1) Do only once:
1.1) Capture two frames I_{k-2}, I_{k-1}
1.2) Extract and match features between them
1.3) Triangulate features from I_{k-2}, I_{k-1}
2) Do at each iteration:
2.1) Capture new frame I_k
2.2) Extract features and match with previous frame I_{k-1}
2.3) Compute camera pose (PnP) from 3-D-to-2-D matches
2.4) Triangulate all new feature matches between I_k and I_{k-1}
2.5) Iterate from 2.1).

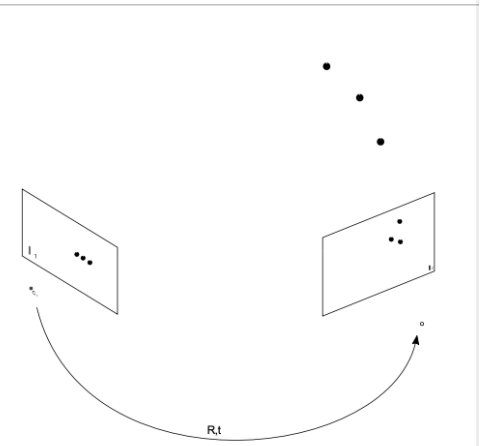


3D-2D

Visual Odometry 2D – to 2D

Algorithm 1. VO from 2-D-to-2-D correspondences.

1) Capture new frame I_k
2) Extract and match features between I_{k-1} and I_k
3) Compute essential matrix for image pair I_{k-1}, I_k
4) Decompose essential matrix into R_k and t_k, and form T_k
5) Compute relative scale and rescale t_k accordingly
6) Concatenate transformation by computing $C_k = C_{k-1}T_k$
7) Repeat from 1).



Week 10:

Visual SLAM (simultaneous localization & mapping)

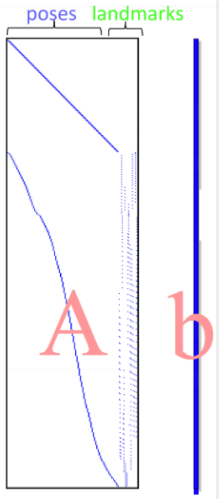
- Pose-landmark graph-slam
- Graph-based slam

Bundle Adjustment

- "Minimizing the reproduction error between the image locations of observed and predicted image points, which is expressed as the sum of squares of a large number of nonlinear, real-valued functions."
- High computational cost

Bundle Adjustment – Comments

- Bundle adjustment (and graph optimization) is the backbone of all SLAM algorithms
- Keep in mind that:
  - We need to provide the Jacobian of the projection
  - We usually provide a covariance matrix (See the linear case for uncertainty)
  - It is solved using Levenberg-Marquadt
  - There are a lot of computational issues which are overcome exploiting the sparsity of the function AX=b (see least squares)
    Look at the following A and b Matrices
    This solution is called **Sparse Bundle Adjustment!**



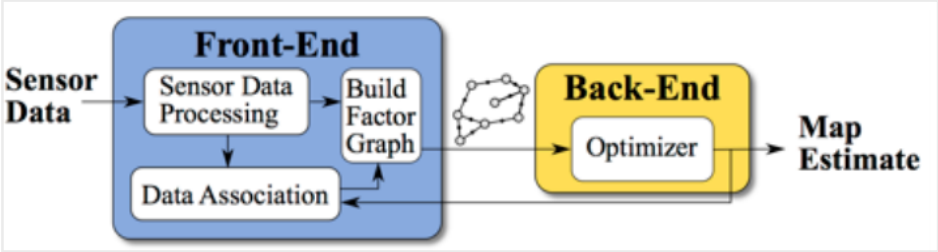
point i by a vector $\mathbf{b}_i$. Bundle adjustment minimizes the total reprojection error with respect to all 3D point and camera parameters, specifically

$$\min_{\mathbf{a}_j,\,\mathbf{b}_i} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij}\, d(\mathbf{Q}(\mathbf{a}_j,\,\mathbf{b}_i),\,\mathbf{x}_{ij})^2,$$

where $\mathbf{Q}(\mathbf{a}_j,\,\mathbf{b}_i)$ is the predicted projection of point i on image j and $d(\mathbf{x},\,\mathbf{y})$ denotes the Euclidean distance between the image points represented by vectors $\mathbf{x}$ and $\mathbf{y}$. Because the minimum is computed over

Most recent visual SLAM methods are split into 2 parts:

- **Frontend**: where the raw data are converted into pose graphs and loop constraints.
- **Backend**: where, given a graph with constraints, the new pose of the robot is calculated as well as the surrounding map points.



Recent Visual Slam Solutions – Common Architecture

- Front End
  - Data Association
    - Frame to Frame
    - Multi-frame
    - Loop Closure Detection
  - Geometric Initialization
    - Pose Estimation
    - Landmark Triangulation
  - System Formation
    - Observation Matrix
    - Covariance Matrix
    - Graph Generation and Update
- Back End
  - Filter-Based State Estimation
    - Extended Kalman Filter
    - Particle Filters
  - Least squares optimization
    - Bundle Adjustment
    - Graph Optimization
      - Key Frame

Keyframing:

- The idea of identifying and describing some of the frames to be used for graph optimization (moving towards realtime operation)

Bag-of-words for robust loop closure.

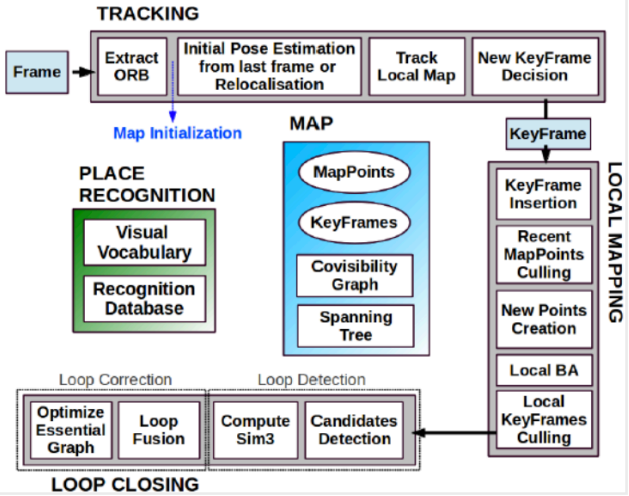
ORBSLAM algorithm

- One of the most well performing opensource implementations of visual slam.
- 3 parallel threads:
  - Tracking
  - Local mapping

- Loop closing

ORB SLAM

- The ORBSLAM algorithm is one of the most well performing opensource implementations of visual slam.
- Three parallel threads:
  - tracking,
    - localizing the camera with every frame and deciding when to insert a new keyframe
  - local mapping
    - Processes new keyframes and performs local BA to achieve an optimal reconstruction in the surroundings of the camera
  - loop closing
    - The loop closing searches for loops with every new keyframe
    - Essential Graph



OTHER:
ICP: provides a rigid transformation between 2 point clouds

In mathematics, a **rigid transformation** (also called Euclidean transformation or Euclidean isometry) is a geometric transformation of a Euclidean space that preserves the Euclidean distance between every pair of points. The rigid transformations include rotations, translations, reflections, or their combination.

An **affine transformation** is any transformation that preserves collinearity (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation).