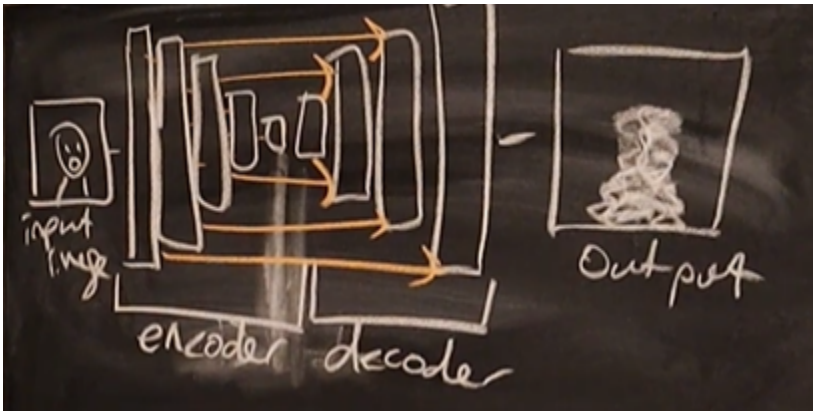
January 17th - Deep Learning

Most neural networks for images can be abstracted into a function. On a input image, f(image) returns if it's a cat/dog, or the alpha mette, etc.

This is a very hard task, so we break up f into composite simple functions to achieve this task where f(image) = $f_1(f_2(f_3\ldots(\text{image})))$. This is imitating a simplified neuron. So define simple functions as
f(image) = $f_1(f_2(f_3\ldots f_N(\text{image})))$. We can measure our image in functions of filters, image into filters where the specific kernel weights are learned afterward. If the kernel matches what is going on in the image very well, it will give a high response.
We can also rectify layers by putting all negative/non activated values to 0. (think of abs).
Also convolution layers.

We can also have a compression within the functions, with a encoder/decoder combo. Where the functions are compression layers (possibly lossy) to a smaller representation, operated on and then decoded for our output. We can save lossy compressions with a skip connection where we skip compression layers and forward some applied filters right to their corresponding decoder layer.



Here, the orange arrows are skip connections.

In supervised learning, we have an input that we already know the output answer to. Here, we train and tweak the network based on the output it gives and how close it is to the ground truth.

We measure this with a loss function, where the function measures how incorrect it is and where in the filters it's wrong. To measure how to train our model we take gradient of loss function to find min of loss and train toward that. (high-level idea)
Another high level idea is think of back-propagation as taking derivative of $f(\text{image}) = f_1(f_2(f_3 \ldots f_N(\text{image})))$ and applying chain rule in each composite function in LHS, where each derivative's iteration's loss function is tweaked at it's respective layer (function).

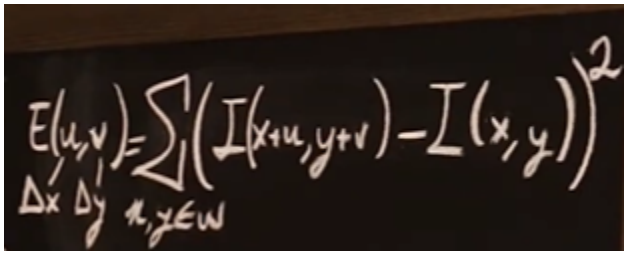
We can also get an alpha mapping objective where 0 is background and 1 is foreground. We can split this into a trimap where black is background, gray are edges so they could be either and white is foreground.

January 18th - Signals and frequency

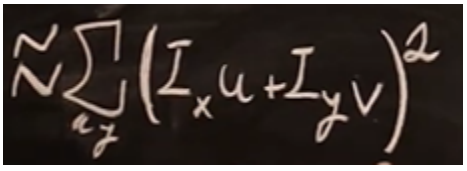
Essentially, any box wave (0/1 switch) can be approximated by different combinations of sine waves. These fourier decomposition waves are referred to as the frequency.

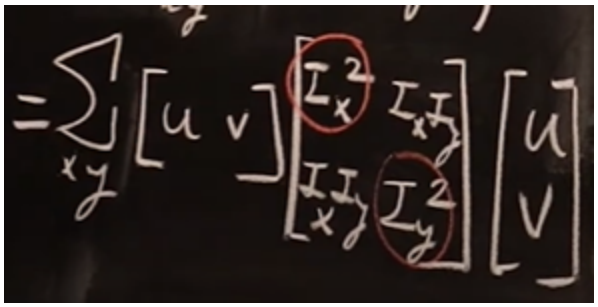
January 25th - Harris corner detection

Say we have a window of constant size that moves along a picture. This movement is described by E(u,v), where u is the Δx and v is the Δy.
Then the difference in the location can be computed with a sum of squares difference.



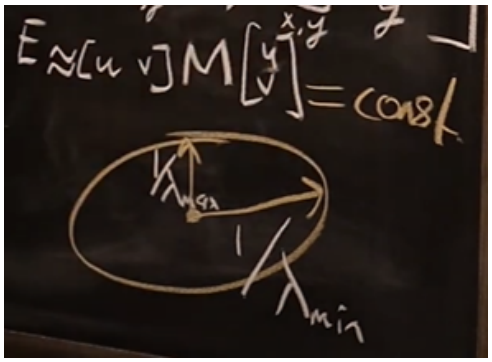
This E is very high, we know this is a descriptive point.
We can keep E very small and use the taylor series representation of L(x+u, y+v). This is approximately $L(x,y) + L_x u + L_y v$. The L(x,y) - L(x,y) will cancel out and we're left with



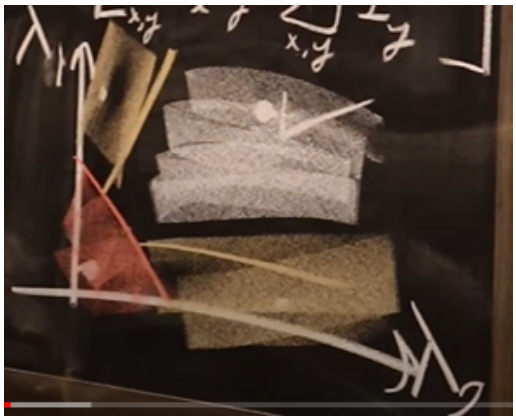


Written in matrix form. We can define this, this is our second moment matrix where M is the center 2x2 matrix.

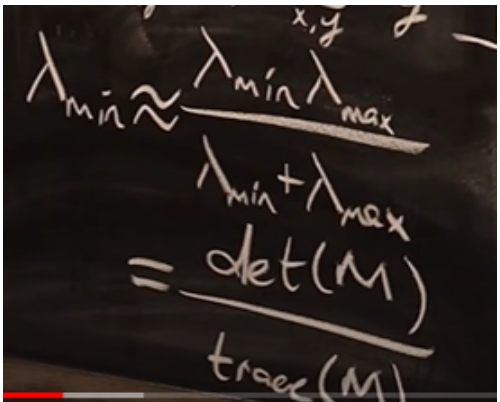
What's interesting is when this is equal to a constant, this represents an ellipsoid where the shape of the ellipse is determined by the eigenvalues of M.

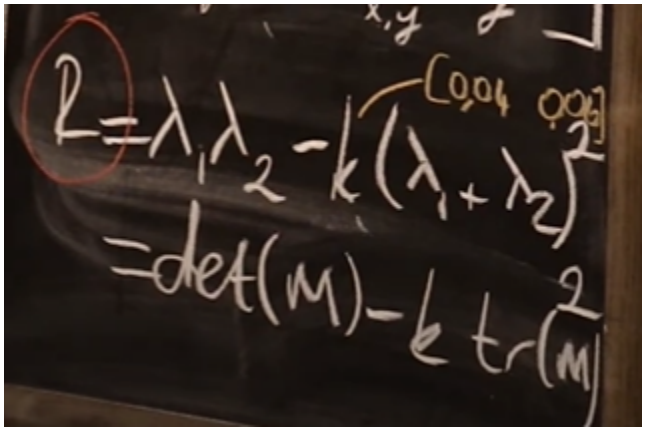


When one of the gradients is zero, all entries will become 0 meaning smallest eigenvalue 0, the ellipse will be infinitely long.
Having a gradient of zero means you can only measure in one direction, but we don't want that we essentially want as small an ellipse as possible to have the strongest edge possible.



We can do this by plotting the two eigenvalues against each other. If both are very high we probably have a corner, if both are very low its probably just a flat part of the image. If one of them is low and the other is high, we have an edge.
Red is flat
Yellow is edge
White is corner





This is how we benchmark the eigenvalues, and much like canny edge detection we need to threshold these.
We can also multiply M by a scalar weight w to get a more robust calculation, this is like applying a gaussian filter.

January 25th - Matching and invariance

These identifiable features should be independent of certain geometric properties, meaning rotating, scaling or changing intensity shouldn't change the features.

2) Matching
Invariant: Image is transformed and corner location does not change.
Equivariance: Features should be detected in corresponding locations of the two transformed versions of same image
Discriminability: Descriptor should be highly unique for each point

The Harris detector corner location is equivariant. When we change the location of a corner we expect it to do the same thing. Same thing goes for rotation of image. Second moment ellipse rotates but its shape (eigenvalues) stays the same.
Intensity change is a different story as our thresholds are affected, meaning harris corner detection is only partially invariant to intensity change.
Because scaling can scale a group of edges to a corner, harris corner detection is neither invariant to equivariant to scaling.

This is the motivation behind our scale invariant detection, where our neighbourhood comparison should be an optimal size. We do this by varying our neighbourhood size and taking maximum corner response to give us optimal neighbourhood.

We can do this with a gaussian pyramid where we use a fixed neighbourhood size but scale down image using gaussian. We will also use a laplacian of gaussian which detects "blobs".
DoG is the difference of gaussian kernel, not the derivative of gaussian kernel.
Each image will have a dominant orientation for the edges, which is given by the largest eigenvalue of our Harris matrix or the orientation of the gradient.

MOPS?

Scale Invariant Feature Transform (SIFT): Look at gradient direction of a square window around a pixel, and see which direction appears the most. This will identify features. SIFT is very robust, it can handle changes in brightness and viewpoint, and it is very fast.

When we have found our features in I_1 and described them in an invariant way, we can match them to possible correlations in I_2.
1) Define distance function that compares two descriptors
2) Test all features in the I_2, find the one with min distance.

We can trivally do 1) by measuring distance between the features in I_1 & I_2, but this doesn't work if we have many repeated features in the image. So a better way is to use ratio of distance.

How to define the difference between two features f_1, f_2?

- Better approach: ratio distance = $||f_1 - f_2|| / ||f_1 - f_2'||$
  - f_2 is best SSD match to f_1 in I_2
  - f_2' is 2nd best SSD match to f_1 in I_2
  - gives large values for ambiguous matches

We can measure performance of feature matcher by comparing true positives and false positives

January 31st - Transformation and image alignment

We already know how to rotate and scale an image down or up. Now, we will learn about image warping where we tie two images of the same place taken from a different angle.

Image filtering: change range of image
Image warping: change domain of range

Parametric global warping: Transformations, rotations and scaling.

Transformation: get pixel of image, apply transform operator.
A global transform operator is the same for any point p and can be described with just a few numbers.

$$\mathbf{p}' = \mathbf{T}\mathbf{p} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scaling: Uniform scaling by S

$$\mathbf{S} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

We also have transformations:
rotating:

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

What is the inverse?
For rotations:
$\mathbf{R}^{-1} = \mathbf{R}^T$

Which transformations can we represent with a 2x2 matrix?

2D mirror about Y axis?

$$\begin{aligned} x' &= -x \\ y' &= y \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

2D mirror across line y = x?

$$\begin{aligned} x' &= y \\ y' &= x \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

But translation cannot be a 2x2 matrix. 2D linear transformations are combinations of scale, rotation, shear and mirroring.
Linear transformations also: map origin to origin, lines to lines, parallel lines

remain parallel, ratios are preserved and are closed under composition.

In order to do transformations, we can define homogeneous coordinates. Basically a pixel that is meaningless. For each pixel located by (x,y), add one more coordinate (x, y, 1). If we end up with homogeneous coordinates that don't have 1 at the end, we can convert it back from (x, y, w) → (x/w, y/w).

Homogenous coordinates essentially a line from each pixel.

- Solution: homogeneous coordinates to the rescue

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Any transformation with a 0 0 1 in the bottom row are called affine transformations. Affine transformations have six degrees of freedom.

(a b c
d e f
0 0 1)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D *in-plane* rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear

Difference from affine transformations compared to linear transformations, origin doesn't map to origin always now.

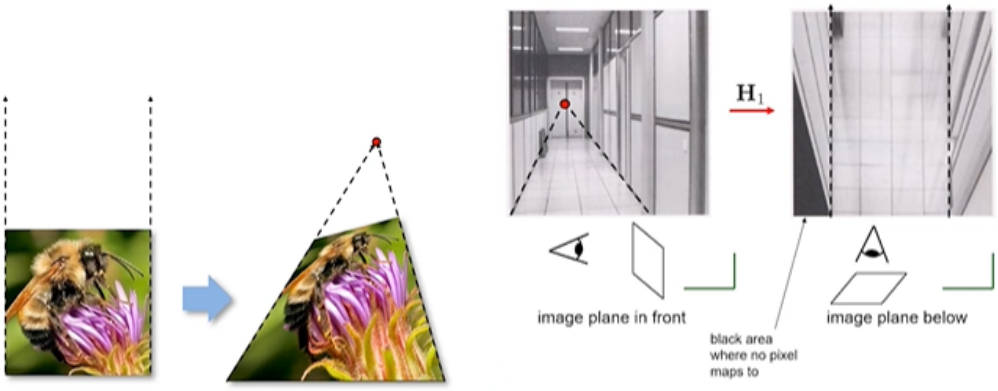
In order to stitch images together, we need to change the last row aswell, with homographic transformation matrix. 8 degrees of freedom with the last element 3,3 being 1.

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ gx + hy + 1 \end{bmatrix}$$

What happens when the denominator is 0?

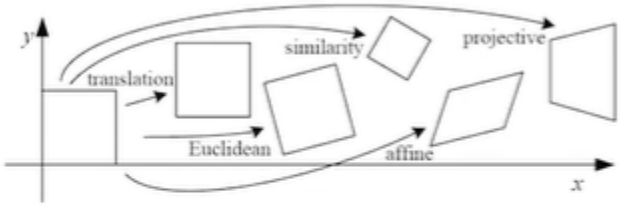
$$\sim \begin{bmatrix} \frac{ax+by+c}{gx+hy+1} \\ \frac{dx+ey+f}{gx+hy+1} \\ 1 \end{bmatrix}$$

When denominator is 0, x and y will become infinite. This is basically saying that we can warp it to have image borders meet outside of the image.



Difference from homographies (affine transformations and projective warps) compared to linear transformations is that projective warps don't have origin to origin, parallel lines do not remain parallel and ratios are not preserved.

2D image transformations



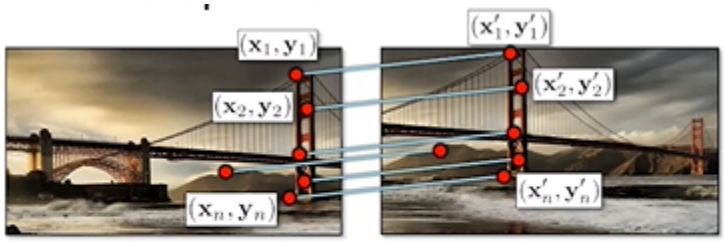
Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I \mid t \end{bmatrix}_{2\times3}$	2	orientation + ···	
rigid (Euclidean)	$\begin{bmatrix} R \mid t \end{bmatrix}_{2\times3}$	3	lengths + ···	
similarity	$\begin{bmatrix} sR \mid t \end{bmatrix}_{2\times3}$	4	angles + ···	
affine	$\begin{bmatrix} A \end{bmatrix}_{2\times3}$	6	parallelism + ···	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3\times3}$	8	straight lines	

Forwarding warping: when a transformation matrix sends a pixel to not an integer location and lands "in between" two pixels, we need to add "contribution" and splatting. We can interpolate it in several ways, like nearest neighbour. Or bilinear, bicubic, sync, etc.

Warping - transform pixels to new location, if they don't match up take average of neighbours and estimate new location.

We can also do inverse warping, where we get each pixel's colour value with T inverse.

Given a set of matches between images A and B, how can we compute the transform T from A to B? Find transform T that best agrees with A to B.



Displacement of match i = $(x'_i - x_i, y'_i - y_i)$

Look at the matching pairs and compute the displacement of the matches. We can then the mean of these matches.

Or we can look at this problem as a system of linear equations. With many matches we have an overdetermined system. In order to find the best solution we will use least squares.

Least squares formulation

- For each point (x_i, y_i)

$$x_i + x_t = x'_i$$
$$y_i + y_t = y'_i$$

- we define the *residuals* as

$$r_{x_i}(x_t) = (x_i + x_t) - x'_i$$
$$r_{y_i}(y_t) = (y_i + y_t) - y'_i$$

First define the residuals. For each transformed point, if we put all transformed pixels together, how good are we matching the final location for this pixel pair.

Minimise square of residuals. Perfect x_t will be 0, so we want to minimise it.

$$C(x_t, y_t) = \sum_{i=1}^{n} \left(r_{x_i}(x_t)^2 + r_{y_i}(y_t)^2 \right)$$

Least squares solution is minimum of the function C. We can also write is as a matrix equation.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ \vdots & \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x'_1 - x_1 \\ y'_1 - y_1 \\ x'_2 - x_2 \\ y'_2 - y_2 \\ \vdots \\ x'_n - x_n \\ y'_n - y_n \end{bmatrix} \rightarrow$$

$$\mathbf{A} \quad \mathbf{t} = \quad \mathbf{b}$$

$$\mathbf{At} = \mathbf{b}$$

- Find **t** that minimizes

$$||\mathbf{At} - \mathbf{b}||^2$$

- To solve, take derivative w.r.t. **t** and equate to 0

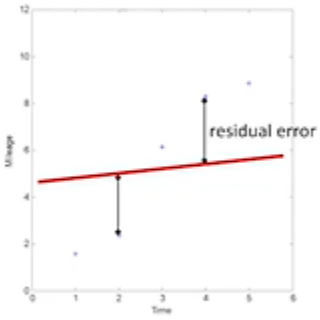
$$\mathbf{A}^{\mathrm{T}}\mathbf{At} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$$

$$\mathbf{t} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$$

This error function will never be zero. When derivative of error function is zero that means we are at a min/max of error.

This is similar to a best fit line for linear regressions.

Linear regression



$$\text{Cost}(m,b) = \sum_{i=1}^{n} |y_i - (mx_i + b)|^2$$

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

For affine transformations:

- Residuals:

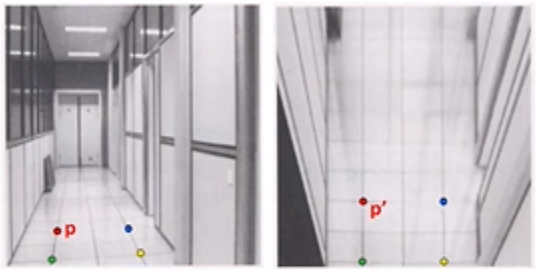
$$r_{x_i}(a,b,c,d,e,f) = (ax_i + by_i + c) - x_i'$$
$$r_{y_i}(a,b,c,d,e,f) = (dx_i + ey_i + f) - y_i'$$

- Cost function:

$$C(a,b,c,d,e,f) = \sum_{i=1}^{n} \left(r_{x_i}(a,b,c,d,e,f)^2 + r_{y_i}(a,b,c,d,e,f)^2\right)$$

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ & & & \vdots & & \\ x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ \vdots \\ x_n' \\ y_n' \end{bmatrix}$$

To unwarp an image in a homography, we need at least 4 points.



To unwarp (rectify) an image

- solve for homography **H** given **p** and **p'**
- solve equations of the form: **wp' = Hp**
  - linear in unknowns: w and coefficients of **H**
  - H is defined up to an arbitrary scale factor

Solving for homographies

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x_i' = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}$$
$$y_i' = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

Not linear!

$$x_i'(h_{20}x_i + h_{21}y_i + h_{22}) = h_{00}x_i + h_{01}y_i + h_{02}$$
$$y_i'(h_{20}x_i + h_{21}y_i + h_{22}) = h_{10}x_i + h_{11}y_i + h_{12}$$

Transform this into

$$
\begin{aligned}
x_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\
y_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12}
\end{aligned}
$$

$$
\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix}
\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
$$

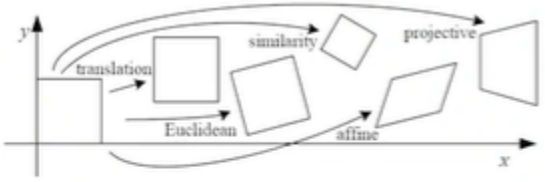
Solving for homographies

$$
\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n & -x_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n & -y_n' \end{bmatrix}
\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}
$$

$\mathbf{A}_{2n \times 9}$ $\mathbf{h}_{9}$ $\mathbf{0}_{2n}$

Defines a least squares problem: minimize $\|\mathbf{A}\mathbf{h} - \mathbf{0}\|^2$

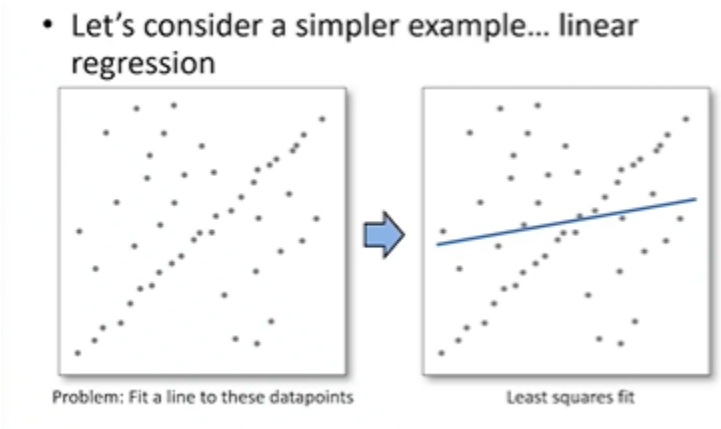
- Since $\mathbf{h}$ is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
- Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue
- Works with 4 or more points



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation + ···	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths + ···	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles + ···	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism + ···	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

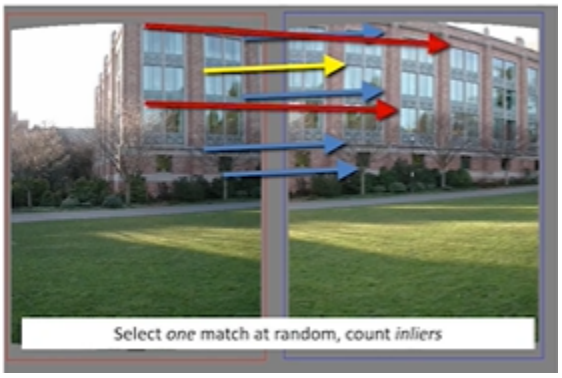
February 2nd - RANSAC

We have seen how to identify features and outliers. We want a robust method.

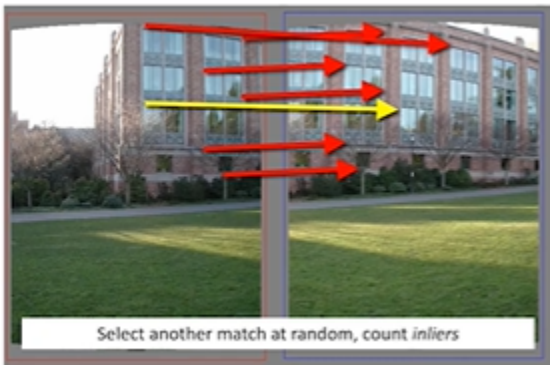


This happens when we take average of all points. We can fix this by giving a hypothesised line and count the number of points that "agree" (inliners) with the line.

For all possible lines, select the number with the largest number of inliers. This is a hypothesise and test method.
RAndom SAmple Consensus: Pick one single match at random and count how many other matches agree with the transformation.



Picking an inliner point will give mostly positive match results

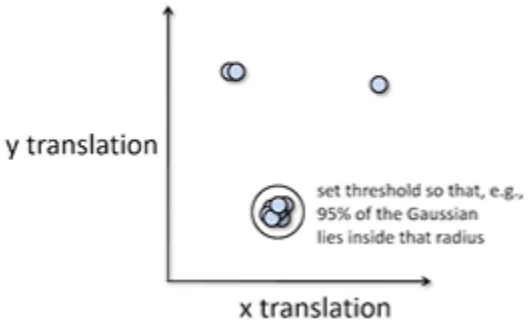


Picking an outlier point.

This is the main idea, iterate over all transformation vectors and pick the most matches.
Inlier threshold: how much of an error we expect from a point that is correctly matched. For a correct match we will still have noise, so we need to differentiate between inlier and outliers.
We often model noise as gaussian with some std deviation.

Number of rounds: How many rounds we need to get solution.

RANSAC



General version:

1) Randomly choose s samples
    a) Typically s = minimum sample size that lets you fit a model
2) Fit a model (e.g) to those samples
3) Count number of inliers that approximately fit model
4) Repeat N times
5) Choose model that has largest set of inliers

How do we determine how many rounds?

- If we have to choose *s* samples each time
  - with an outlier ratio *e*
  - and we want the right answer with probability *p*

$$N \geq \frac{\log(1-p)}{\log(1-(1-e)^s)}$$

After RANSAC with our set of inliers, we can use least squares to get average of translation vector after over inliers.

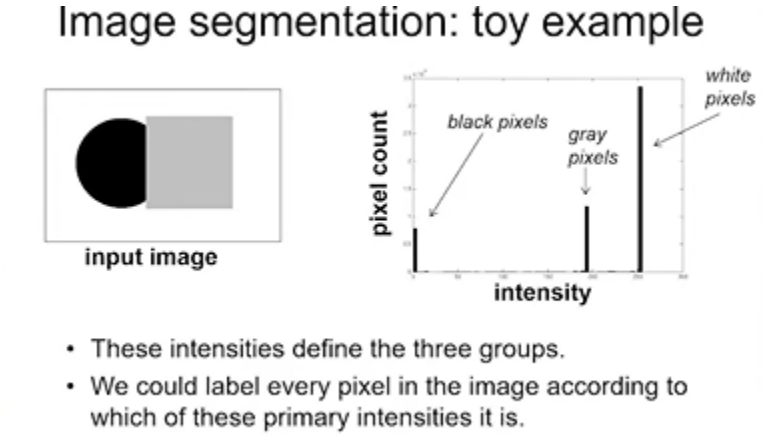
- Now we know how to create panoramas!

- Given two images:
  - Step 1: Detect features
  - Step 2: Match features
  - Step 3: Compute a homography using RANSAC
  - Step 4: Combine the images together

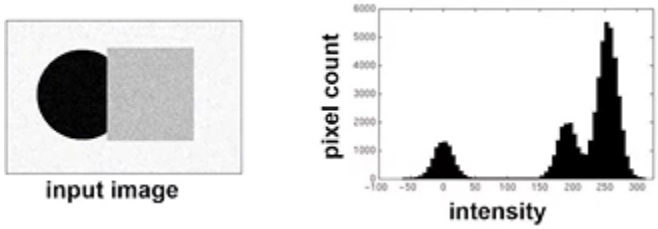
https://www.mathworks.com/help/vision/ug/feature-based-panoramic-image-stitching.html

Image segmentation

Segmentation - separating image regions into different parts.
Either colour, semantic (AI), brightness, etc.



Real life is not that simple, adding some noise to image we get rounded peaks.



We can identify the three main intensities by clustering. Choose 3 centres as the representatives of those intensities and label every pixel according to which of the centres it is nearest to.

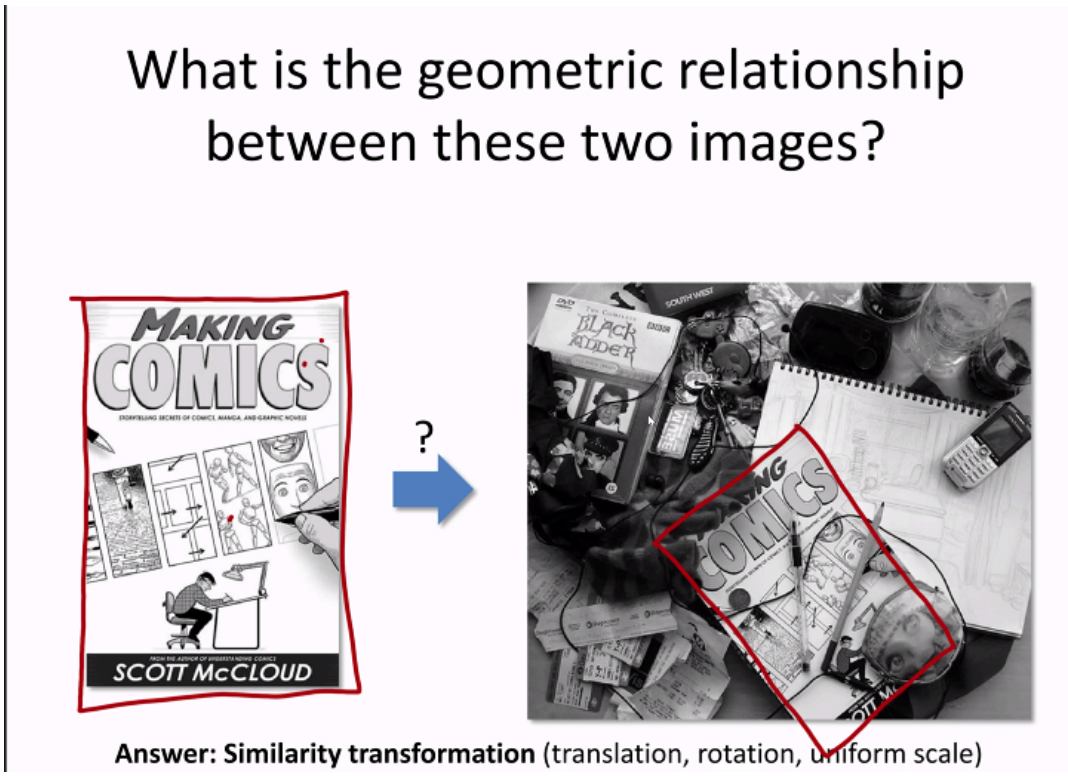
K-means: assume each group is a gaussian distribution, we can measure the distance of each point to the peak of the gaussian distributions.

$$P(x_i|\mu_j) \propto e^{-\frac{1}{2\sigma^2}\|x_i-\mu_j\|^2}$$

- Suppose we know all μ_j. Which group should a point x_i belong to?
  - The j with highest $P(x_i|\mu_j)$
  - = The j with smallest $||x_i - \mu_j||^2$

This only works when you know the mean and std deviations, but we know the set of points that belong to each cluster. Then we can calculate mean and standard dev.
Stopped at 6:31 of video.

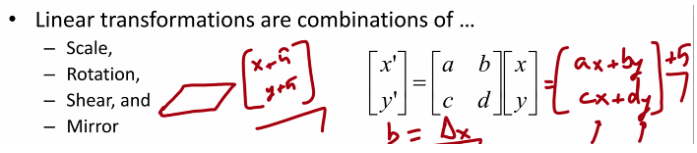
February 2nd - Online class on campus

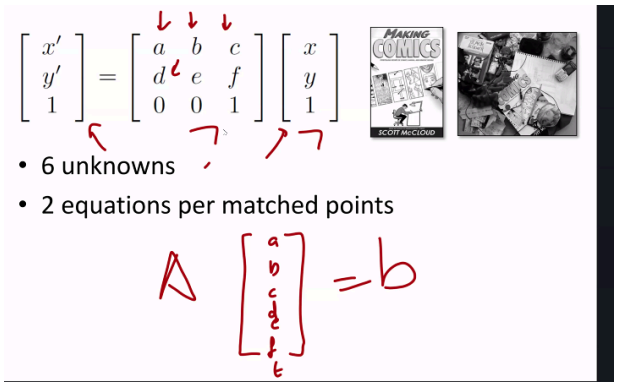


This is typically just a matrix transformation.

The images we need for our assignment to stitch back together can overlap. For the matching parts in a pair photographs we need to decide how to combine them.

Lining up them exactly will create a blur from the slight difference in viewing angle and perspective, we can't just directly overlay them.
We achieve this goal with homography.





Continued Feburary 4th
Step 1: Assign k centers randomly
Step 2: Assign each point to each assigned center
Step 3: Given assigned points, compute new mean values and centers
Step 4: Repeat until result is stable

K-means

Input: set of data points, k

1. Randomly pick k points as means
2. For i in [0, maxiters]:
   1. Assign each point to nearest center
   2. Re-estimate each center as mean of points assigned to it

K-means - the math

- An objective function that must be minimized:

$$\min_{\mu,y} \sum_i \|x_i - \mu_{y_i}\|^2$$

- Every iteration of k-means takes a downward step:
  - Fixes μ and sets y to minimize objective
  - Fixes y and sets μ to minimize objective

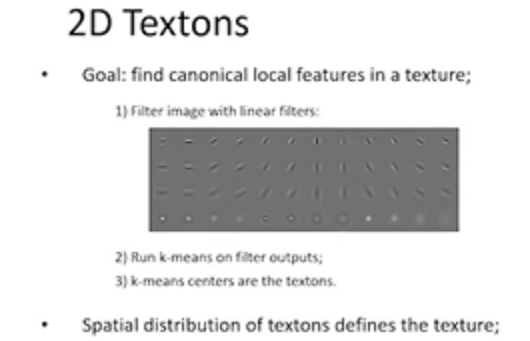
K-means can be inaccurate as it will misinterpret nearby pixels as not being part of the feature by being an odd colour. We can correct this by incorporating pixel position for segmenting.

- Captures pixel similarity but
  - Doesn't capture continuity of contours
  - Captures near/far relationships only weakly
  - Can merge far away objects together
- Requires knowledge of k!
- Can it deal with texture?

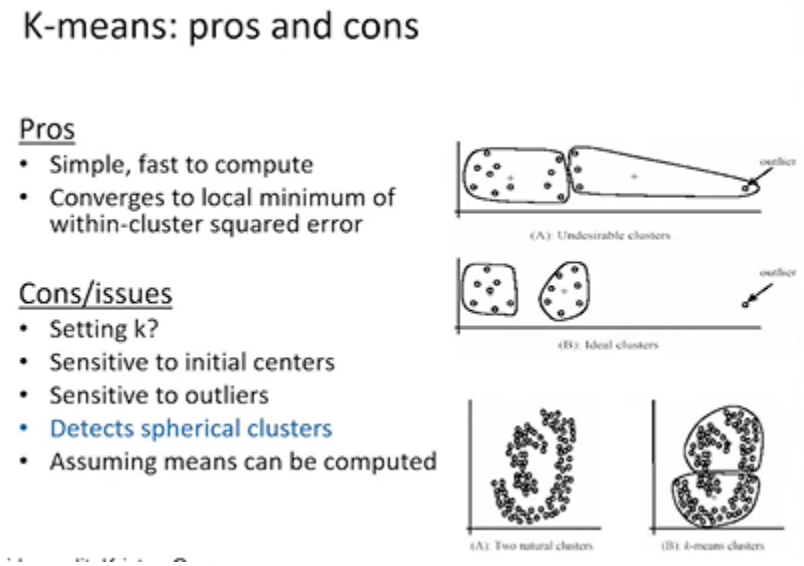
Texture: Kinda ambiguous to define. Some sort of repeating pattern.
When can we detect texture boundaries? (Change of texture)
We can do this with Julesz's texton theory. We can detect "density" of certain elements.

- Textons are:
  - Elongated blobs - e.g. rectangles, ellipses, line segments with specific orientations, widths and lengths
  - Terminators - ends of line segments
  - Crossings of line segments
- Julesz arrived at these by experimenting on which textures were distinguishable

Using a set of filters to detect oriented edges (distinguishable small edges) we can detect textons and repeated structures.

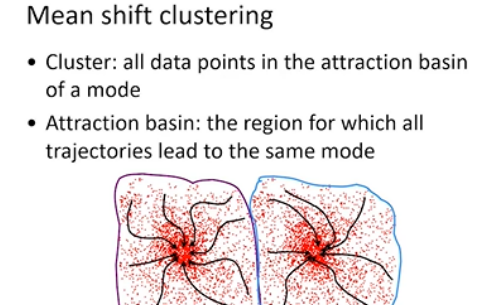


We can model convolution as cross correlation. When we are convolving an image with a kernel, we are determining how well fit that kernel is for a specific pixel.



Another classical approach to segmentation: Mean shift algorithm
The Mean Shift Algorithm seeks modes or local maxima of density in feature space. It also uses the LUV colour space.

For a specific pixel, define a search window of a specific radius around the pixel. Then define a centre of mass (weight average) of that search window. Shift search window to centre of mass.
Repeat until the search window isn't changing. This is like a path finding (this could totally be modelled by a vector field)



Mean shift segmentation:

1) Find features (colour, gradients, texture, etc.)
2) Initialise windows at individual feature points
3) Perform mean shift for each window until convergence
4) Merge windows that end up near same “peak” or mode

- Pros:
  - Does not assume shape on clusters
  - One parameter choice (window size)
  - Generic technique
  - Find multiple modes
- Cons:
  - Selection of window size
  - Does not scale well with dimension of feature space

Superpixels: Separate image into semantic objects. Small coherent regions that encapsulate multiple pixels.

Algorithm 1 Efficient superpixel segmentation

1: Initialize cluster centers $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ by sampling pixels at regular grid steps S.
2: Perturb cluster centers in an $n \times n$ neighborhood, to the lowest gradient position.
3: **repeat**
4: **for** each cluster center C_k **do**
5: Assign the best matching pixels from a $2S \times 2S$ square neighborhood around the cluster center according to the distance measure (Eq. 1).
6: **end for**
7: Compute new cluster centers and residual error E {$L1$ distance between previous centers and recomputed centers}
8: **until** $E \leq$ threshold
9: Enforce connectivity.

Graph-based segmentation

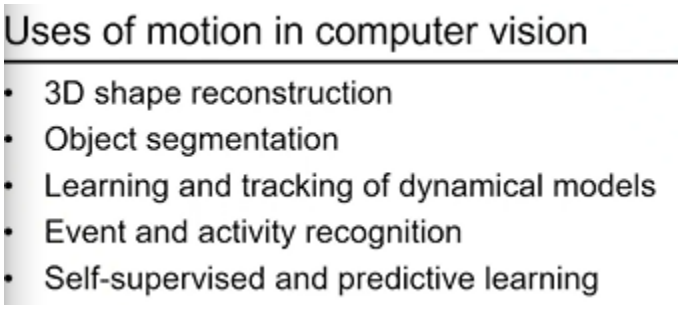
- Define a node for each pixel or the superpixel in the graph
- Define a similarity metric (for example color) between pixels and add them as edges between nodes in the graph
- Solve…

https://www.mathworks.com/help/images/ref/imsegkmeans.html

PHD thesis of yegiz: https://yaksoy.github.io/papers/ETH19-PhD-Aksoy.pdf

February 4th: Optical flow

Measuring motion between two photos or multiple frames.

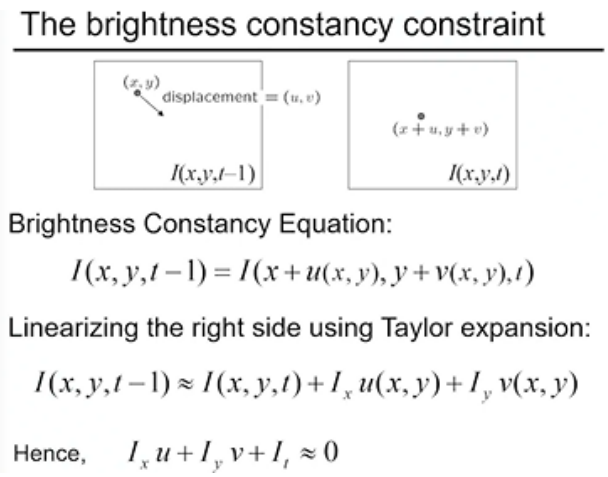


Motion field is projection of 3D scene motion into the image, these motions get translated into different field vectors when we detect them.
Definition: optical flow is the apparent motion of brightness patterns in the image. Ideally, optical flow would be the same as the motion field. We have to be cautious though, apparent motion can be caused by lighting effects without any actual motion.

Given two subsequent frames, estimate apparent motion field u(x,y) and v(x,y) between them.
Two assumptions:

1) Brightness constancy: projection of same point looks the same in every frame
2) Small motion: points do not move very far
3) Spatial coherence: points move like their neighbours.

Now, instead of having just x and y for pixel position, we also have t for specifying time and which frame it is. In this picture, we use bright constancy assumption.

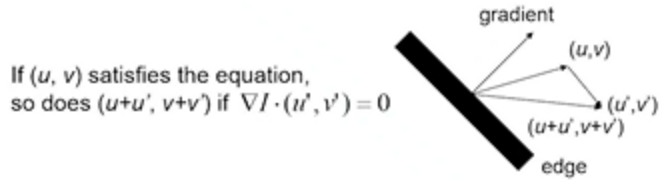


This is one equation, two unknowns.

- What does this constraint mean?

$$\nabla I \cdot (u,v) + I_t = 0$$

- The component of the flow perpendicular to the gradient (i.e., parallel to the edge) is unknown!



Here, the u,v and u',v' will give the same amount of motion in the direction of the gradient.
This is called the aperture problem, where the perceived motion isn't the actual motion where this comes from our under constrained equations with two unknowns.

Think of it as a barber pole illusion, where we think the stripes are moving down, when we know they're following a helix.
We can fix this with the spatial coherence constraint.

- How to get more equations for a pixel?
- **Spatial coherence constraint:** assume the pixel's neighbors have the same (u,v)
  - E.g., if we use a 5x5 window, that gives us 25 equations per pixel

$$\nabla I(\mathbf{x}_i) \cdot [u,v] + I_t(\mathbf{x}_i) = 0$$

$$\begin{bmatrix} I_x(\mathbf{x}_1) & I_y(\mathbf{x}_1) \\ I_x(\mathbf{x}_2) & I_y(\mathbf{x}_2) \\ \vdots & \vdots \\ I_x(\mathbf{x}_n) & I_y(\mathbf{x}_n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{x}_1) \\ I_t(\mathbf{x}_2) \\ \vdots \\ I_t(\mathbf{x}_n) \end{bmatrix}$$

- Linear least squares problem:

$$\begin{bmatrix} I_x(\mathbf{x}_1) & I_y(\mathbf{x}_1) \\ I_x(\mathbf{x}_2) & I_y(\mathbf{x}_2) \\ \vdots & \vdots \\ I_x(\mathbf{x}_n) & I_y(\mathbf{x}_n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{x}_1) \\ I_t(\mathbf{x}_2) \\ \vdots \\ I_t(\mathbf{x}_n) \end{bmatrix} \qquad \underset{n\times 2}{\mathbf{A}}\ \underset{2\times 1}{\mathbf{d}} = \underset{n\times 1}{\mathbf{b}}$$

- Solution given by $(\mathbf{A}^T\mathbf{A})\mathbf{d} = \mathbf{A}^T\mathbf{b}$

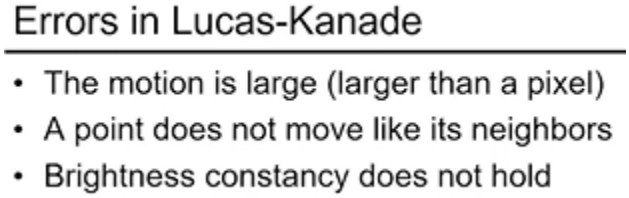
$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

(summations are over all pixels in the window)

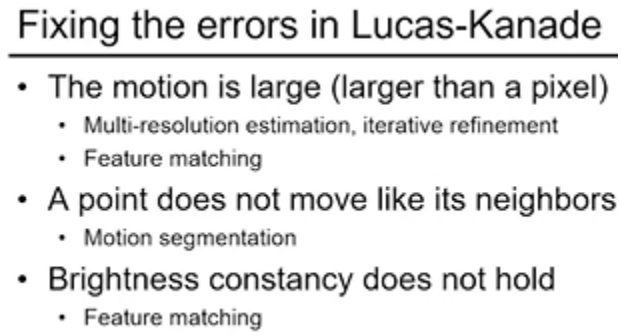
$\mathbf{M} = \mathbf{A}^T\mathbf{A}$ is the second moment matrix!

Second moment matrix from harris corner detection.
Estimation of optical flow is well-conditioned for regions with high corner detection.

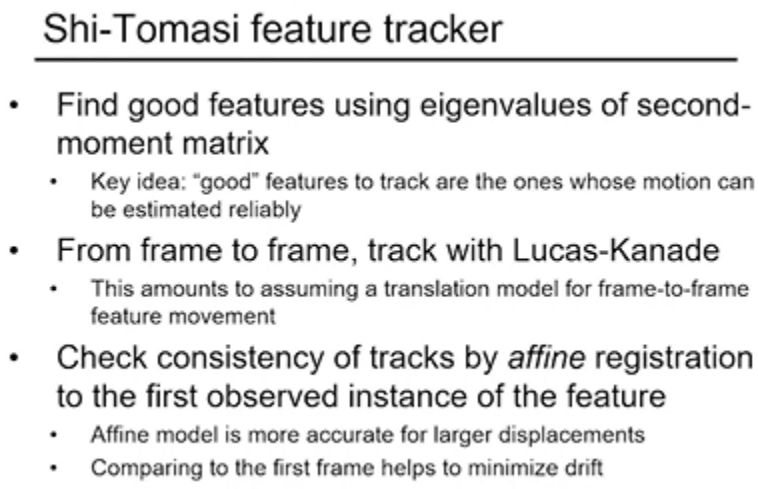
This means bad cases will be single straight edges moving, while good cases will be corners moving.
Lucas-Kanade flow: One of the best optical flow estimations today.



We can fix this with multi-resolution (gaussian blur/sub sampling to make smaller images), making smaller and smaller images to make the large motion appear small.
Iterative refinement: estimate motion in smaller image and keep estimating to larger and larger image.



Feature tracking: Detect features, for each subsequent image try to match features to themselves and track their motion.
Features could fall out of frame or get added, so we have to consider that.

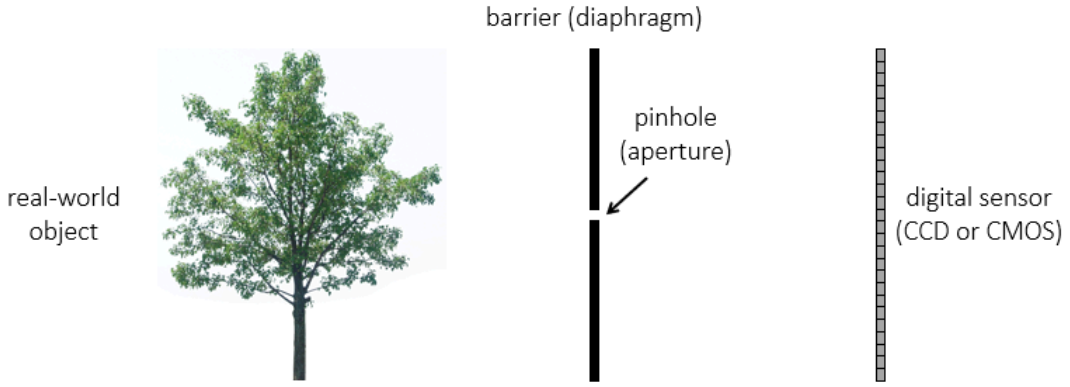


February 9th - Cameras and projection

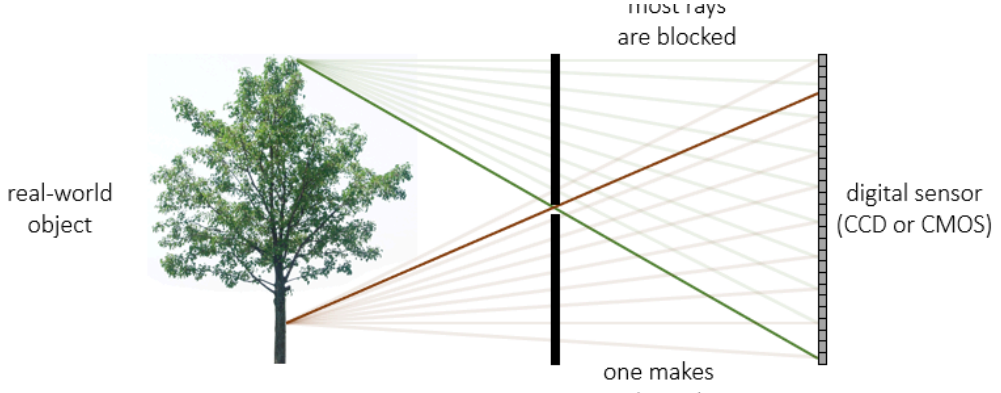
“The questions are designed to check your understanding of the course in general”
“No multiple choice, just written. Conceptual, mathematical, pretty much stuff we do in video lectures”
“When studying: do not think of maximising your score. Try to study your general comprehension of the course”
“Filtering for example, you should understand why filtering is useful”
“No MATLAB code, just theoretical stuff”
“There won’t be any matrix number calculations, there will be math but it’s simple”
“For example, harris corner derivation has very specific math. You need to understand how the derivation steps are connected. Like why we use taylor series representation”

“Finally we are starting to live in the third dimension”

We have a digital sensor that projects a 3D object into a 2D image. We model cameras by a “pinhole”.

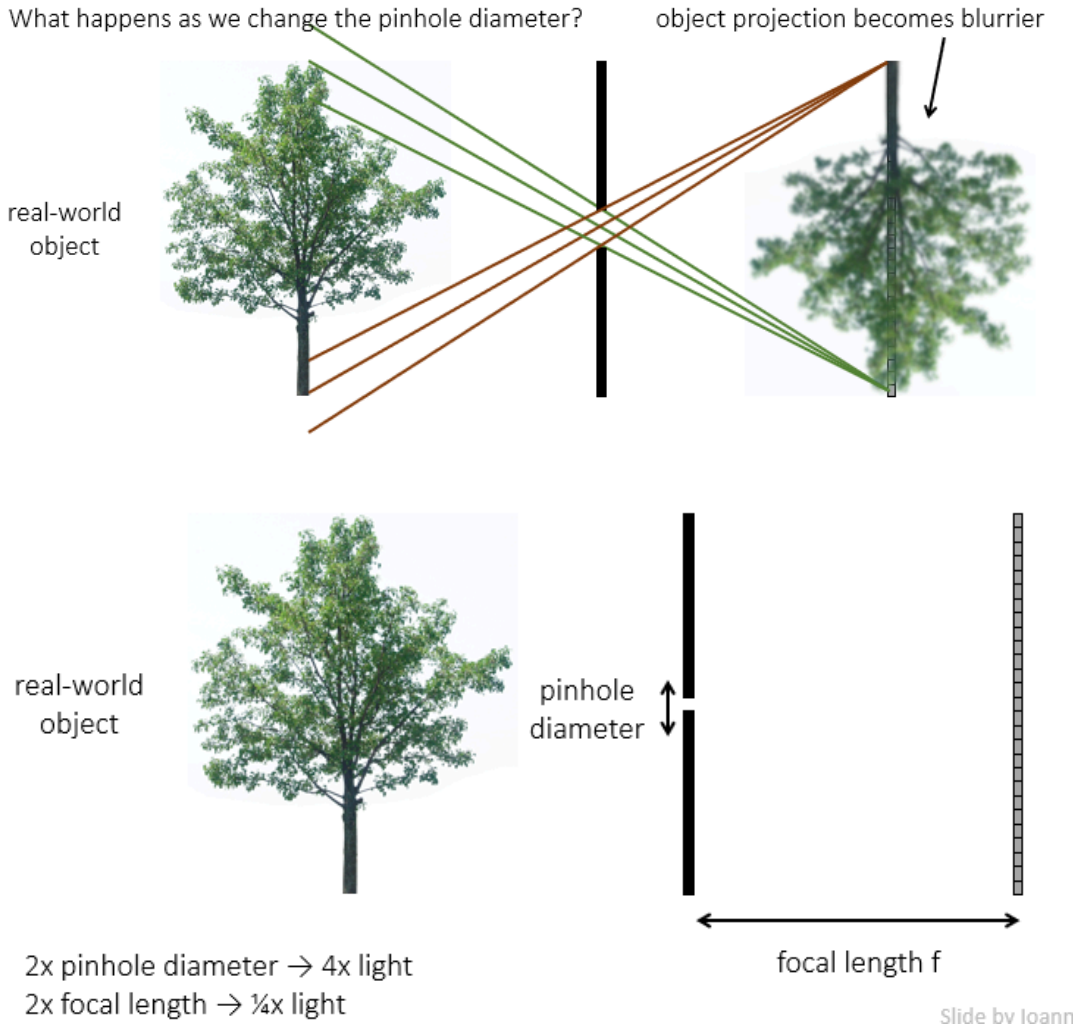


We have an aperture for the directional aspect, where we capture light coming from only the proper angle.
To model the aperture, we simulate the slit being infinitely small. Aperture is also called “centre of projection”



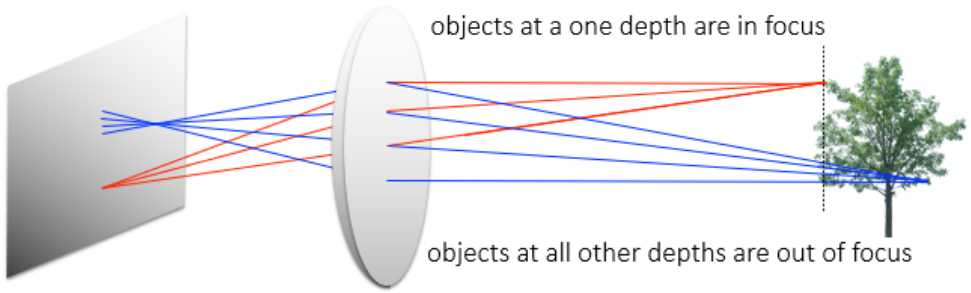
The detection will also be

relatively upside down.
Focal length is distance from aperture to detection screen. Moving it closer to the aperture will make the field of view narrower.



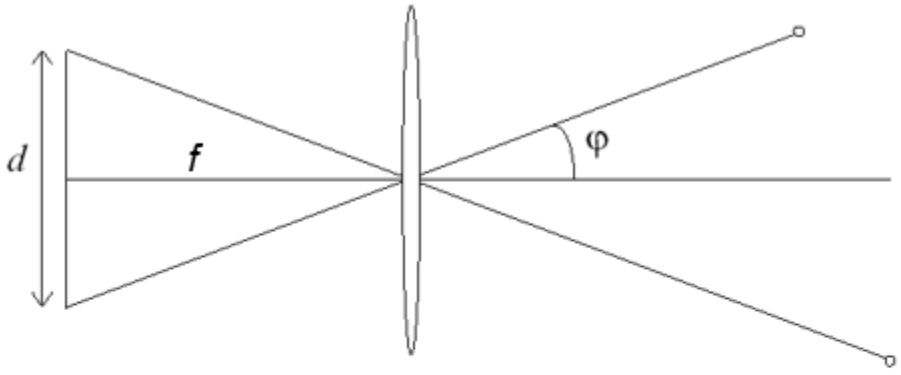
Using a convex lenses we focus the rays of light to the sensor.

1) Image is sharp
2) Signal-to-noise ratio is high



This is called bokeh, where a certain distance and part of the image is

focused. This is caused by the position of the lenses.
We can also change the shape of the pinhole to change the light perception and the bokeh.
Field of view is directly related to focal length. The more focused our image is, the less things we will see.



Size of field of view governed by size of the camera retina:

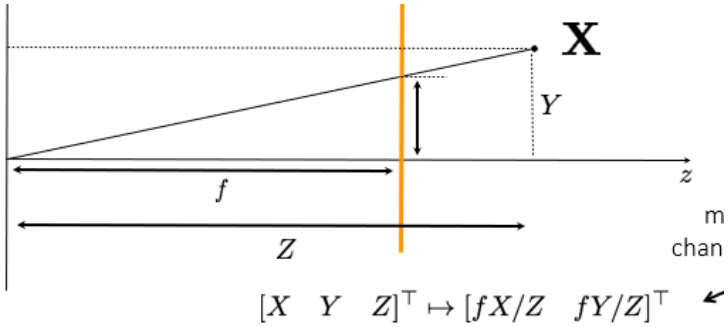
$$\varphi = \tan^{-1}\left(\frac{d}{2f}\right)$$

Smaller FOV = larger Focal Length

Changing the fov will shift the perspective of edges in the image. This is called the perspective effect.

Large FOV, small f
Camera close to car

Small FOV, large f
Camera far from the car



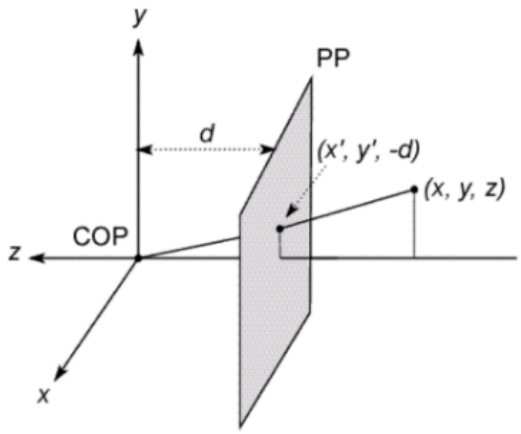
$$[X \quad Y \quad Z]^\top \mapsto [fX/Z \quad fY/Z]^\top$$

Here in the picture, the left corner point of the triangle is the

center of projection, and the yellow line is the image plane. f is the focal length and X is the 3d location of the object we're taking the photo of.

Projection

Modeling projection



- The coordinate system
  - We will use the pinhole model as an approximation
  - Put the optical center (**C**enter **O**f **P**rojection) at the origin
  - Put the image plane (**P**rojection **P**lane) *in front* of the COP
  - The camera looks down the *negative* z axis
    - we like this if we want right-handed-coordinates

- Compute intersection with PP of ray from (x,y,z) to COP
- Derived using similar triangles

$$(x, y, z) \rightarrow (-d\frac{x}{z}, \ -d\frac{y}{z}, \ -d)$$

- We get the projection by throwing out the last coordinate:

$$(x, y, z) \rightarrow (-d\frac{x}{z}, \ -d\frac{y}{z})$$

We represent this mapping in linear terms with homogenous coordinates by adding a dummy variable.

Homogeneous coordinates to the rescue—again!

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates

$$(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

homogeneous scene coordinates

Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)$$

Projection is a matrix multiply using homogeneous coordinate

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ -z/d \end{bmatrix} \Rightarrow (-d\frac{x}{z}, \quad -d\frac{y}{z})$$

divide by third coordinate

This a perspective projection, where the matrix is known as the projection matrix. We can also scale the matrix by d to change the transformation. This will alter the field of view.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ -z/d \end{bmatrix} \Rightarrow (-d\frac{x}{z}, \quad -d\frac{y}{z})$$

$$\begin{bmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} -dx \\ -dy \\ z \end{bmatrix} \Rightarrow (-d\frac{x}{z}, \quad -d\frac{y}{z})$$

We can see we can model an infinite FOV with focal length

getting infinitely large. This is called orthographic projection where the distance from the COP to PP is infinite.

Projection properties have a many-to-one mapping of points along the same ray map.
Parallel lines converge at a vanishing point where each direction has it's own vanishing point. This is how we perceive depth.

To project a point (x,y,z) in world coordinates into camera coordinates (x,y,z).
Need to know:

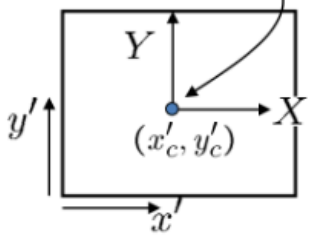
- Camera position
- Camera orientation
    - Then project into the image plane to get a coordinate

A camera is described by several parameters

- Translation T of the optical center from the origin of world coords
- Rotation R of the image plane
- focal length f, principal point (x'_c, y'_c), pixel size (s_x, s_y)
- blue parameters are called "extrinsics," red are "intrinsics"

Projection equation

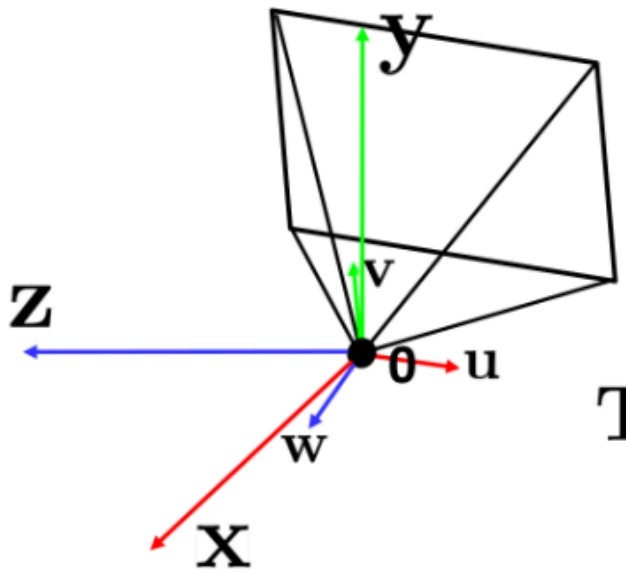
$$\mathbf{x} = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{\Pi X}$$



$$\mathbf{\Pi} = \begin{bmatrix} -fs_x & 0 & x'_c \\ 0 & -fs_y & y'_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{3x3} & \mathbf{0}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3x3} & \mathbf{T}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix}$$

intrinsics projection rotation translation

Extrinsics related our pin hole location to the real world camera.
We get the camera in canonical form (cop at origin, right hand orientation)



Step 1: Translate by -**c**

How do we represent translation as a matrix multiplication?

$$\mathbf{T} = \begin{bmatrix} \mathbf{I}_{3\times 3} & -\mathbf{c} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} -f & 0 & 0 \\ 0 & -f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

K (intrinsics) (converts from 3D rays in camera coordinate system to pixel coordinates)

in general, $\mathbf{K} = \begin{bmatrix} -f & s & c_x \\ 0 & -\alpha f & c_y \\ 0 & 0 & 1 \end{bmatrix}$ (upper triangular matrix)

α : **aspect ratio** (1 unless pixels are not square)

s : **skew** (0 unless pixels are shaped like rhombi/parallelograms)

(c_x, c_y) : **principal point** ((0,0) unless optical axis doesn't intersect projection plane at origin)

$$\mathbf{\Pi} = \mathbf{K} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{projection}} \underbrace{\begin{bmatrix} \mathbf{R} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} \mathbf{I}_{3\times 3} & -\mathbf{c} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix}}_{\text{translation}}$$

intrinsics

The k matrix converts 3D rays in cameras coord systems to 2d image points.

The right 3 matrices converts 3D points in world to 3D rays in 3 cameras coordinate system. There ate 6 parameters represented, 3 for positon/translation and 3 for rotation.