



Computer Vision

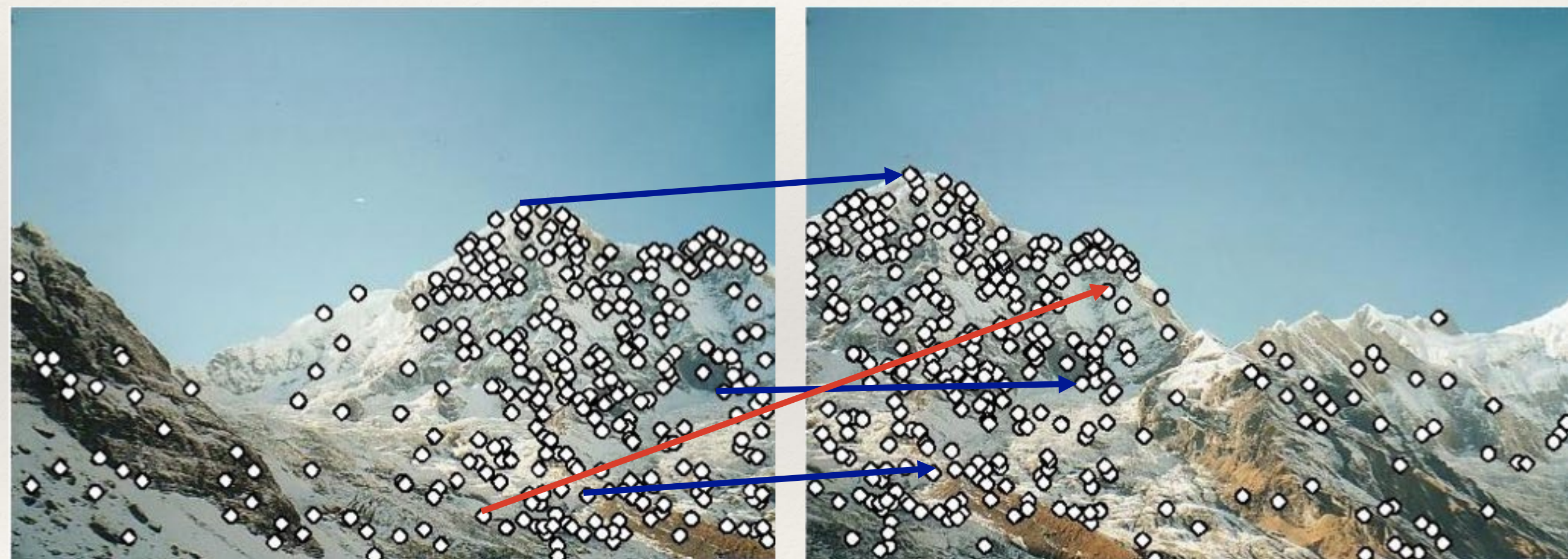
Consistent matching

Hansung Kim
h.kim@soton.ac.uk

Recap: local interest points, features and matching

- ❖ In the previous lectures we've looked at how we can:
 - ❖ Locate stable interest points in multiple images
 - ❖ Extract highly robust and discriminative local descriptors around each interest point
 - ❖ Match interest points across images by comparing their local features

How can we eliminate mismatches?



Feature distinctiveness

- ❖ Even the most advanced local feature can be prone to being mismatched.
- ❖ There is always a tradeoff in feature distinctiveness.
- ❖ If it's too distinctive will not match subtle variations due to noise of imaging conditions.
- ❖ If it's not distinctive enough it will match everything.



Constrained matching

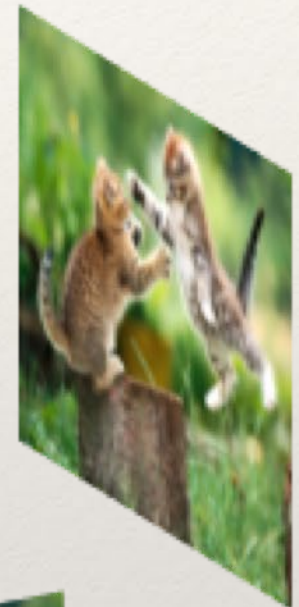
- ❖ Assume we are given a number of correspondences between the interest points in a pair of images
- ❖ Is it possible to estimate which of those correspondences are **inliers** (i.e. correct) or **outliers** (i.e. incorrect/mismatches)?
- ❖ What **assumptions** do we have to make?

By assuming a geometric mapping
between the two scenes can we
recover that mapping and eliminate
the mismatches?



Geometric Mappings

What are geometric transforms?



Point Transforms

We're interested in transforms that take the following form:

$$\mathbf{x}' = \mathbf{T}\mathbf{x}$$



Point Transforms

The transform matrix

$$x' = Tx$$

the transformed coordinate

the original coordinate



The Affine Transform

The affine transform is defined as

$$\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{b}$$

or

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$



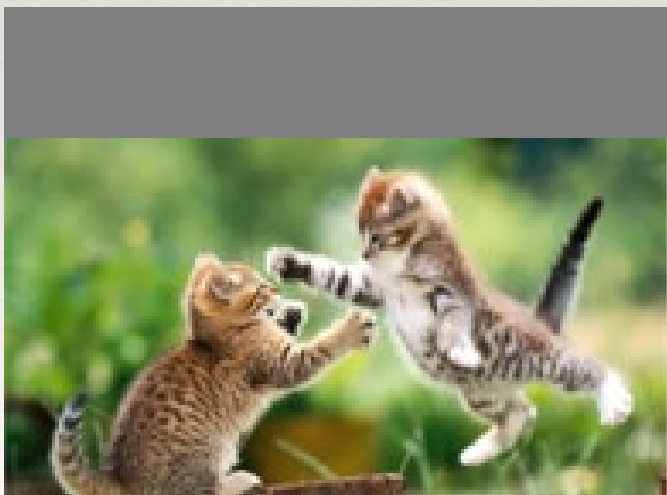
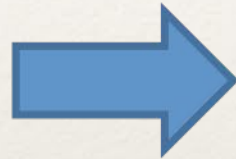
The Affine Transform

It's more convenient to write this as a single transform matrix by adding an extra dimension to each vector:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



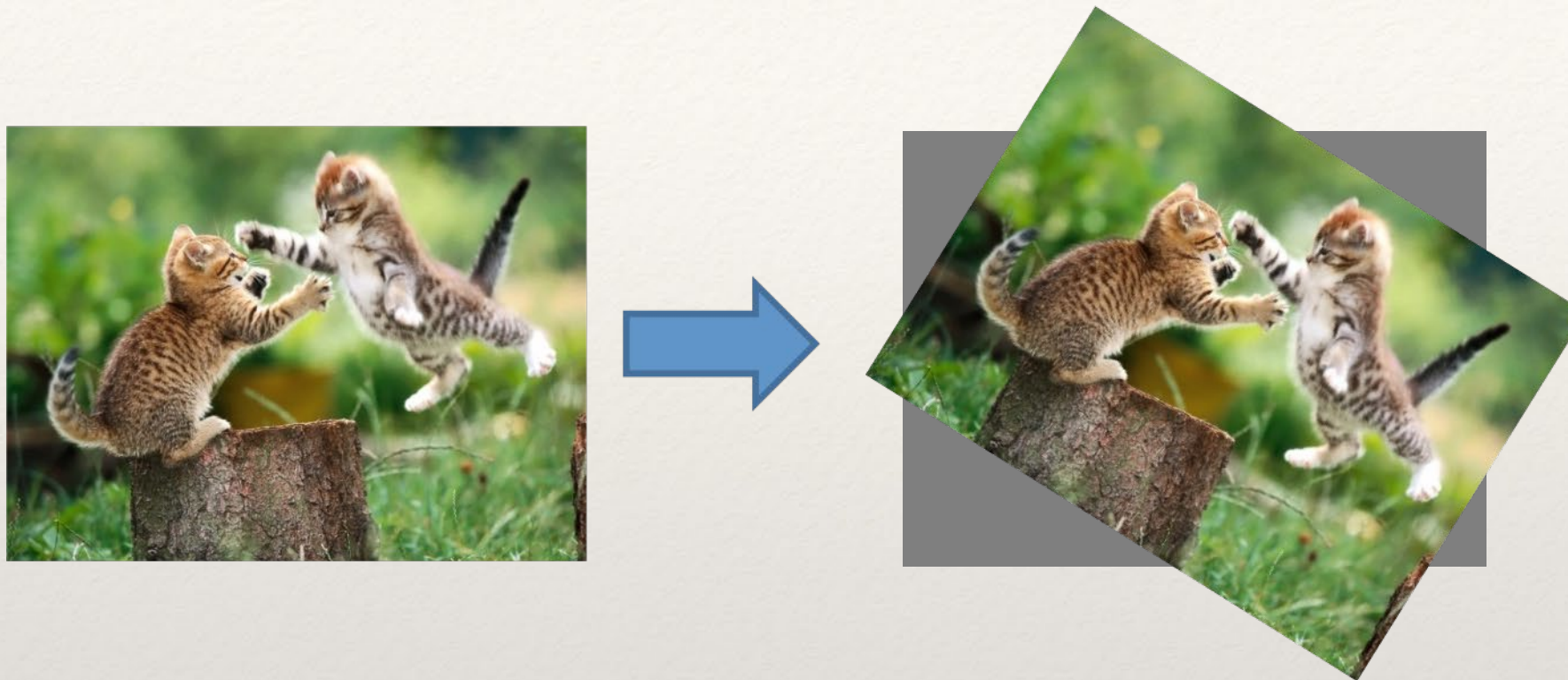
Translation



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



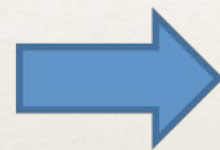
Translation and Rotation



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



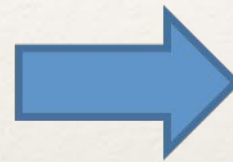
Scaling



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



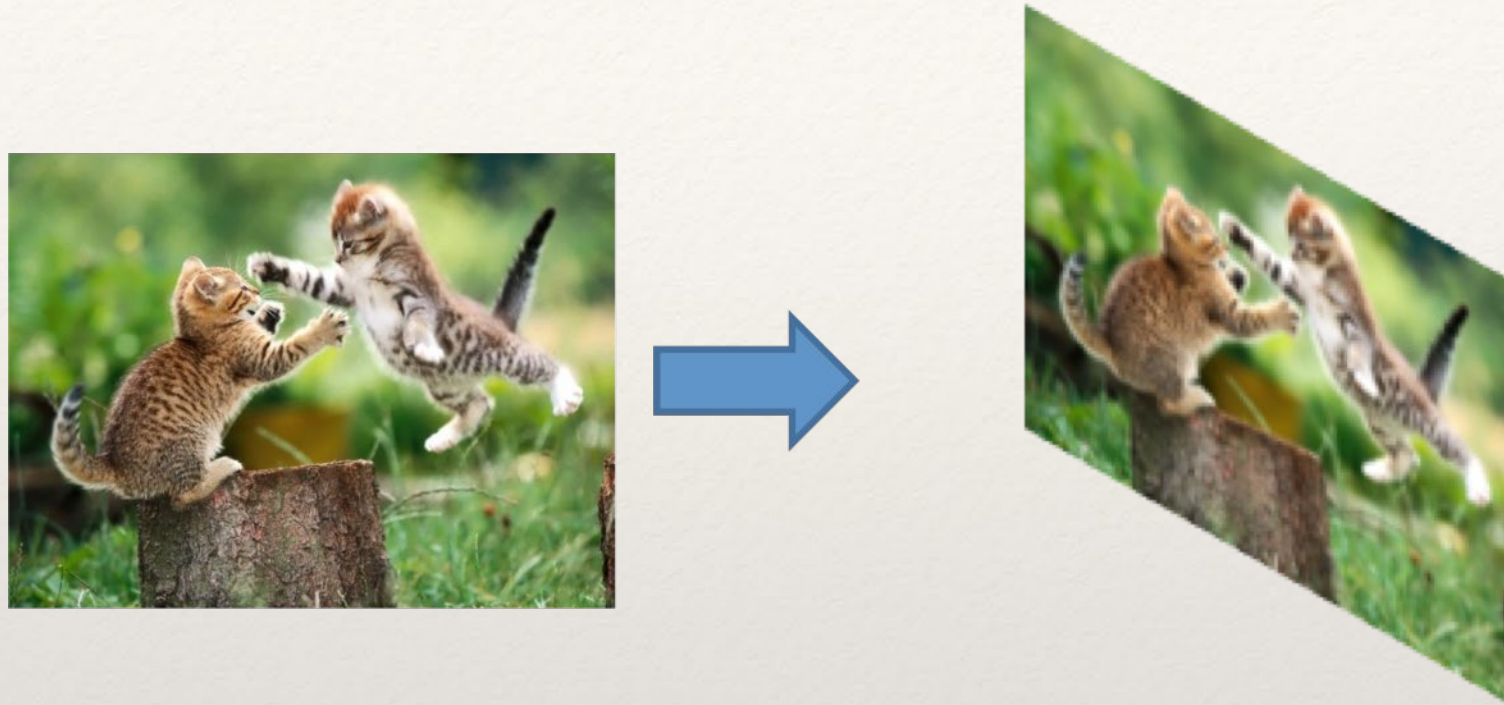
Aspect Ratio



$$\begin{bmatrix} a & 0 & 0 \\ 0 & \frac{1}{a} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$



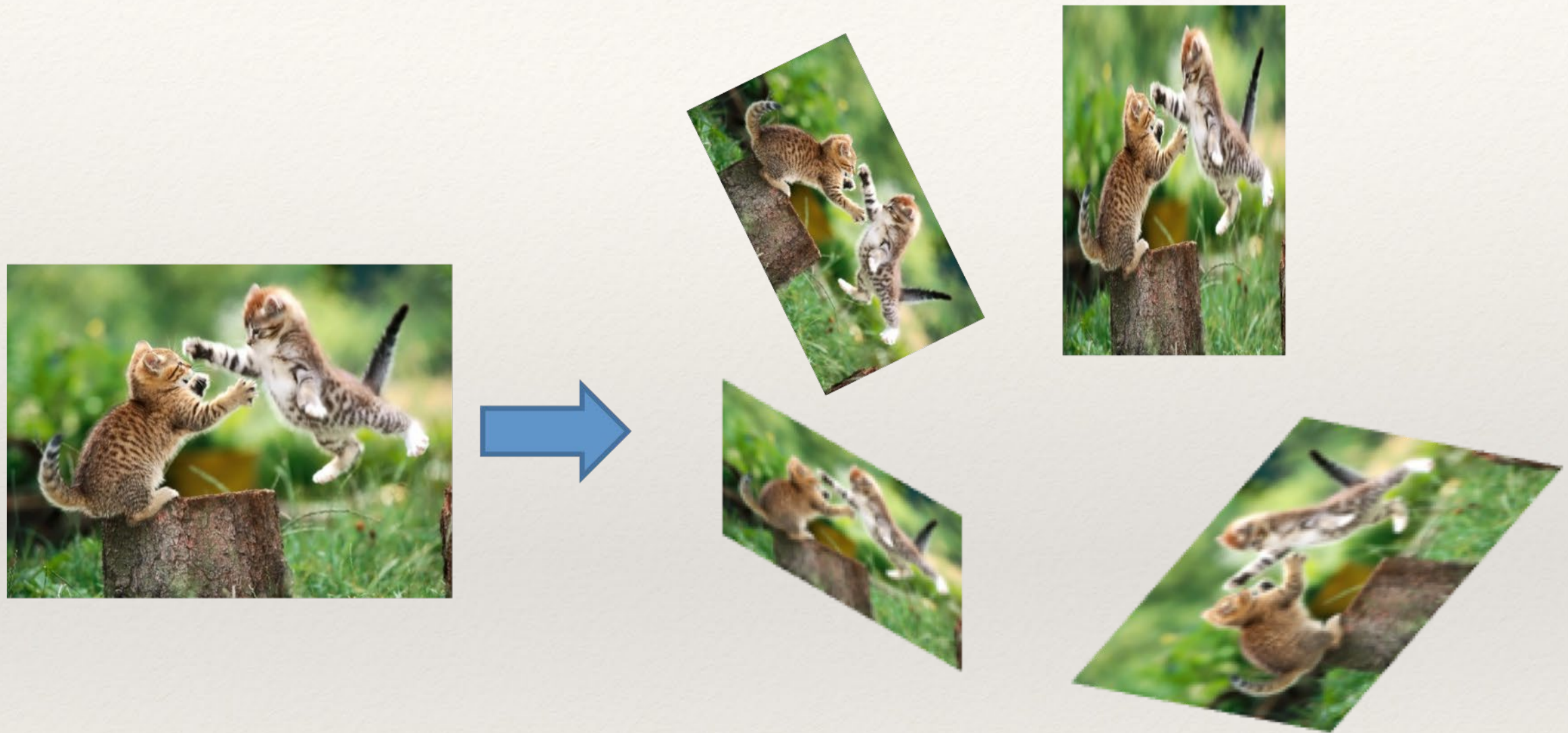
Shear



$$\begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$



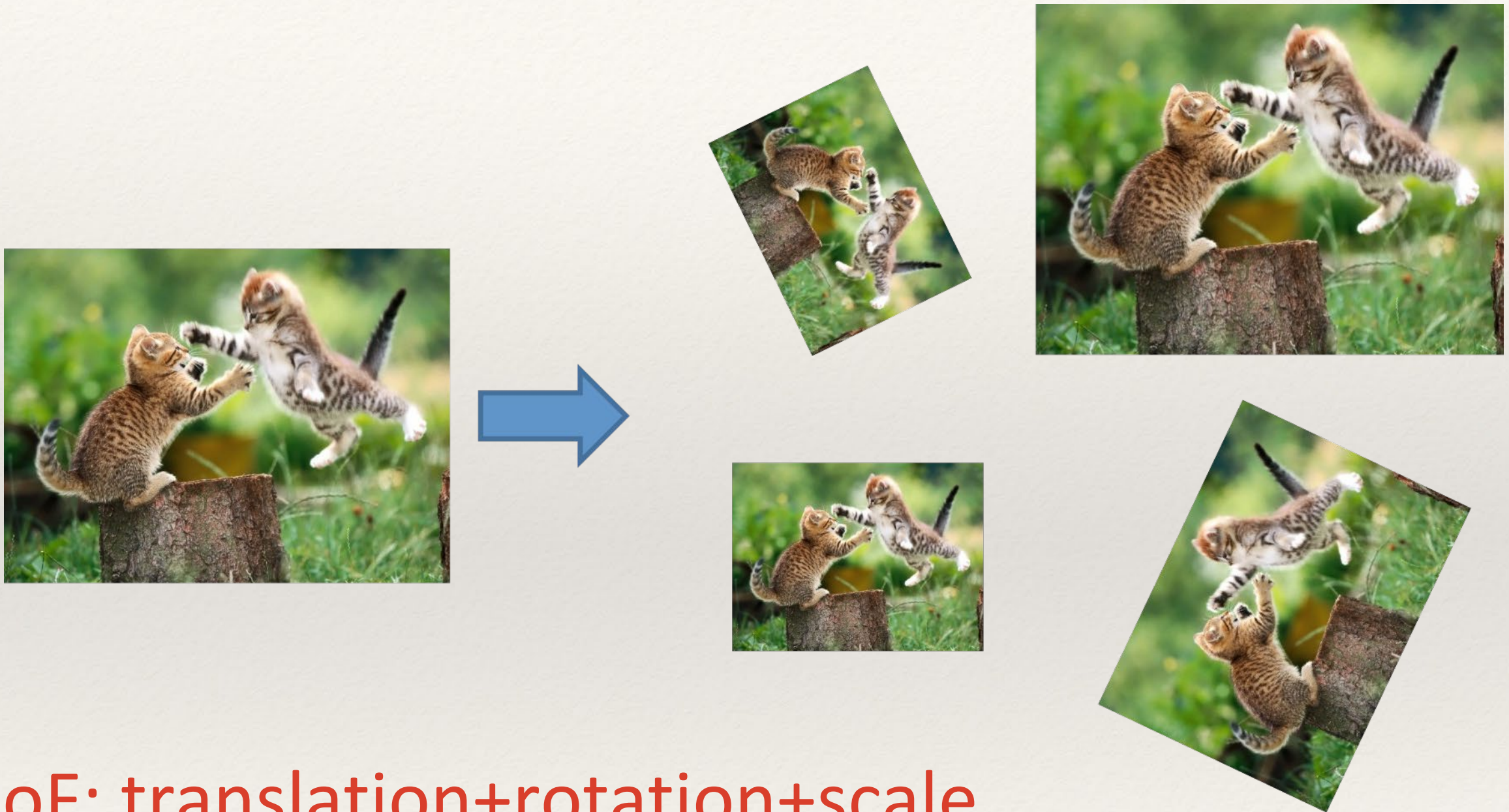
Affine Transform



6 DoF: translation + rotation + scale + aspect ratio + shear



Similarity Transform



4 DoF: translation+rotation+scale



What's missing?



More degrees of freedom

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

Normalise by w so that the transformed vector is $[\bullet, \bullet, 1]$

$$x' = u/w$$

$$y' = v/w$$



Homogeneous coordinates

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}$$



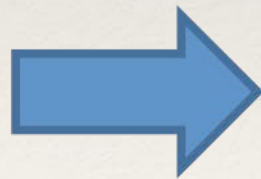
The Planar Homography (Projective Transformation)

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

$$x' = u/w$$

$$y' = v/w$$

Keystone distortions



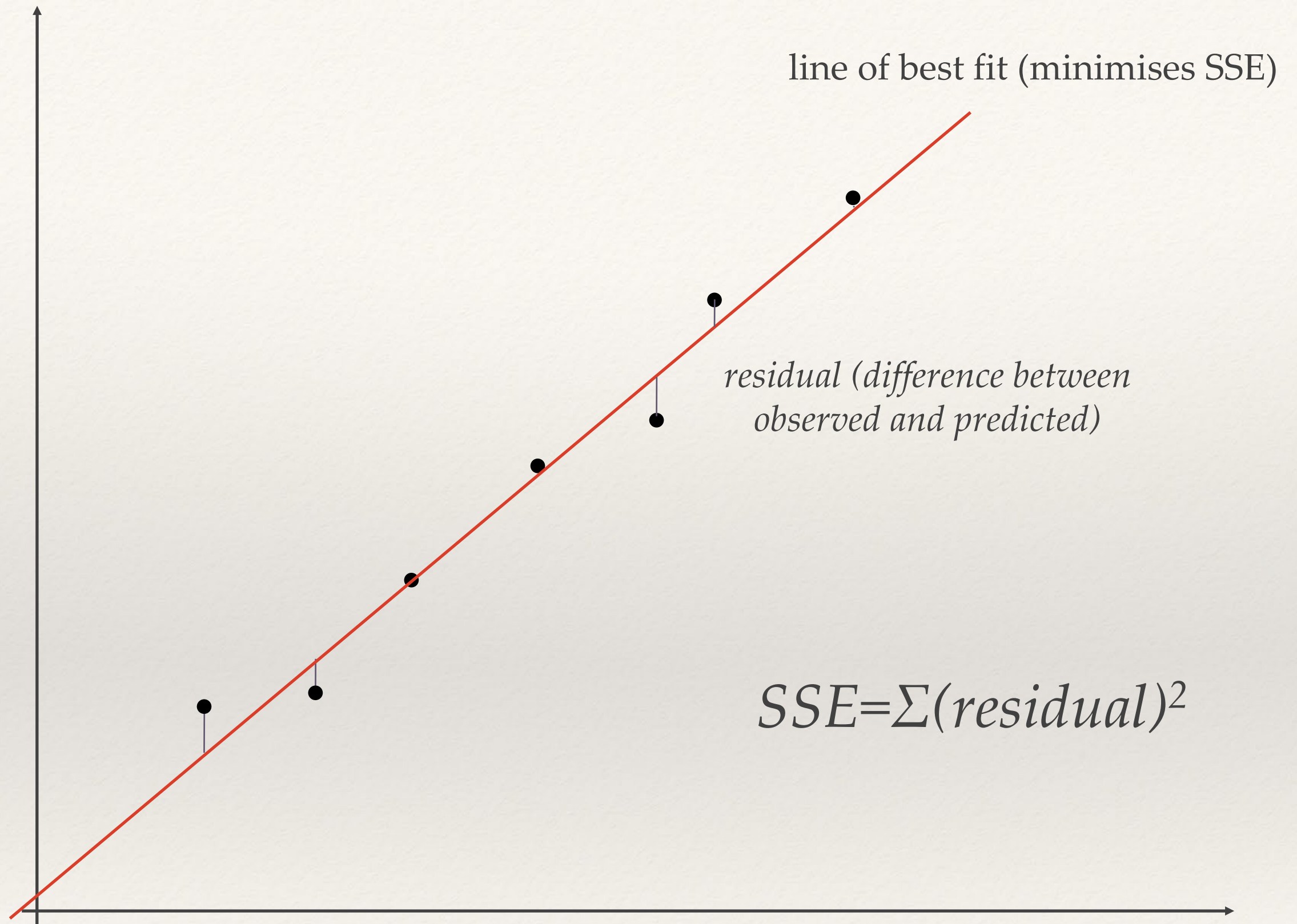
Recovering a geometric mapping

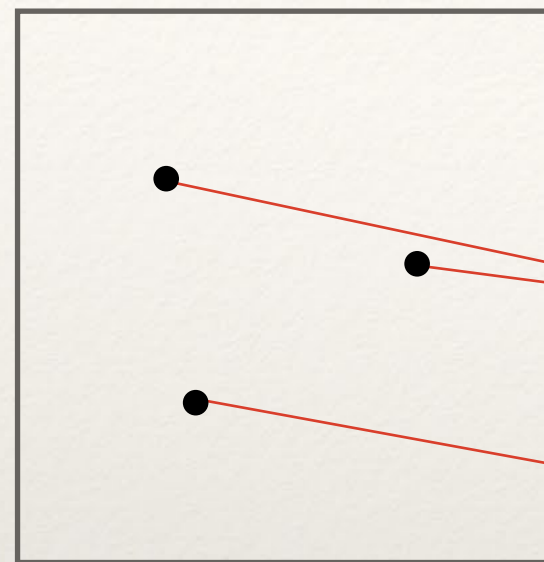
Simultaneous equations

- ❖ It is possible to estimate a transform matrix from a set of point matches by solving a set of simultaneous equations
 - ❖ Need at least 4 point matches to solve a Homography or 3 to solve an affine transform
- ❖ The actual solution technique isn't important...
 - ❖ It is important to note that in the presence of noise, and with potentially more matches than required, that we have to solve an *overdetermined system*
 - ❖ We need to seek the minimum error or **least-squares** solution

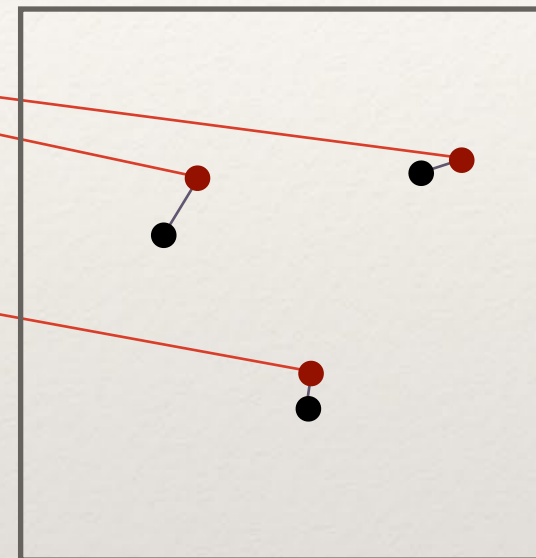


Least-squares





Red dots are predicted
positions from the estimated
transform



residuals are
Euclidean
distance between
observed and
predicted
positions

$$SSE = \sum (residual)^2$$



Robust Estimation

Problem: Noisy data

- ❖ Need a way to deal with estimating a model (i.e. a transform matrix) in the presence of high amounts of noise (i.e. mis-matches)
- ❖ Least-squares will be highly suboptimal, and probably find a very bad solution.
- ❖ Ideally, we want to identify the correct data (the inliers) and the bad data (the outliers)
 - ❖ Then estimate the model using only the good data.

Robust estimation techniques

- ❖ The problem of learning a model in the presence of inliers and outliers comes under an area of mathematics called **robust estimation** or **robust model fitting**
- ❖ There are a number of different possible techniques
 - ❖ ...lets look at one of the simplest...

RANSAC: RAndom SAmple Consensus

Assume:

M data items required to estimate model T

N data items in total

Algorithm:


1.) select M data items at random

2.) estimate model T

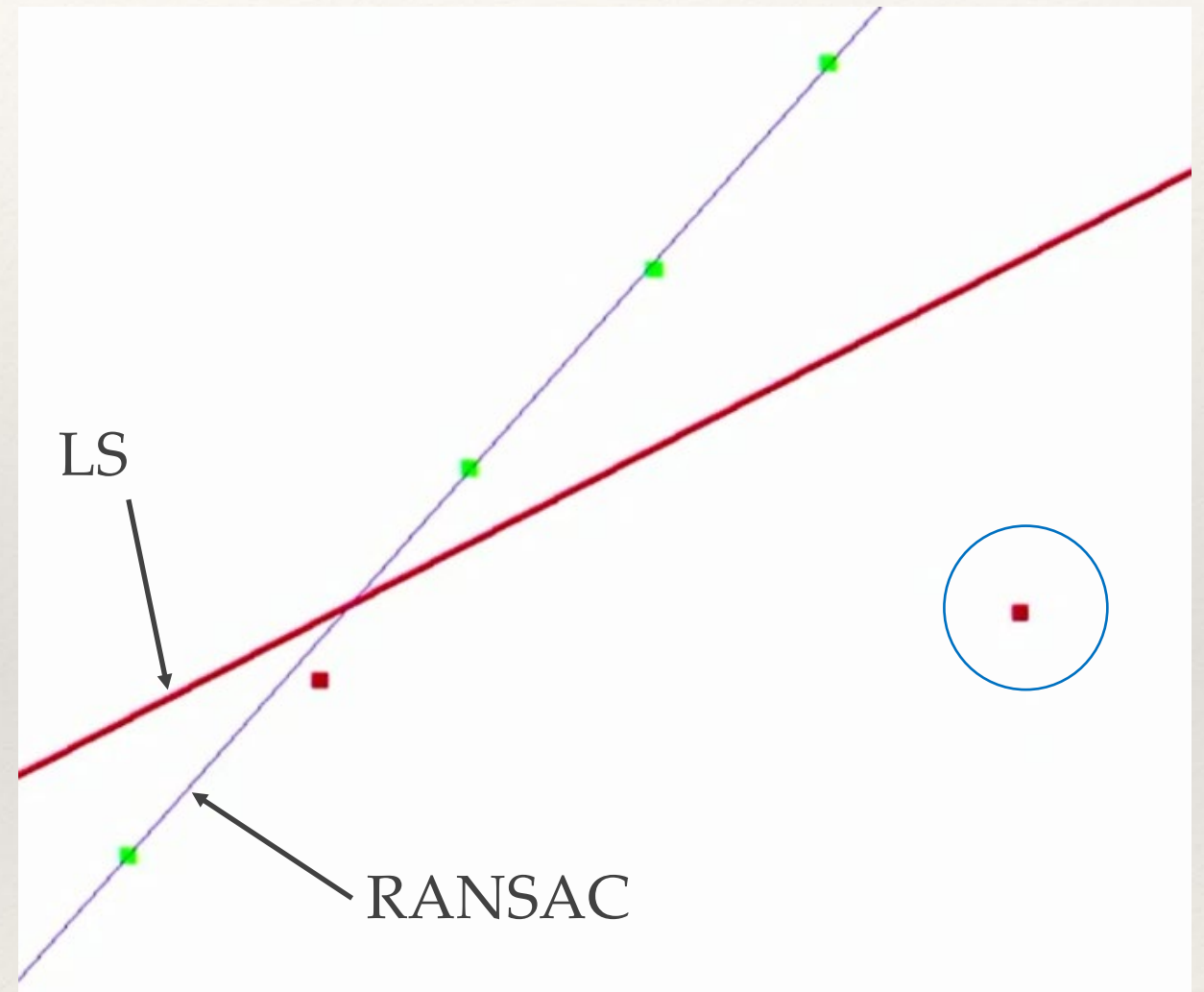
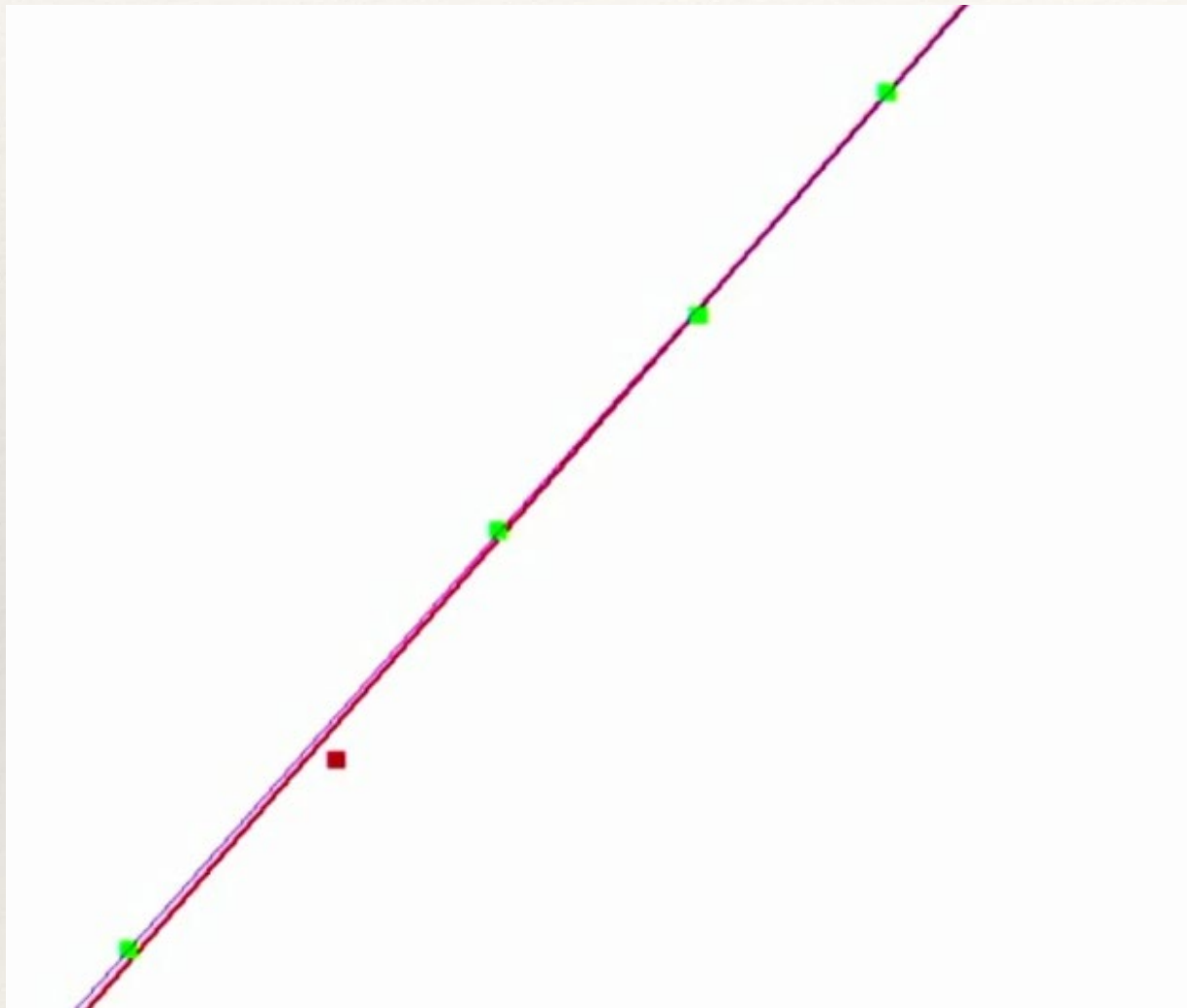
3.) find how many of the N data items fit T within tolerance Tol , call this K (i.e. compute how many times the absolute residual is less than Tol). The points that have an absolute residual less than Tol are the inliers; the other points are the outliers.

4.) if K is large enough, either accept T or compute the least-squares estimate using all inliers, and exit with success.

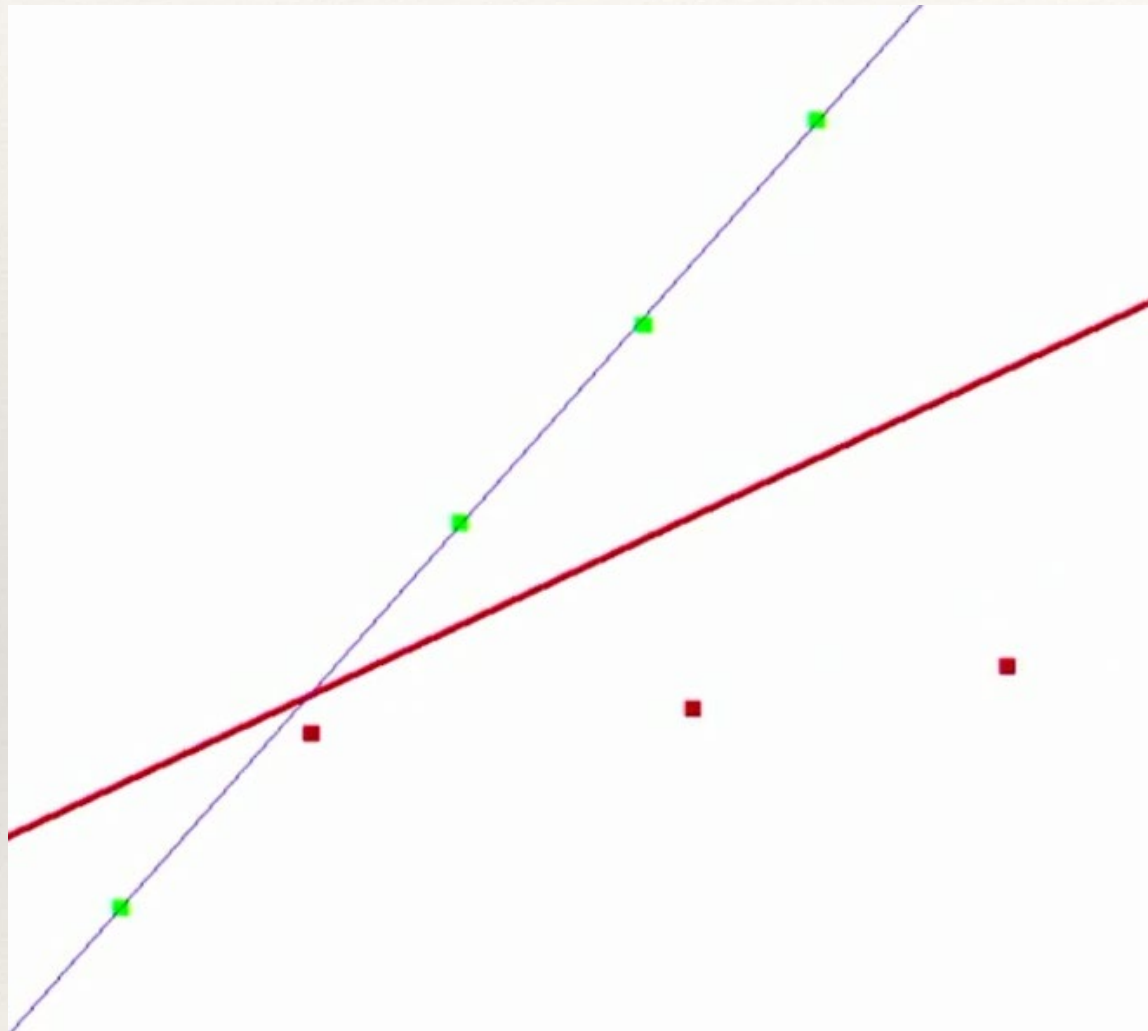
5.) repeat steps 1..4 for n Iterations

 6.) fail - no good T fit of data

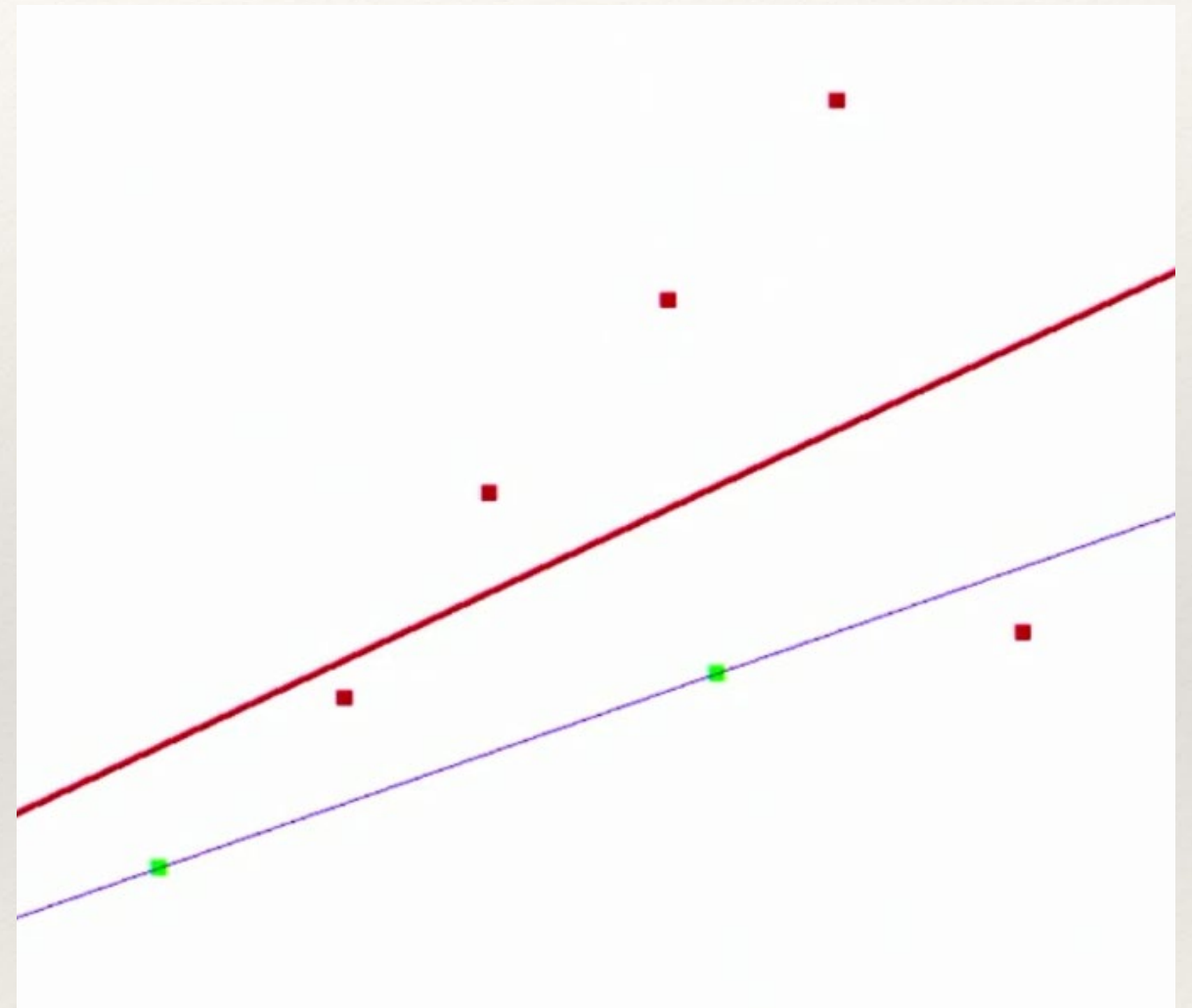
RANSAC vs Least-squares



RANSAC vs Least-squares



Case 1



Case 2

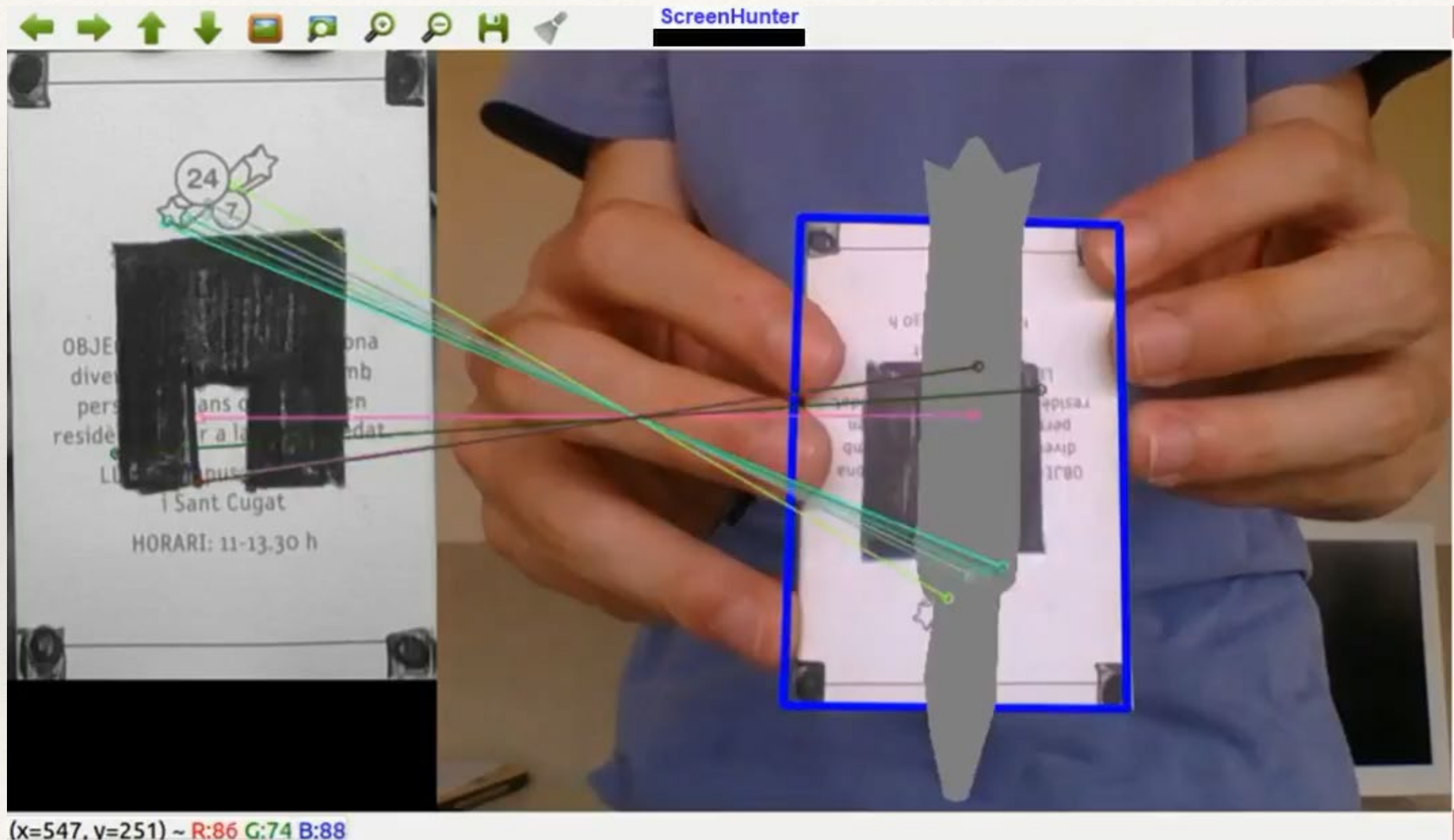


Further applications of robust local matching

Object recognition & AR

- ❖ Object recognition
 - ❖ Image of object is matched against scene, and recognised if there is a consistent match
- ❖ Augmented reality
 - ❖ Data can be added to a scene on the basis of a match

Object recognition & AR



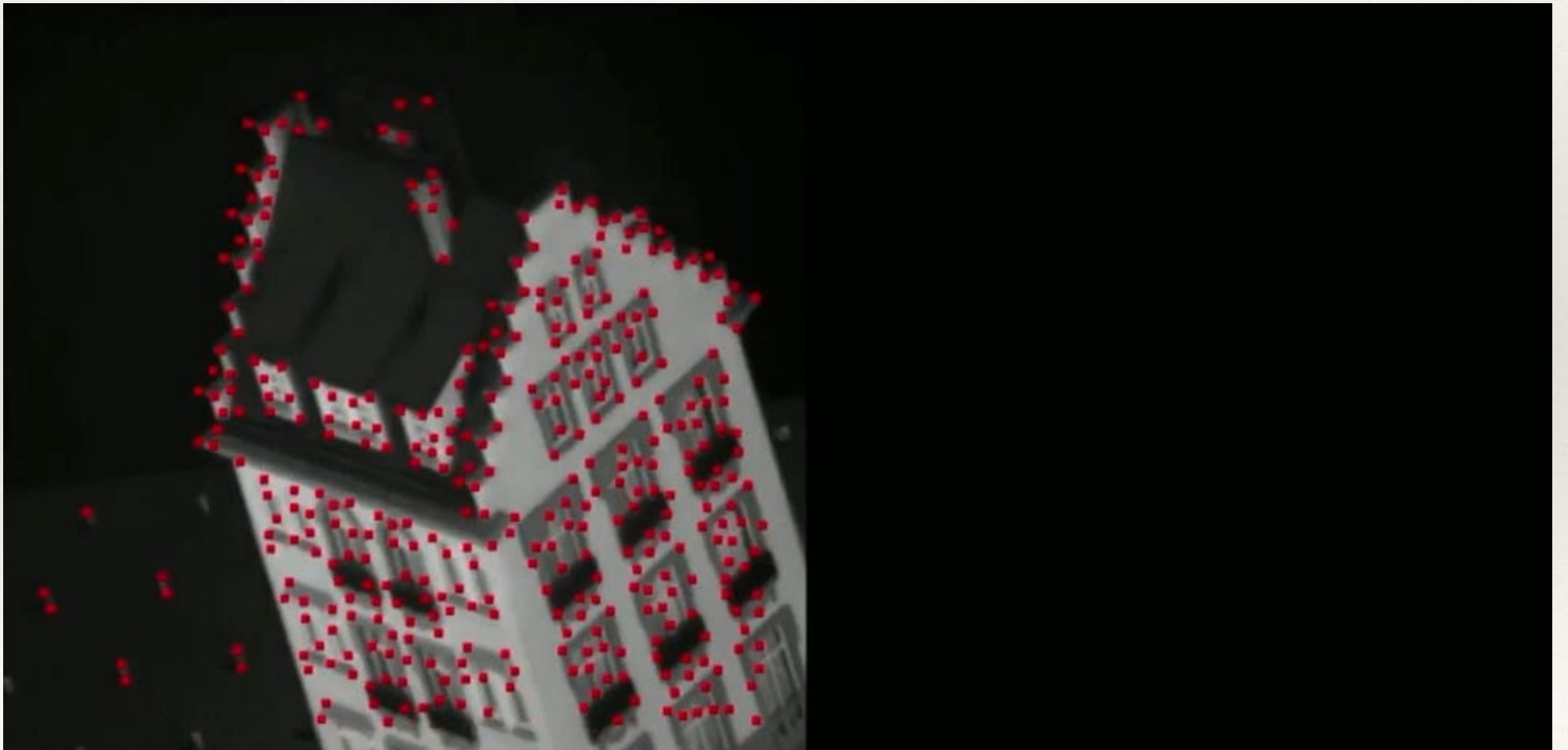
https://youtu.be/FGKkIr_Ily4

3D reconstruction

- ❖ It's possible to estimate depth, and ultimately build a complete 3D scene from sets of point correspondences formed from matching local features

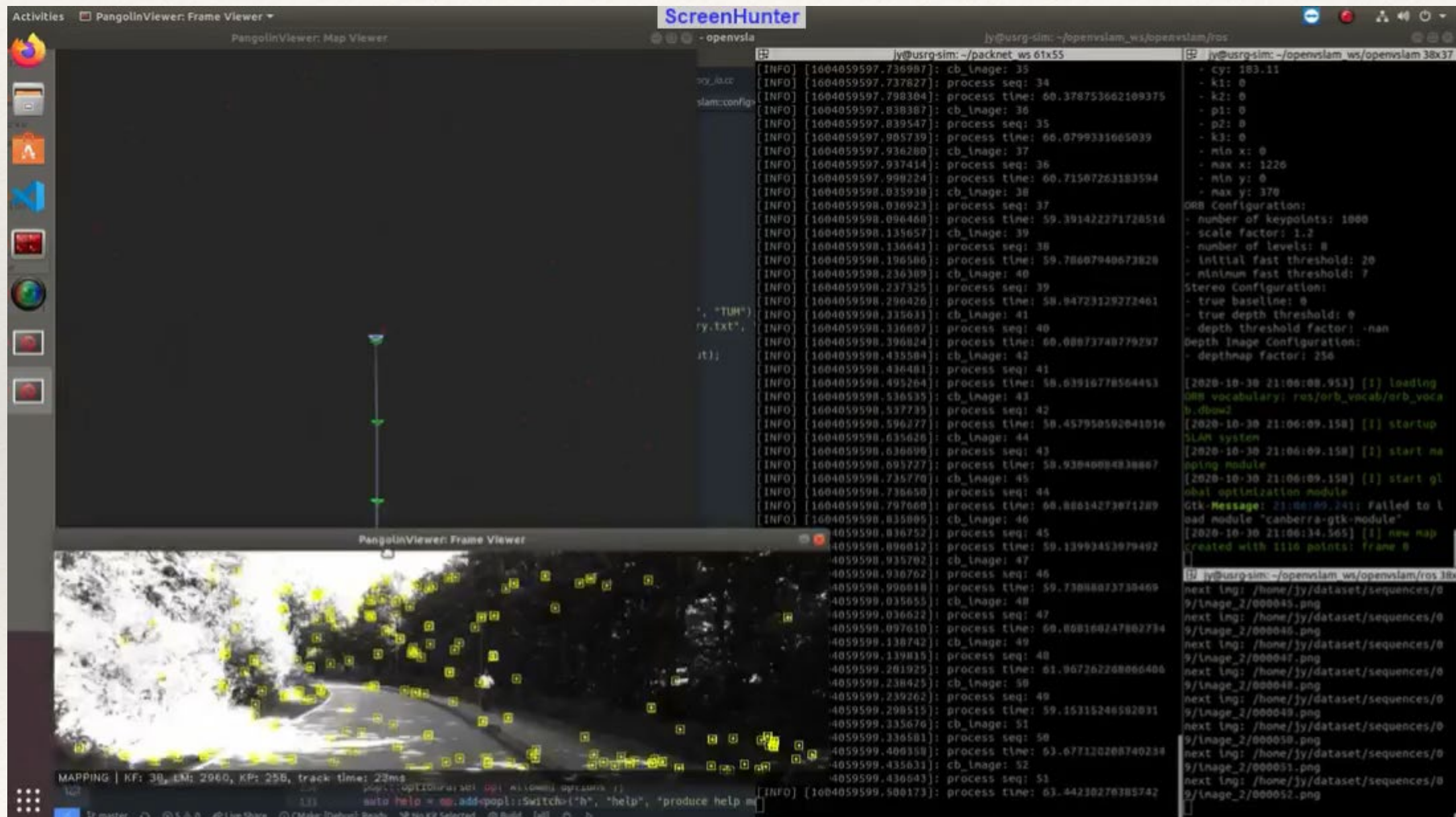
3D reconstruction

- ❖ Real-time 3D points reconstruction



3D reconstruction

- ❖ SLAM (simultaneous localization and mapping)



https://youtu.be/QPy7ygm_McE

Matching music/sounds

- ❖ Image features can be used to match music!
- ❖ That's how systems like Shazam work
- ❖ A recording of a piece of music can be turned into a picture called a spectrogram.
- ❖ Local image features can be extracted, described and matched from the spectrogram images



Spectrogram Demo

- ❖ <https://musiclab.chromeexperiments.com/Spectrogram/>

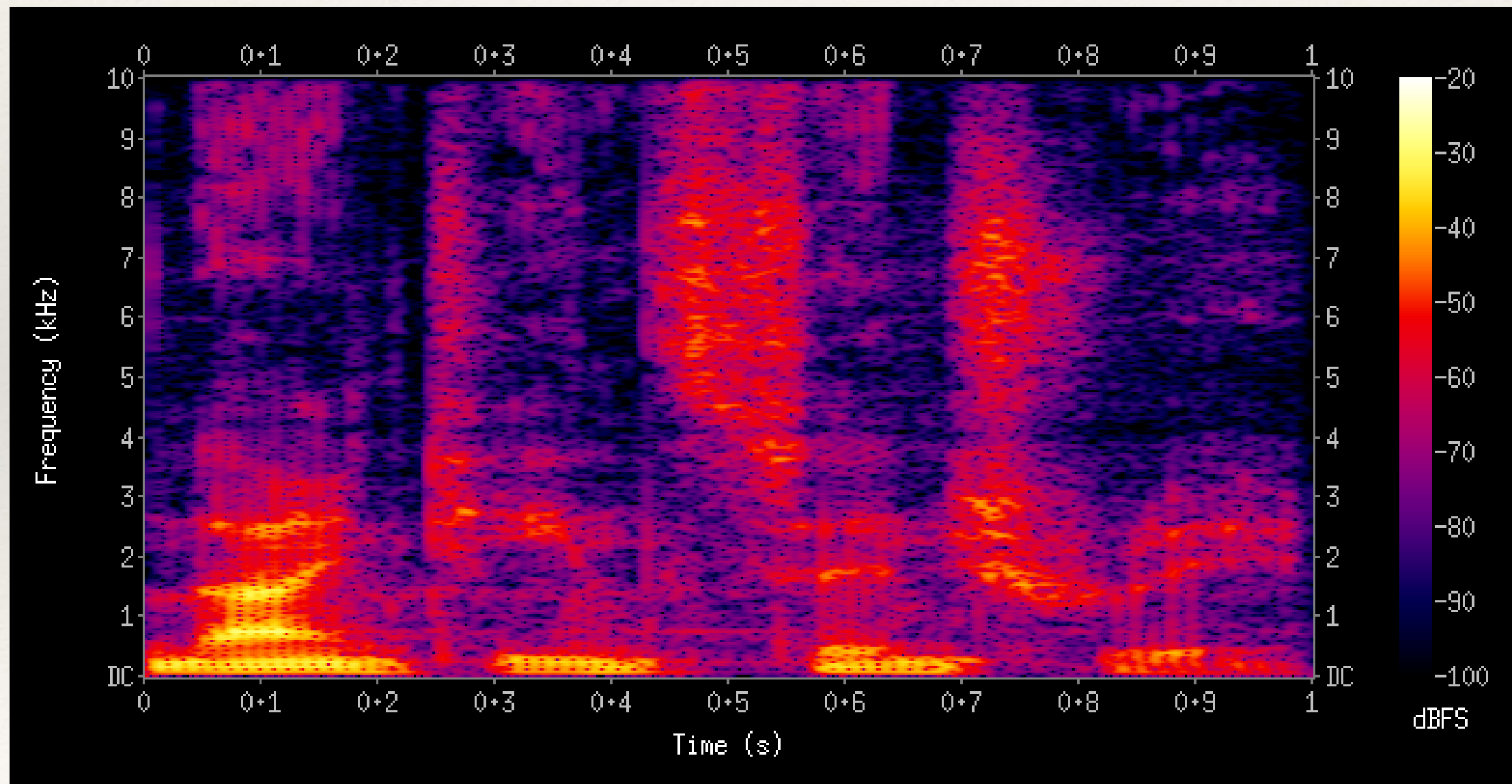


Image from Wikipedia

Problems with direct local feature matching

Local feature matching is slow!

- ❖ Typical image (800x600) might have ~2000 DoG Interest points/SIFT descriptors
- ❖ Each SIFT descriptor is 128 dimensions
- ❖ Now assume you want to match a query image against a database of images...
 - ❖ The distance between every query feature and every other features needs to be calculated
 - ❖ Can this be optimised somehow?



Efficient Nearest Neighbour Search

- ❖ How can we **quickly** find the nearest neighbour to a query point in a high dimensional space?
 - ❖ Index the points in some kind of tree structure?
 - ❖ Hash the points?
 - ❖ Quantise the space (*more on this next time*)

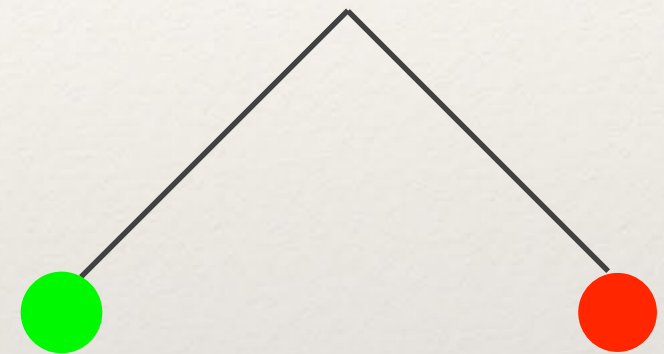
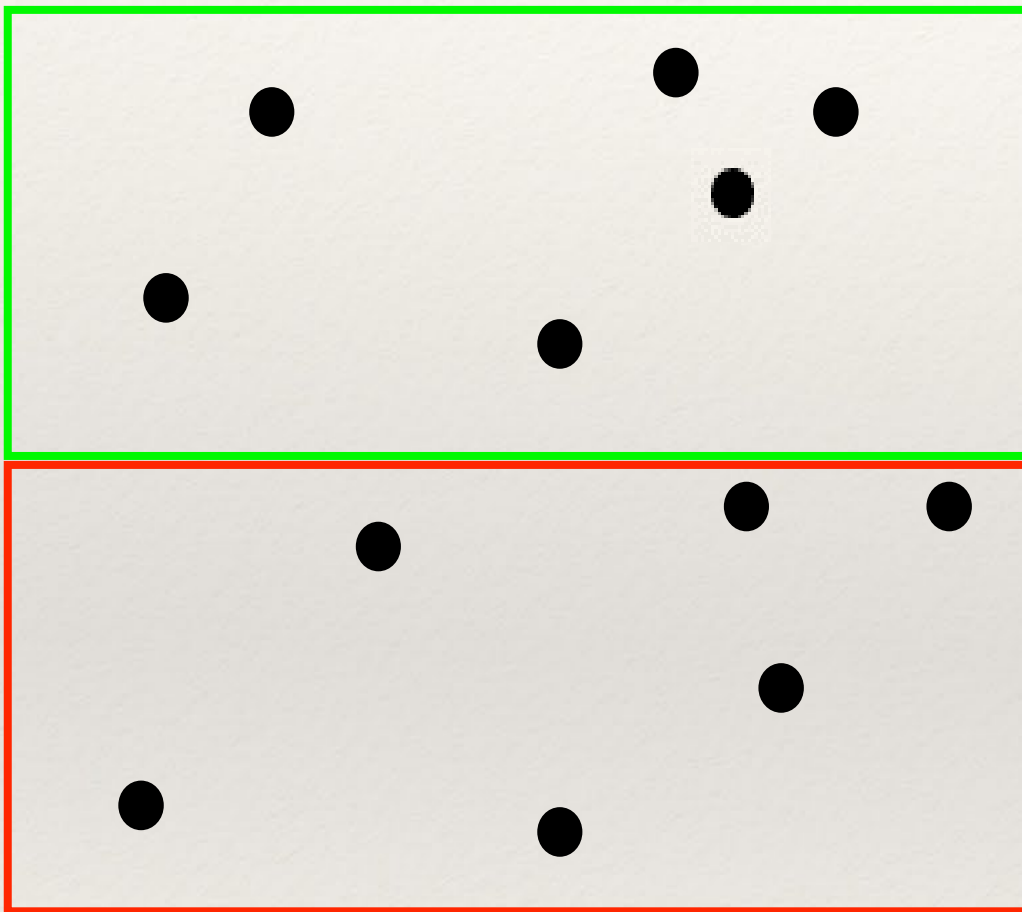


K-D Trees

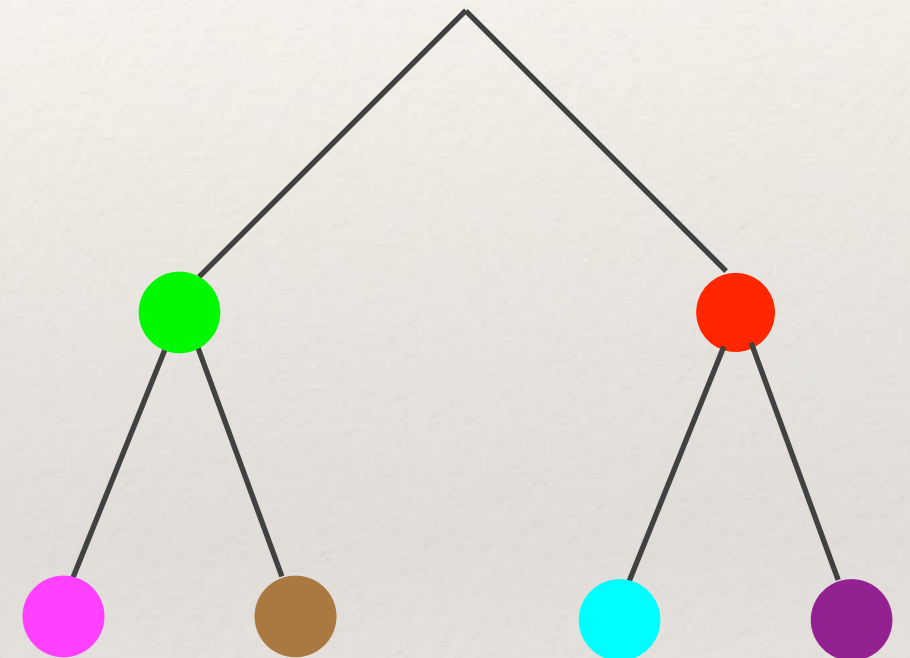
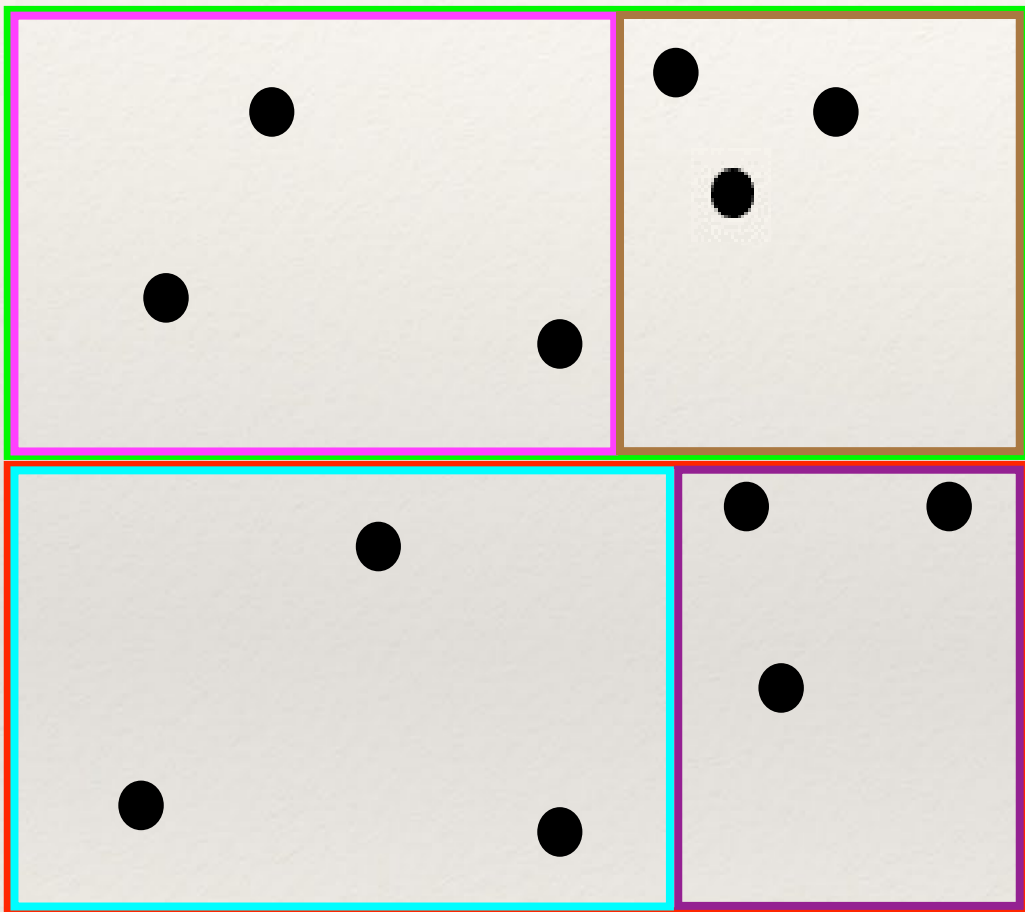
- ❖ Binary tree structure that partitions the space along axis-aligned hyperplanes
- ❖ Typically take each dimension in turn and splits on the median of the points in the enclosing partition.
- ❖ Stop after a certain depth, or when the number of points in a leaf is less than a threshold



K-D Tree



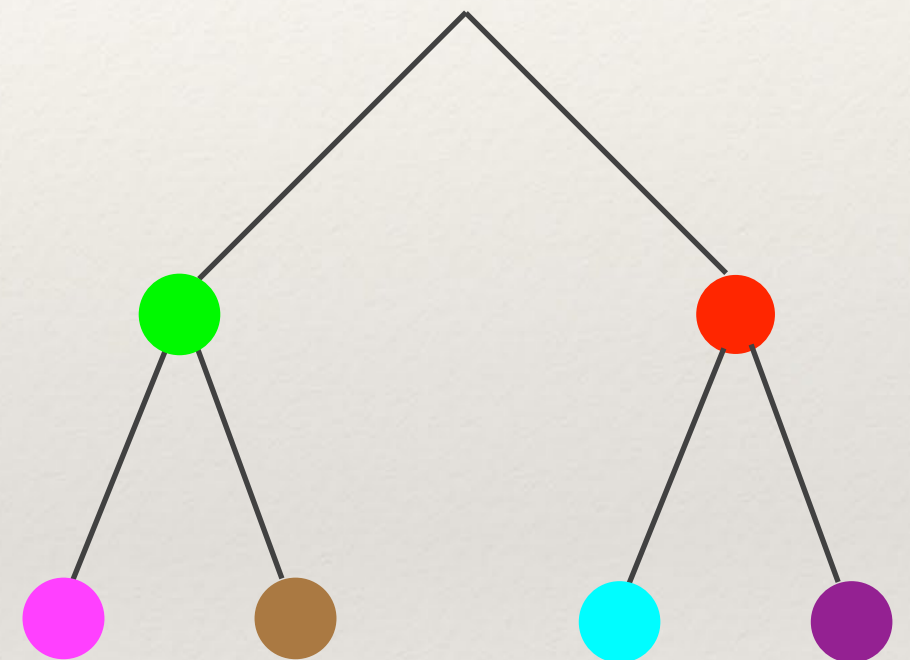
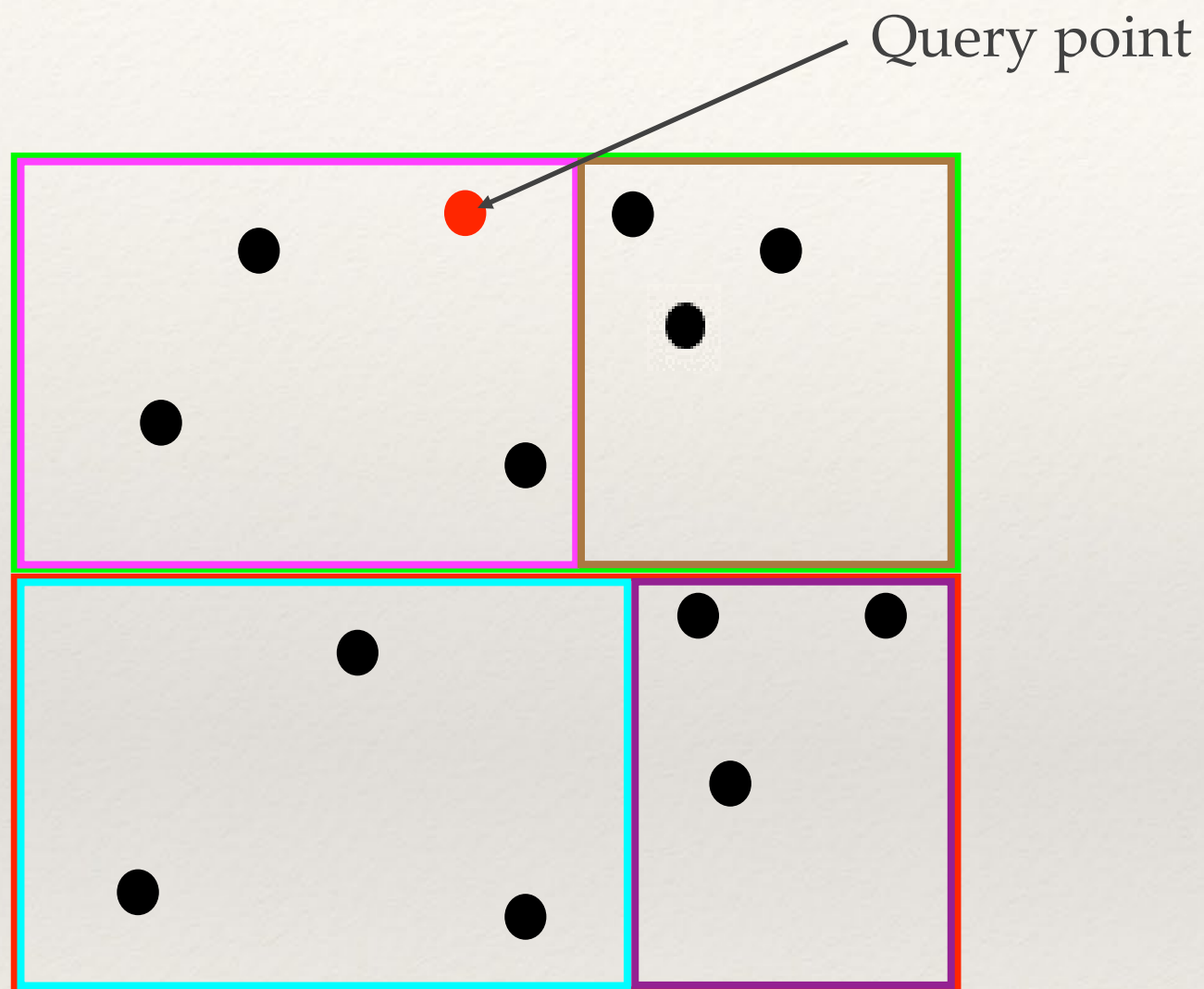
K-D Tree



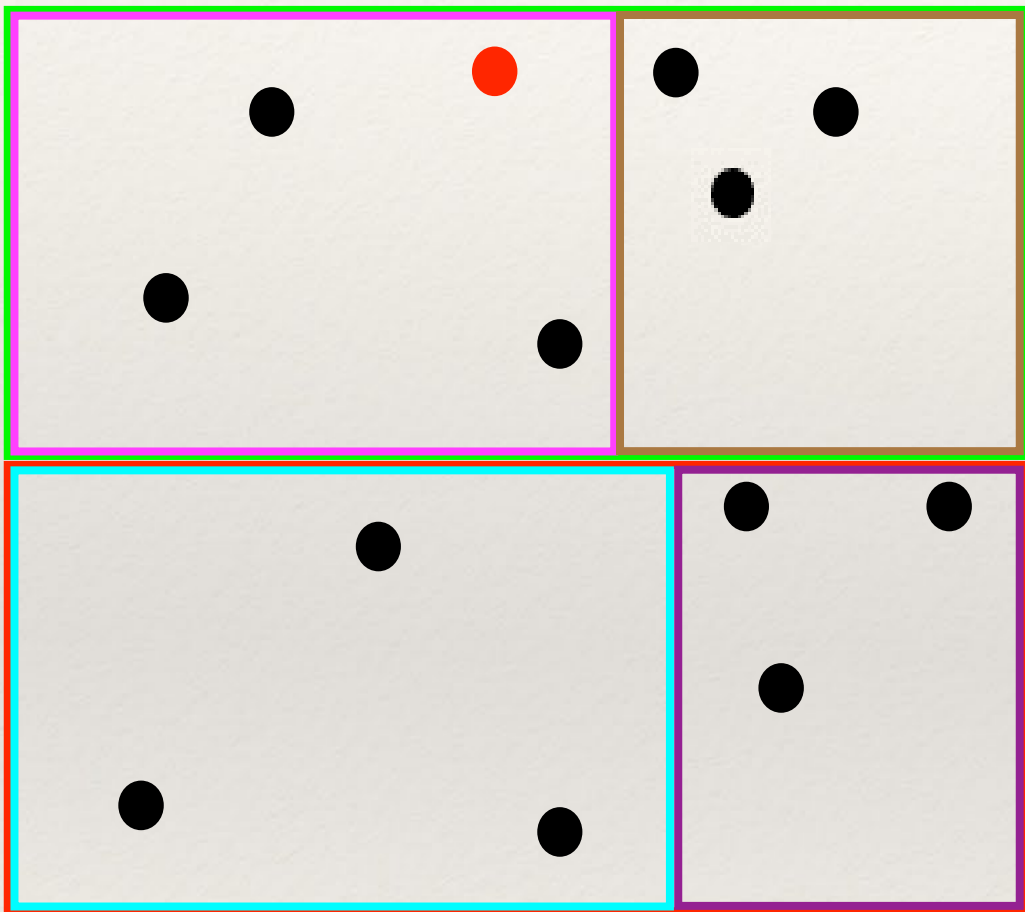
K-D Trees

- ❖ Search by walking down the tree until a leaf is hit, and then brute-force search to find the best in the leaf.
- ❖ This is not guaranteed to be the best though...
- ❖ To have to walk back up the tree and see if there are any better matches, and only stop once the root is reached (note you don't have to check a subtree if it's clear that all points in that subtree are further than the current best).

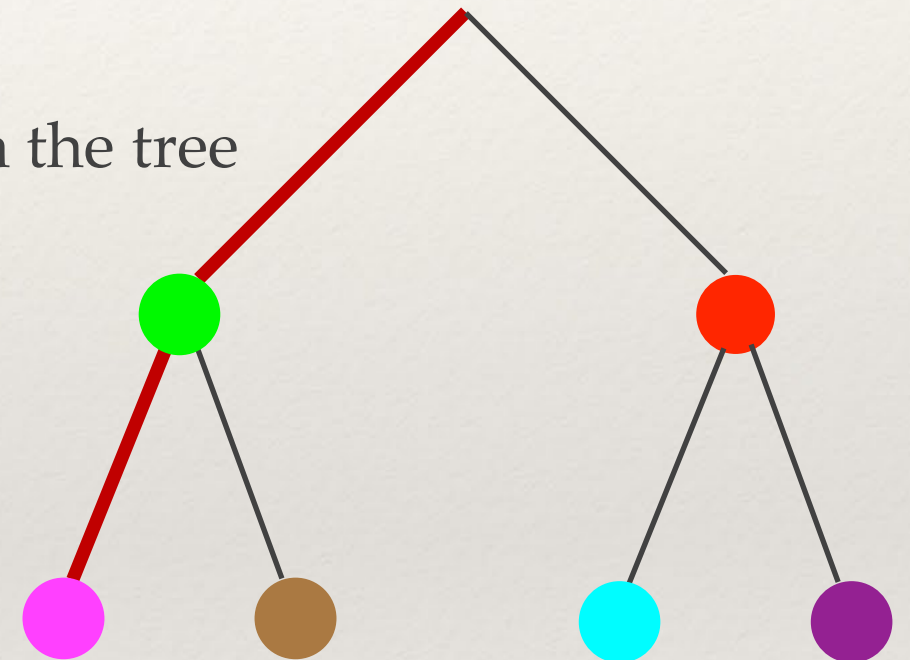
K-D Tree



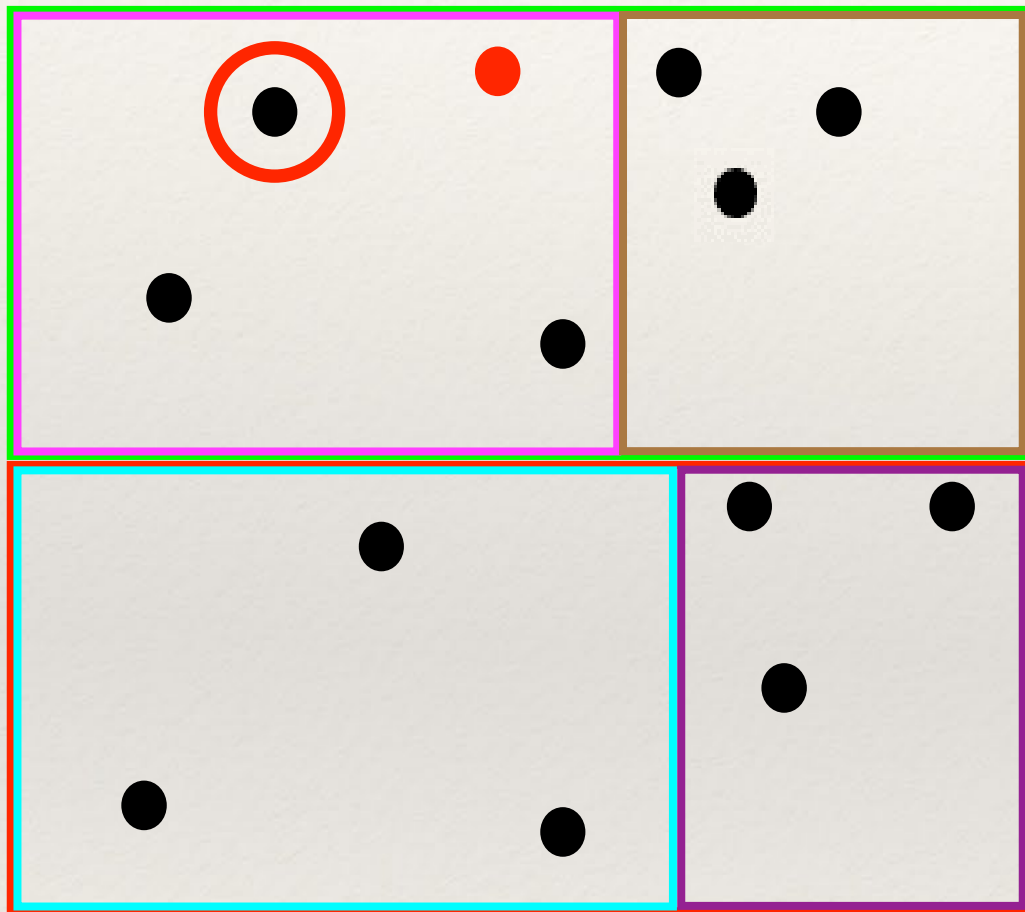
K-D Tree



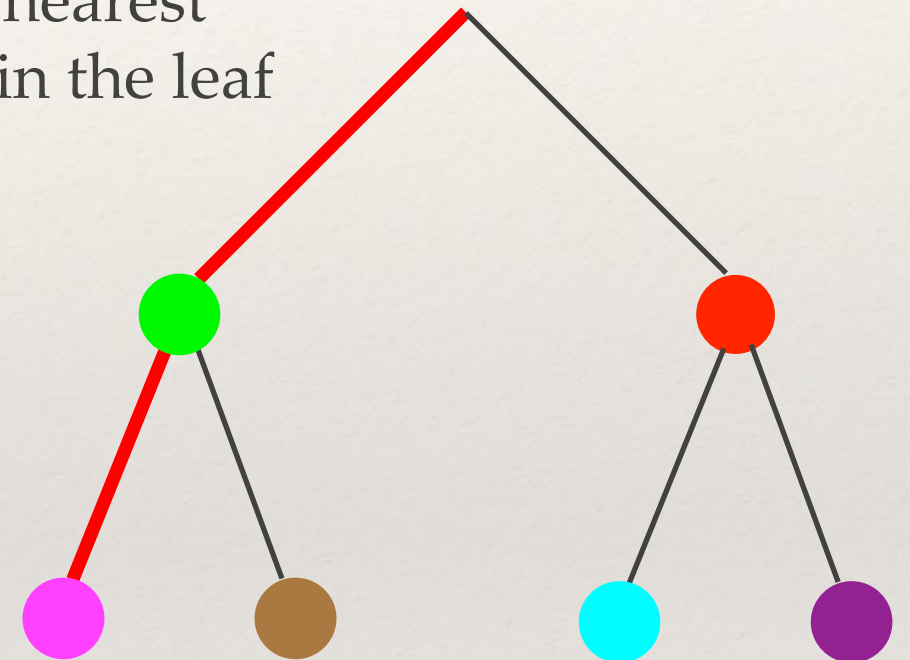
Walk down the tree



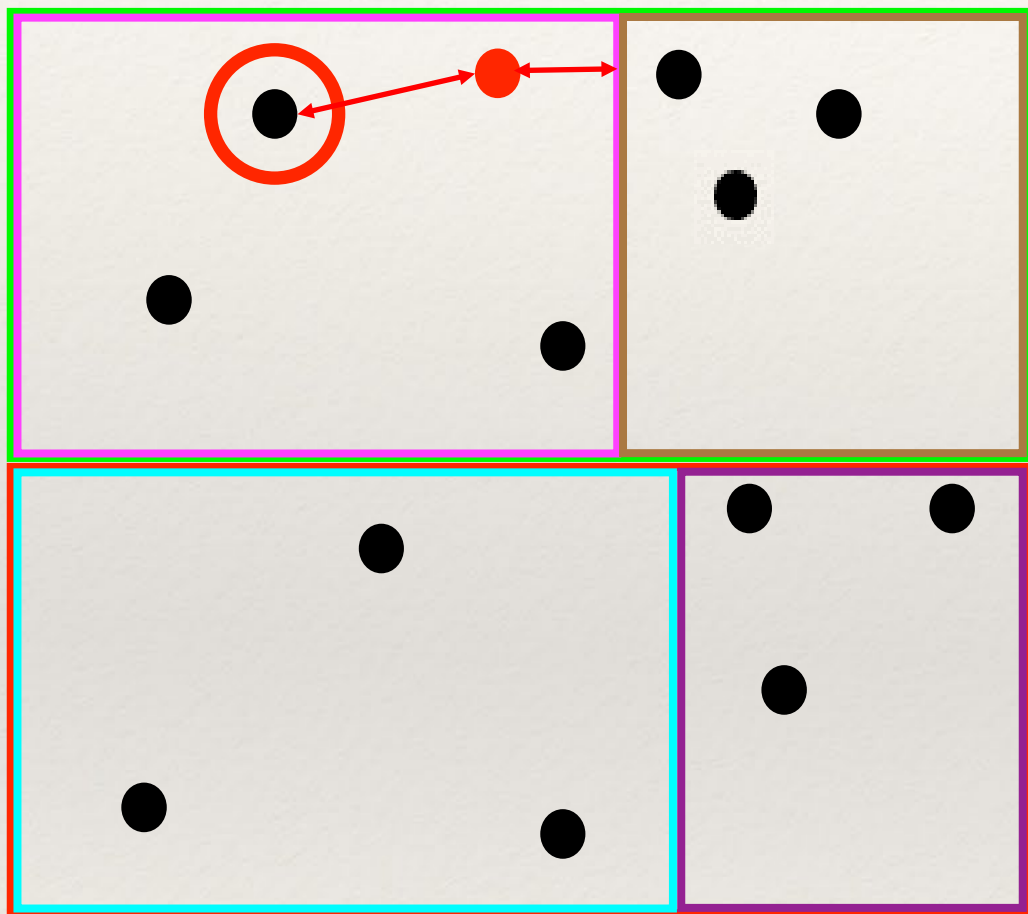
K-D Tree



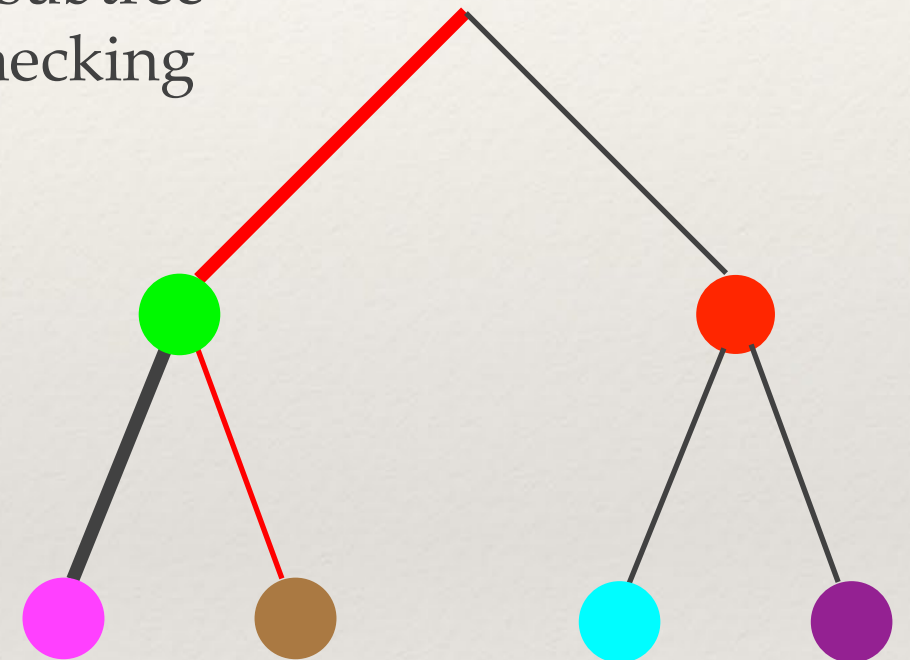
Find the nearest
neighbour in the leaf



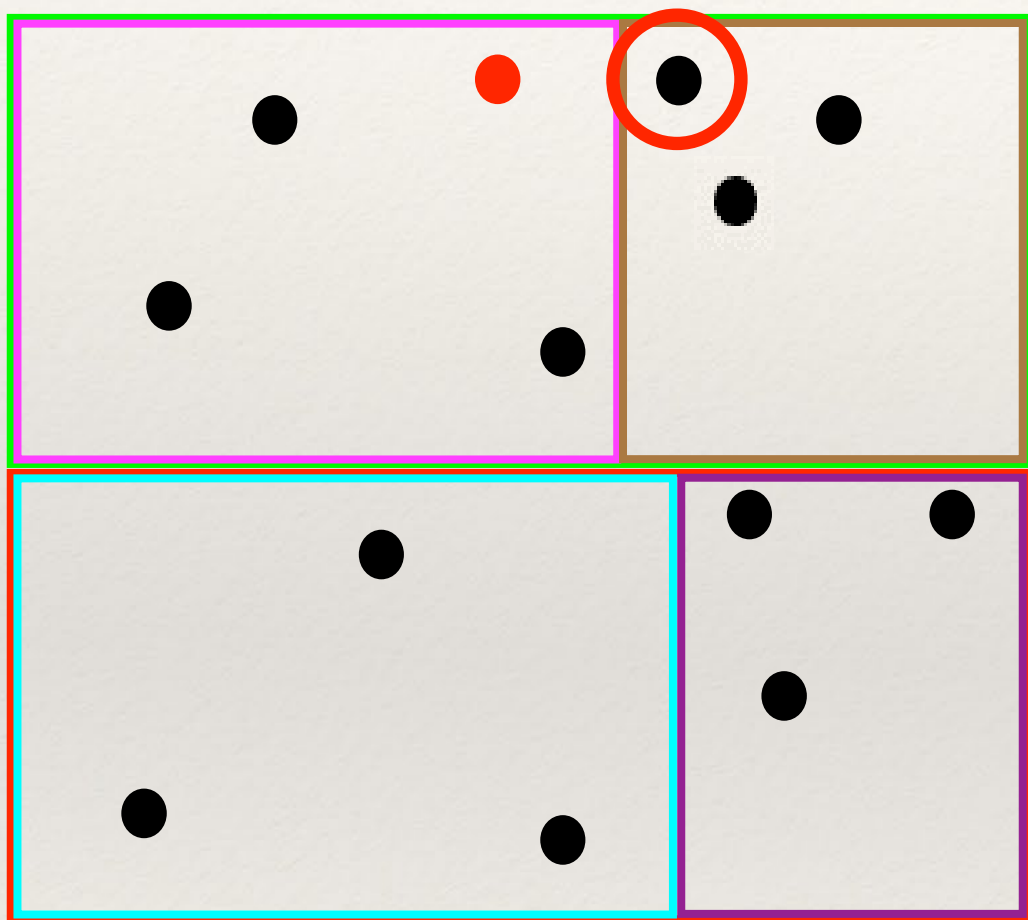
K-D Tree



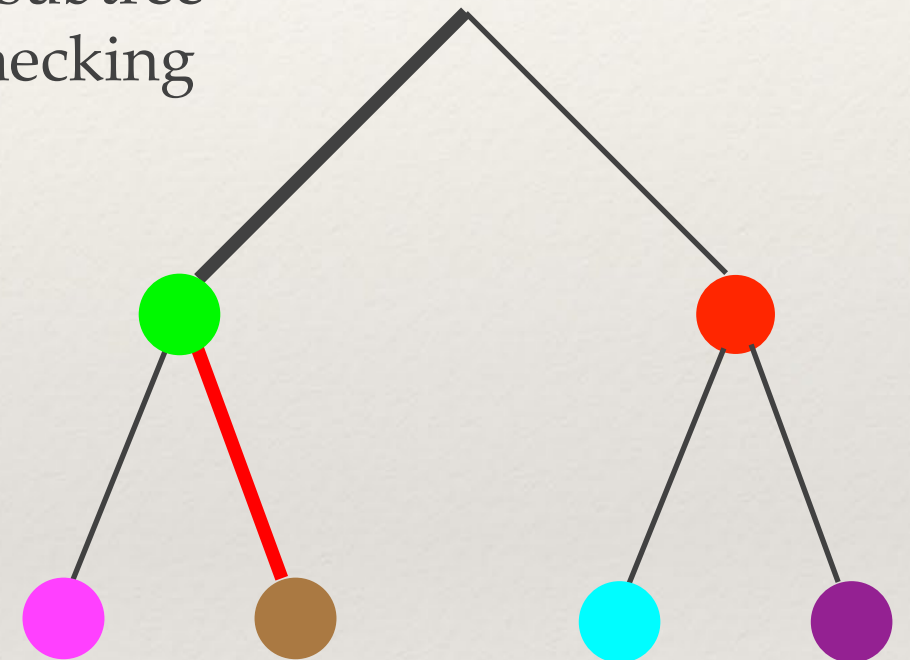
Backtrack, and see if
the next subtree
needs checking



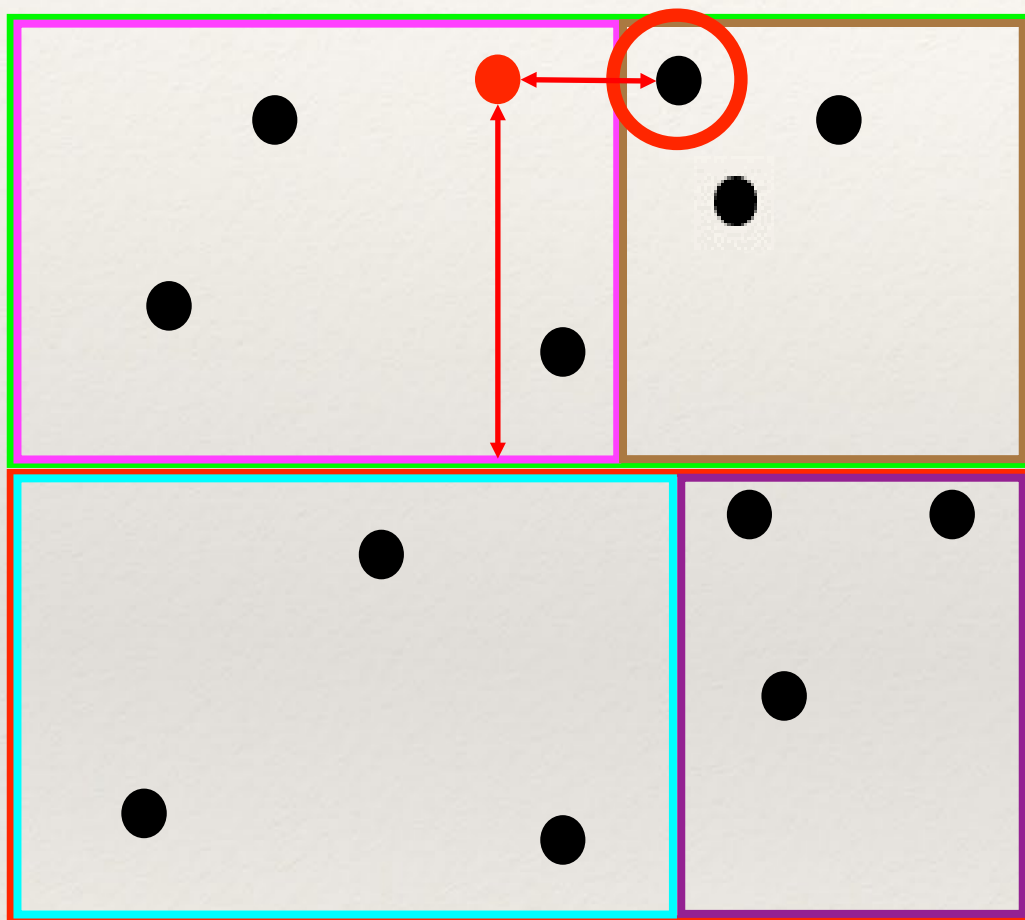
K-D Tree



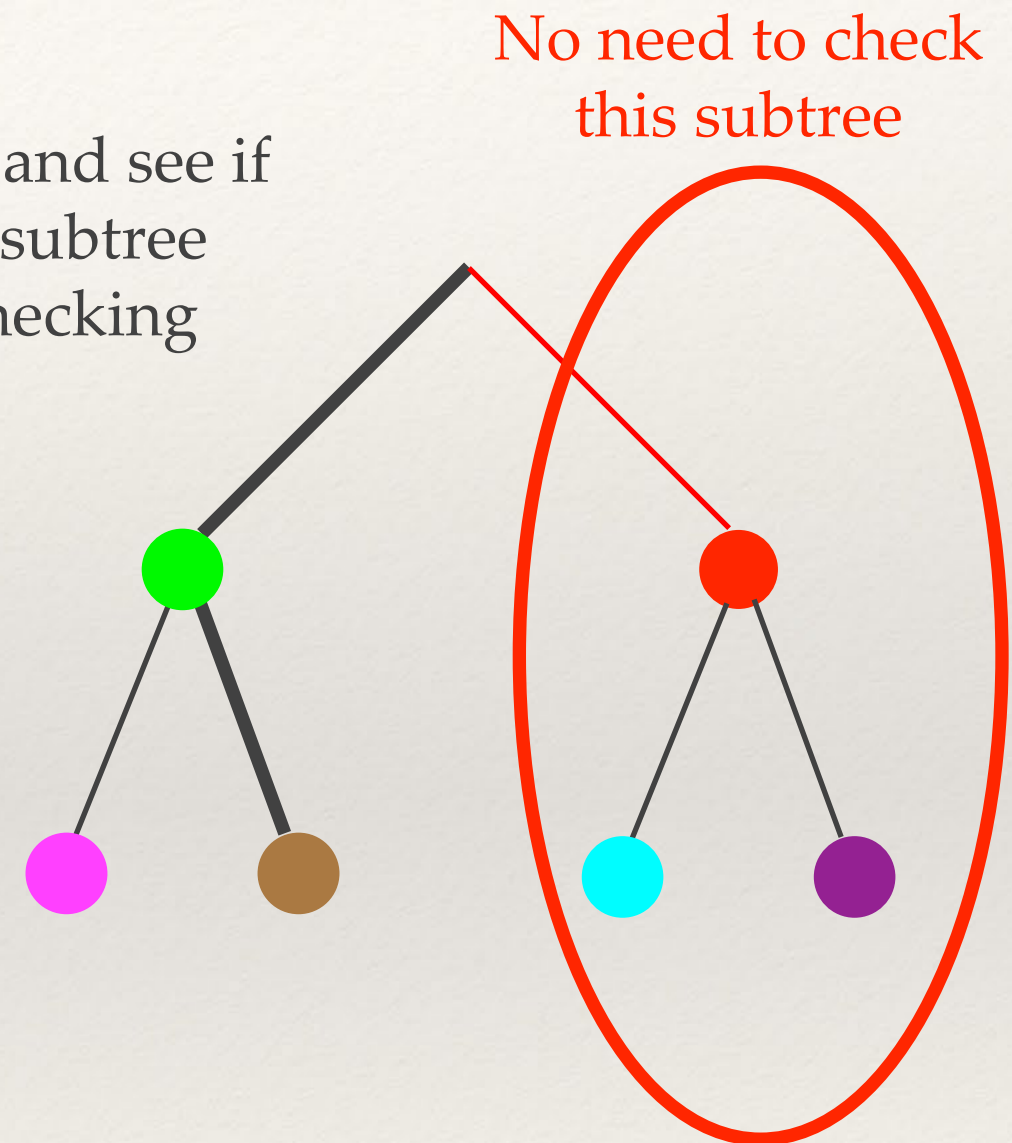
Backtrack, and see if
the next subtree
needs checking



K-D Tree



Backtrack, and see if
the next subtree
needs checking



K-D Tree problems

- ❖ Doesn't scale well to high dimensions
 - ❖ You tend to end up needing to search most of the tree
- ❖ There are *approximate* versions that won't necessarily return the exact answer (if you don't mind the potential for mismatch)



Hashing

- ❖ Locality Sensitive Hashing (LSH) creates hash codes for vectors such that similar vectors have similar hash codes

Sketching

- ❖ A technique called sketching concatenates binary hashes into a bit string.
- ❖ With the correct LSH function, the Hamming distance between a pair of sketches is proportional to the Euclidean distance between the original vectors
- ❖ Can easily compress SIFT features to 128 bits
 - ❖ Hamming distance computation is cheap
 - ❖ Lookup tables and bitwise operations

Summary

- ❖ Inconsistent local feature matches can be removed by assuming some form of constraint holds between the two images
 - ❖ This is usually a geometric mapping
 - ❖ Affine transform or Homography
 - ❖ Can be estimated by finding the least-squares solution of a set of simultaneous equations
 - ❖ Robust methods such as RANSAC allow inliers and outliers to be determined whilst learning the mapping
- ❖ Interest point matching is slow...
 - ❖ K-D Trees and Hashing can help

Further reading and exercises

❖ Further reading

- ❖ The original RANSAC paper: <http://www.ai.sri.com/pubs/files/836.pdf>
- ❖ Wikipedia page on k-d trees and locality sensitive hashing
http://en.wikipedia.org/wiki/K-d_tree
http://en.wikipedia.org/wiki/Locality-sensitive_hashing

❖ Practical exercises

- ❖ Geometric transform
https://docs.opencv.org/4.5.5/dd/d52/tutorial_js_geometric_transformations.html
- ❖ Augmented reality with Python and OpenCV
<https://bitesofcode.wordpress.com/2017/09/12/augmented-reality-with-python-and-opencv-part-1/>
- ❖ Chapter 5 of the OpenIMAJ tutorial covers fitting an affine transform to a set of point correspondences using RANSAC