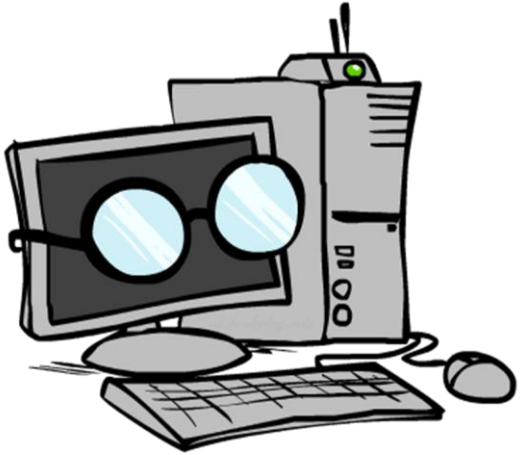
COMS W4733: Computational Aspects of Robotics

Lecture 24: Computer Vision 2



Instructor: Tony Dear

Pose Estimation

- Given a image output from a camera, estimate positions and orientations of objects (including the robot itself) in the world

- Two problems here: Object is first viewed from different distances or angles
- 3D pose then transformed into 2D image

- Assume that we know the object's model a priori
- Rigid bodies, so 6 DOF to figure out
- Also know *feature correspondences*

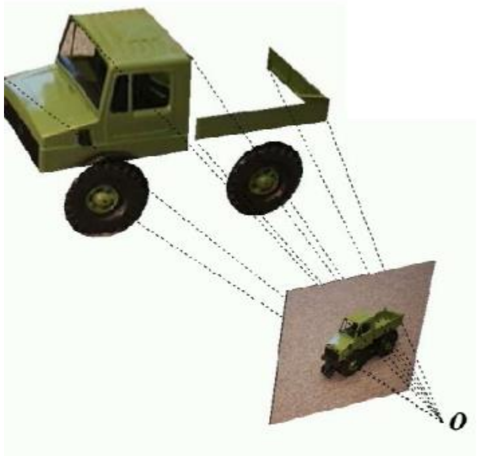


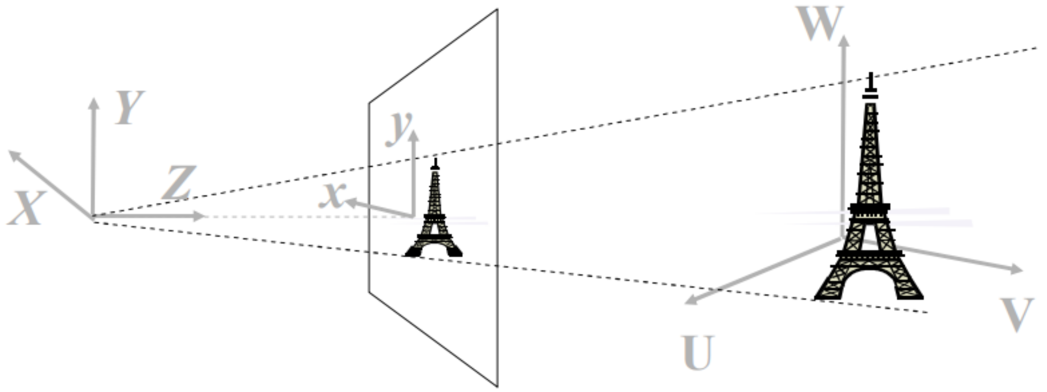
Image Transformations

$$\begin{pmatrix} x \\ y \end{pmatrix} \longleftarrow \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \longleftarrow \begin{pmatrix} U \\ V \\ W \end{pmatrix}$$

2D image in camera coordinates

3D object in camera coordinates

3D object in world coordinates



Calibration Matrix

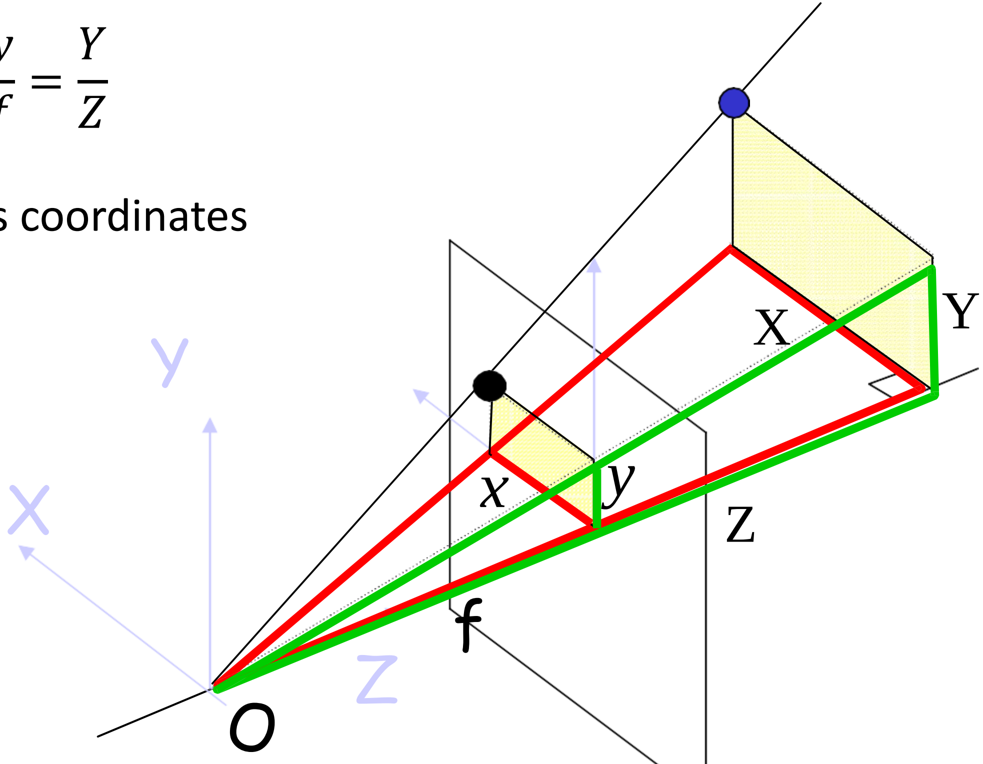
Using similar triangles: $\frac{x}{f} = \frac{X}{Z} \quad \frac{y}{f} = \frac{Y}{Z}$

Define (x', y', z') as 2D homogeneous coordinates

$$x = \frac{fX}{Z} = \frac{x'}{z'} \qquad y = \frac{fY}{Z} = \frac{y'}{z'}$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Calibration matrix ***K***



Focal length f

Solving for Pose

$$\begin{pmatrix}x\\y\end{pmatrix} \leftarrow \begin{pmatrix}X\\Y\\Z\end{pmatrix} \leftarrow \begin{pmatrix}U\\V\\W\end{pmatrix}$$

2D image in camera coordinates	3D object in camera coordinates	3D object in world coordinates

- Suppose we have already performed calibration (we know $\boldsymbol{K}$)
- Second problem is now to figure out mapping between two 3D frames
- Composite mapping brings 3D object to 2D image
- “Inverting” this mapping solves problem of solving for the pose

Full Camera Matrix

- The full **camera matrix** is the concatenation of both operations together

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- We want to solve for $\boldsymbol{R}$ and $\boldsymbol{t}$!
- In general 6 unknowns (3 rotation, 3 translation), so we need at least 6 data
  - We can thus get away with 3 correspondences, since each contributes 2 equations
- System of nonlinear equations, can be solved either analytically or numerically

Perspective-n-Point Problem

- What happens if we "invert" the calibration matrix K?

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \sim \begin{pmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- This corresponds to mapping from a 2D point to a corresponding 3D point
- There is ambiguity in *depth*; we can scale LHS by an arbitrary constant λ

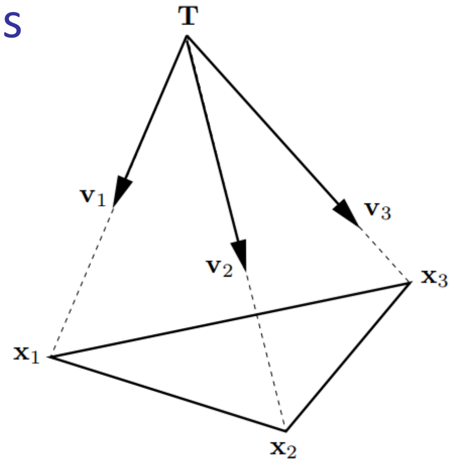
- If we have multiple correspondences, solving for each depth λ_i is equivalent to solving for the pose!

Perspective-n-Point Problem

- **P$\boldsymbol{n}$P** is the general problem of solving for a 3D pose given n correspondences
- Consider simplest case: $n = 3$

- Suppose we obtain points $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$ after mapping through $\boldsymbol{K}^{-1}$
- These points differ from $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ by rigid transformations
- Interpoint distances remain the same!

- We want to find $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ relative to $\boldsymbol{T}$
- It would also suffice to find distances λ_i from $\boldsymbol{T}$ to $\boldsymbol{x}_i$



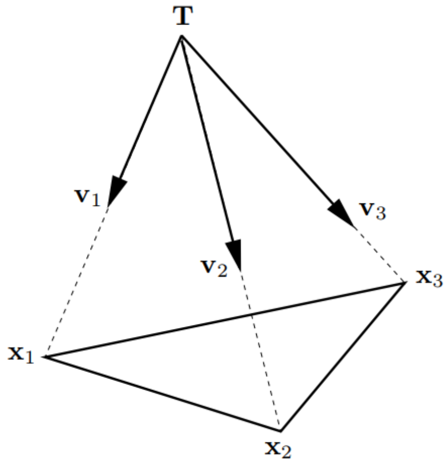
P3P Problem

- Define unknowns $\lambda_1 = \|\boldsymbol{T} - \boldsymbol{x}_1\|, \lambda_2 = \|\boldsymbol{T} - \boldsymbol{x}_2\|, \lambda_3 = \|\boldsymbol{T} - \boldsymbol{x}_3\|$

- Apply law of cosines to the three triangles containing T:

$$\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2 = \lambda_1^2 + \lambda_2^2 - 2\lambda_1\lambda_2 \cos\angle \boldsymbol{v}_1\boldsymbol{T}\boldsymbol{v}_2$$
$$\|\boldsymbol{x}_2 - \boldsymbol{x}_3\|^2 = \lambda_2^2 + \lambda_3^2 - 2\lambda_2\lambda_3 \cos\angle \boldsymbol{v}_2\boldsymbol{T}\boldsymbol{v}_3$$
$$\|\boldsymbol{x}_1 - \boldsymbol{x}_3\|^2 = \lambda_1^2 + \lambda_3^2 - 2\lambda_1\lambda_3 \cos\angle \boldsymbol{v}_1\boldsymbol{T}\boldsymbol{v}_3$$

- 3 quadratic equations in 3 unknowns
- Can be transformed into a quartic polynomial equation
- Typically multiple (up to 16) solutions



Narrowing Down Solutions

- We discarded a lot of information in going from solving for rigid transformation to solving for correspondence “depths”
- E.g., constraints such as sign of the distances, determinant of rotation $\boldsymbol{R}$

- If we still have multiple solutions after applying all other world knowledge, use each to project to an unused correspondence
- Keep the one that produces the best result

- If still multiple solutions left, then we need more information

Nonlinear Least Squares

- Solving for transformation can be formulated as a nonlinear least squares optimization problem by minimizing the residual (error)

  $\min \sum_i \|\boldsymbol{Y_i} - (\boldsymbol{R}\boldsymbol{X_i} + \boldsymbol{t})\|^2$ s.t. orthogonality and determinant conditions on $\boldsymbol{R}$

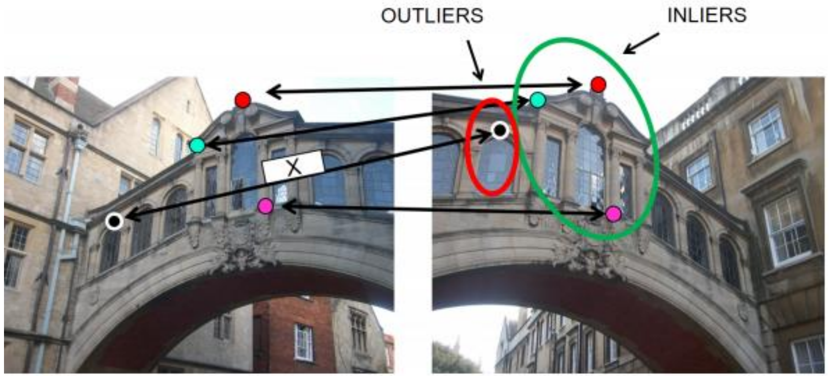
- This optimization problem can be restated as follows: $\max trace(R^T \sum_i \boldsymbol{Y}_i \boldsymbol{X}_i^T)$

- Analytical solution can be found in terms of SVD of above (see notes)
- This is generally more robust than analytical solutions due to presence of noise in images and correspondences

Robust Least Squares

- While optimization algorithms like least-squares can tolerate some noise, outliers (e.g. completely wrong correspondences) can throw them off
- Would be better to somehow find and trust inliers more

- **RANdom SAmple Consensus** (RANSAC): Non-deterministic parameter estimation that (optimally) use inliers only

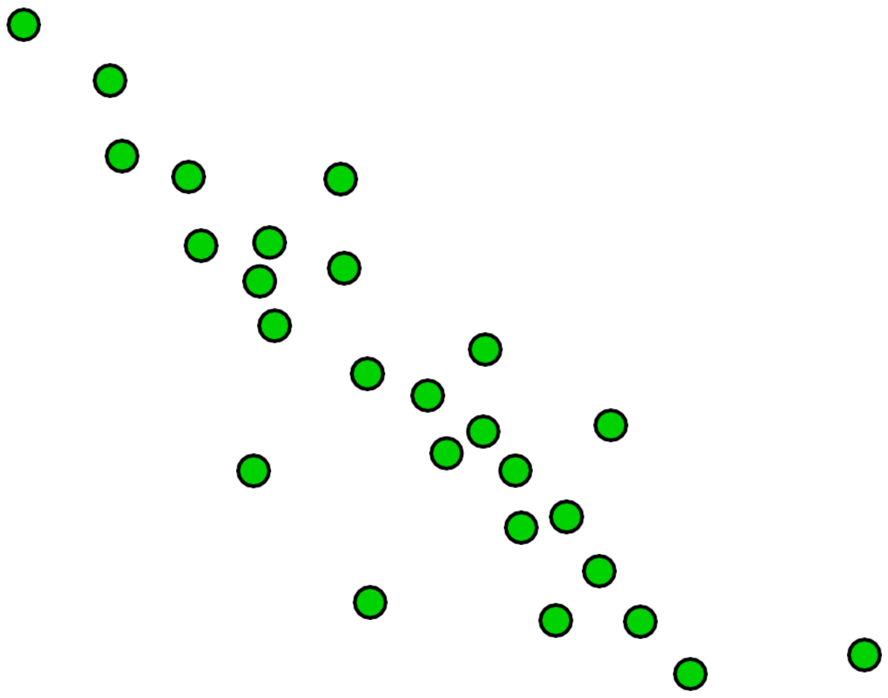


RANSAC

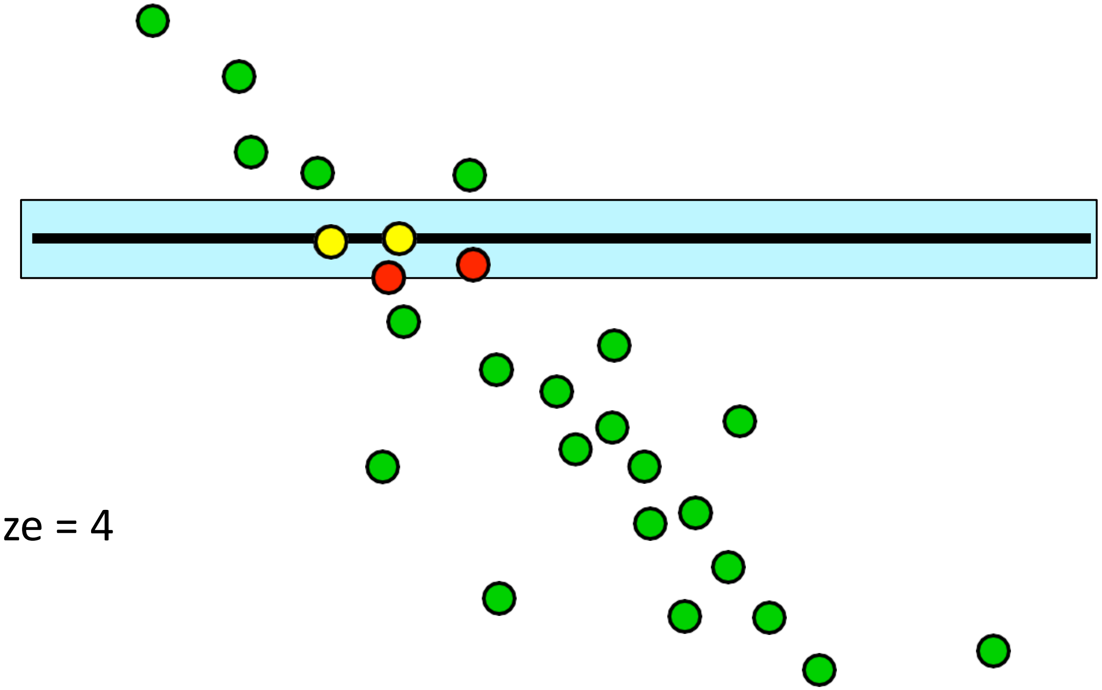
- Randomly subsample original data (e.g. pose correspondences)
- Fit a model to the sampled data only
- Use model to evaluate goodness of fit on all data and assign inliers that fit well (e.g. according to a threshold) to the *consensus set*
- Repeat multiple times and keep the model that has the largest consensus set

- Parameters to consider:
- Number of samples—typically minimum needed to fit a model (e.g. 3 for pose)
- Inlier threshold
- Number of iterations

Example: RANSAC

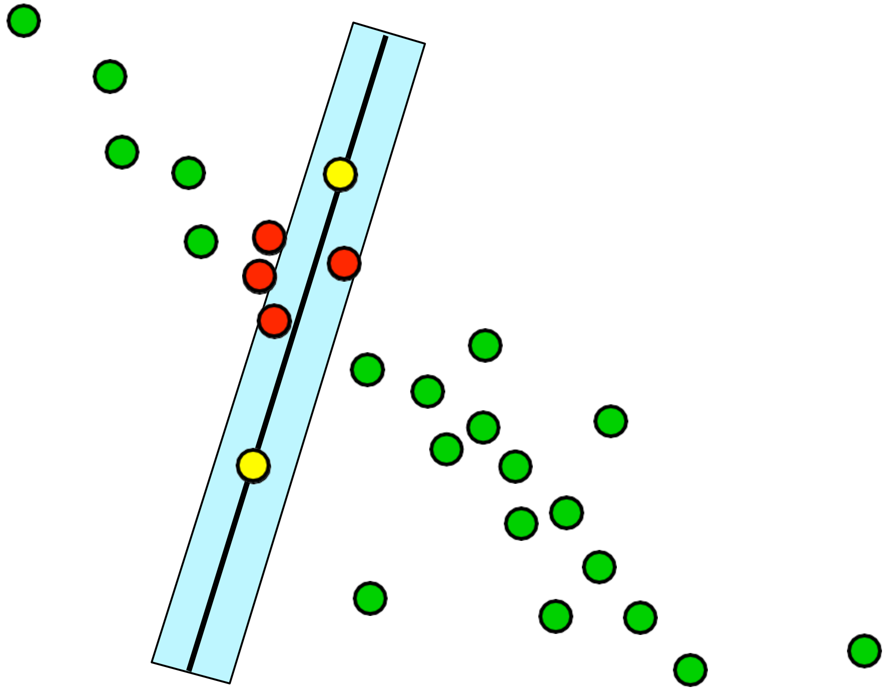


Example: RANSAC



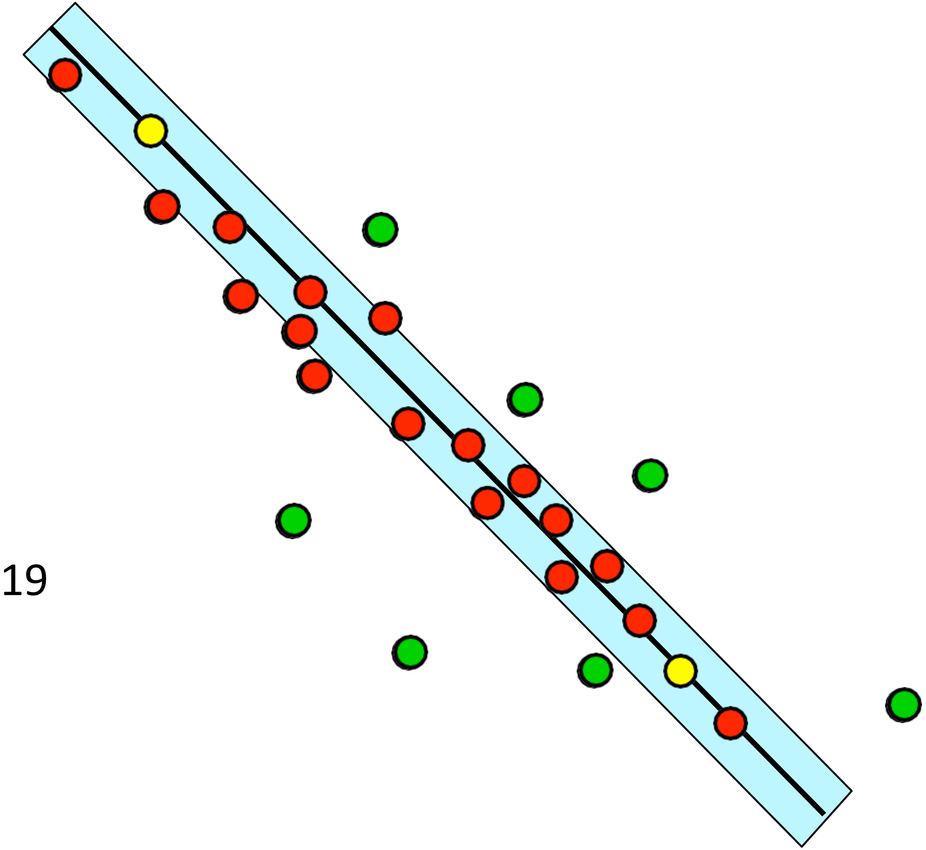
Set size = 4

Example: RANSAC



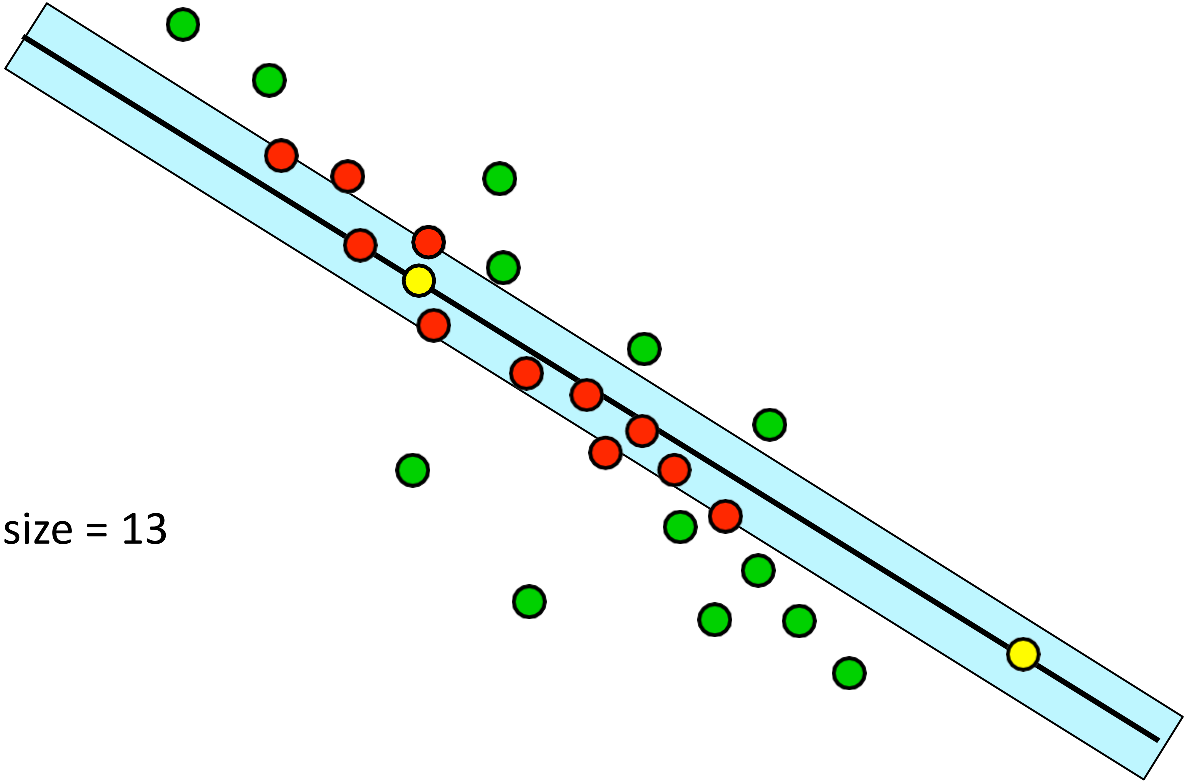
Set size = 6

Example: RANSAC



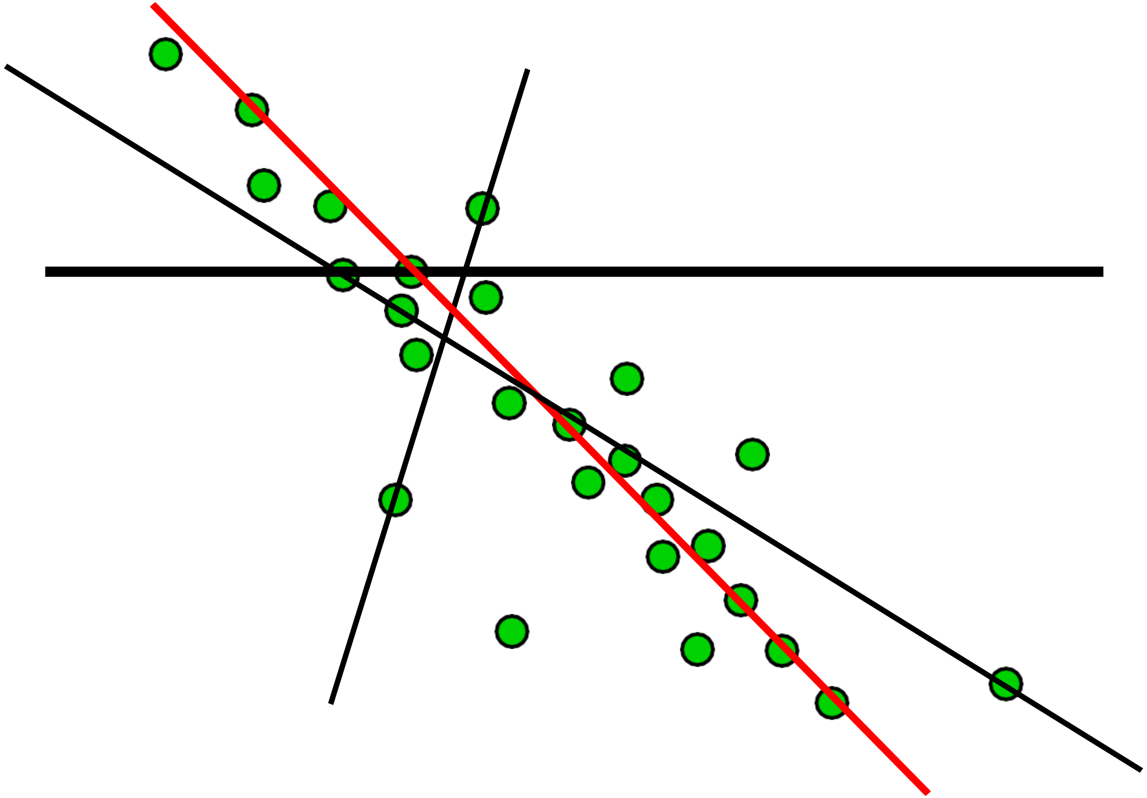
Set size = 19

Example: RANSAC



Set size = 13

Example: RANSAC

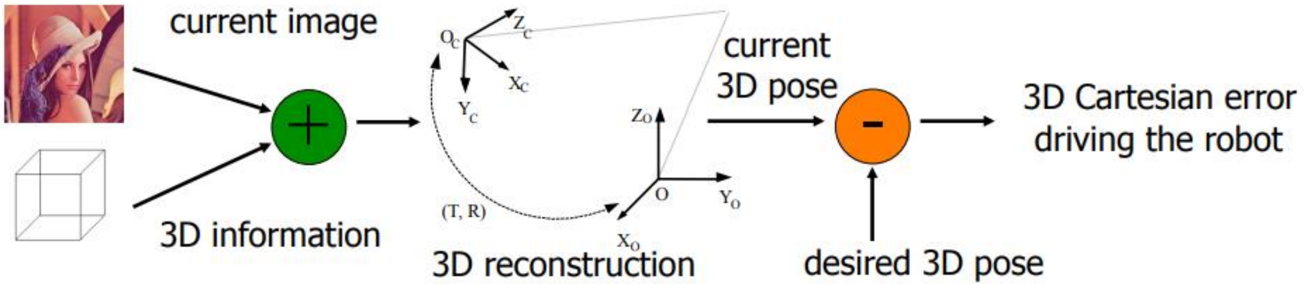


Visual Servoing

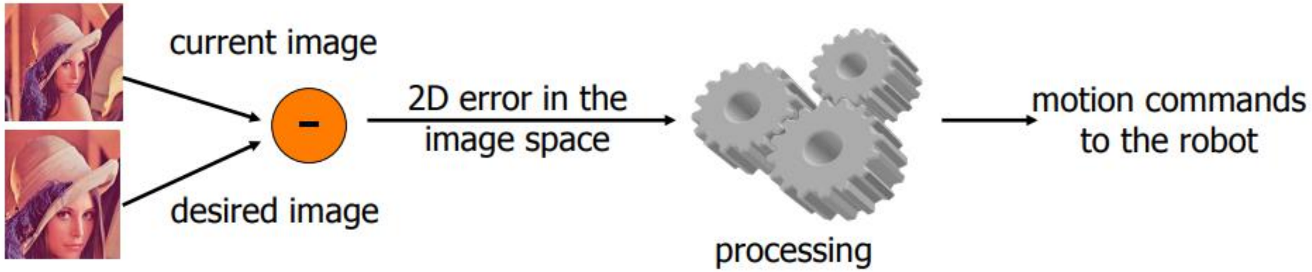
- Idea: Vision can help us “close the loop” and perform feedback control for robot tasks, e.g. moving a manipulator or performing motion control
- *Position-based* visual servoing (**PBVS**): First perform pose estimation, then generate an error based on distance away from objective
- *Image-based* visual servoing (**IBVS**): No 3D estimation; compute error directly from features on 2D image and generate appropriate control for robot

Visual Servoing

- position-based visual servoing (**PBVS**)



- image-based visual servoing (**IBVS**)



PBVS vs IBVS

- For PBVS, coming up with the controller is usually easy, e.g. gradient descent
- 3D pose estimation is often the difficult part

- For IBVS, coming up with a controller in image space will require some care
- On the other hand, no need to worry about camera/calibration parameters

- Let’s again suppose that we already have known features/correspondences

- How to relate feature motion to camera motion?

Feature Jacobian

- Suppose we have a feature vector: $\boldsymbol{f} = (\boldsymbol{f}_1, \boldsymbol{f}_2, \dots, \boldsymbol{f}_k)^T$

- Then we can find a **feature Jacobian** (aka **interaction matrix**) between the camera's 3D velocities and the features' velocities

$$\dot{\boldsymbol{f}} = \boldsymbol{J}_P \begin{pmatrix} \boldsymbol{v}_C \\ \boldsymbol{\omega}_C \end{pmatrix}$$

- Camera velocities are expressed relative to camera frame

- $\boldsymbol{J}_P$ is a $k|\boldsymbol{f}_i| \times 6$ matrix

Point Features

- From pinhole camera model:

$$x = \frac{fX}{Z} \qquad y = \frac{fY}{Z} \qquad \Longrightarrow \qquad \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} f/Z & 0 & -x/Z \\ 0 & f/Z & -y/Z \end{pmatrix} \begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix} = \boldsymbol{J_1} \begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix}$$

- If camera is moving, then point velocity is related by

$$\begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix} = -\left(\boldsymbol{v} + \boldsymbol{\omega} \times \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \right) = \begin{pmatrix} -1 & 0 & 0 & 0 & -Z & Y \\ 0 & -1 & 0 & Z & 0 & -X \\ 0 & 0 & -1 & -Y & X & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{pmatrix} = \boldsymbol{J_2} \begin{pmatrix} v \\ \omega \end{pmatrix}$$

- The full feature Jacobian equation is thus

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \boldsymbol{J_1 J_2} \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{pmatrix} = \begin{pmatrix} -f/Z & 0 & x/Z & xy/f & -f - x^2/f & y \\ 0 & -f/Z & y/Z & f + y^2/f & -xy/f & -x \end{pmatrix} \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{pmatrix} = \boldsymbol{J_P} \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{pmatrix}$$

Point Features

$$J_P = \begin{pmatrix} -f/Z & 0 & x/Z & xy/f & -f - x^2/f & y \\ 0 & -f/Z & y/Z & f + y^2/f & -xy/f & -x \end{pmatrix}$$

- Note that J_P has dependencies on (x, y), location of feature on image plane, as well as Z, or depth of the corresponding feature in 3D

- J_P is 2 by 6, which means it has a null space of dimension 4
- Correspond to camera motions that do not affect feature (e.g. translation along projection ray)

- Additional features are stacked to create a giant $2k$ by 6 matrix

Image Jacobian

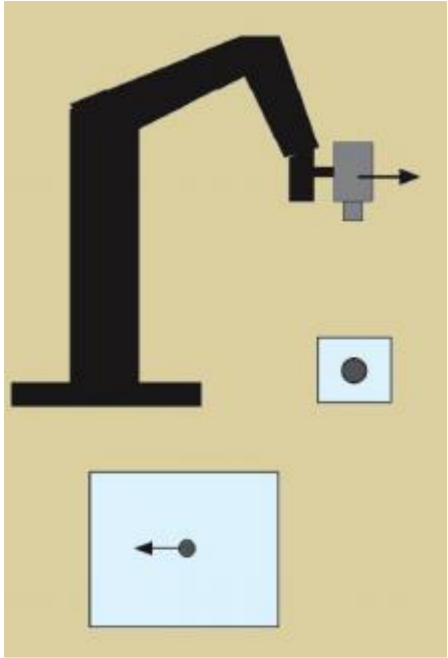
- Suppose our camera is directly mounted on the robot
- Moving the camera is the same as moving the robot!

- Recall Jacobian relationship for robot kinematics:

$$\begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{pmatrix} = \boldsymbol{J}_m(\boldsymbol{q})\boldsymbol{u}$$

- Concatenate with feature Jacobian to obtain **image Jacobian**: $\dot{\boldsymbol{f}} = \boldsymbol{J}_P(\boldsymbol{f}, Z)\boldsymbol{J}_m(\boldsymbol{q})\boldsymbol{u} = \boldsymbol{J}(\boldsymbol{f}, Z, \boldsymbol{q})\boldsymbol{u}$

- We can now do inverse kinematics!



IBVS Inverse Kinematics

$$\dot{\boldsymbol{f}} = \boldsymbol{J}(\boldsymbol{f}, Z, \boldsymbol{q})\boldsymbol{u} \quad \Rightarrow \quad \boldsymbol{u} = \boldsymbol{J}^{+}\left(\dot{\boldsymbol{f}}_{\boldsymbol{d}} + \boldsymbol{K}\boldsymbol{e}\right) + (\boldsymbol{I} - \boldsymbol{J}^{+}\boldsymbol{J})\boldsymbol{u}_0$$

Jacobian pseudoinverse

Desired feature pose

Control gain and error

Null space term

- Similar form to regular manipulator (and mobile) inverse kinematics
- $\boldsymbol{u}_0$ can be used for optimizing controls if there are redundancies
- Error vector $\boldsymbol{e} = \boldsymbol{f}_d - \boldsymbol{f}$ only requires that we know feature values

- What about dependence of $\boldsymbol{J}$ on the feature depth Z?
- One approach: Just use the desired pose depth Z^*
- Otherwise, we can try to estimate it à la state estimation

Practical Considerations

- We never really know the full image Jacobian
- Uncertainties in robot configurations, camera calibration, feature depth…
- Instead we're just using an approximation, e.g. constant feature depth Z^*

- Usually works well enough with longer than optimal convergence time

- Other difficulties: Singularities (e.g. robot motions that do not move features), limited field of view, features leaving image plane…

Hybrid Visual Servoing

- Idea: Control some components in operational space, others in image space
- Specifying operational trajectories can help predict camera trajectories
- Control in image space helps keep features inside field of view

- Common ways of combining IBVS and PBVS:
- IBVS: Translation components, maintaining features in field of view
- PBVS: Rotation components, better guarantees on stability and convergence

Summary

- Camera matrix tells us how 3D objects are transformed into 2D images

- Pose estimation: Figuring out 3D pose given 2D image features
- Analytical solutions in P-n-P problems, also numerical solutions in least squares and RANSAC-like approaches

- Visual servoing: Incorporating image features into the control loop
- PBVS uses pose estimation to find pose first and then control in op space
- IBVS applies IK on image Jacobian to find controls in image space