Robot Autonomy

Lecture 12:
Robot Vision

Oliver Kroemer

Based on Slides of Roberto Cipolla, Wolfram Burgard

Motivation

- Robots need to perceive their surroundings

- Cameras provide large amounts of information quickly
- Detect, recognize, and localise objects in environment

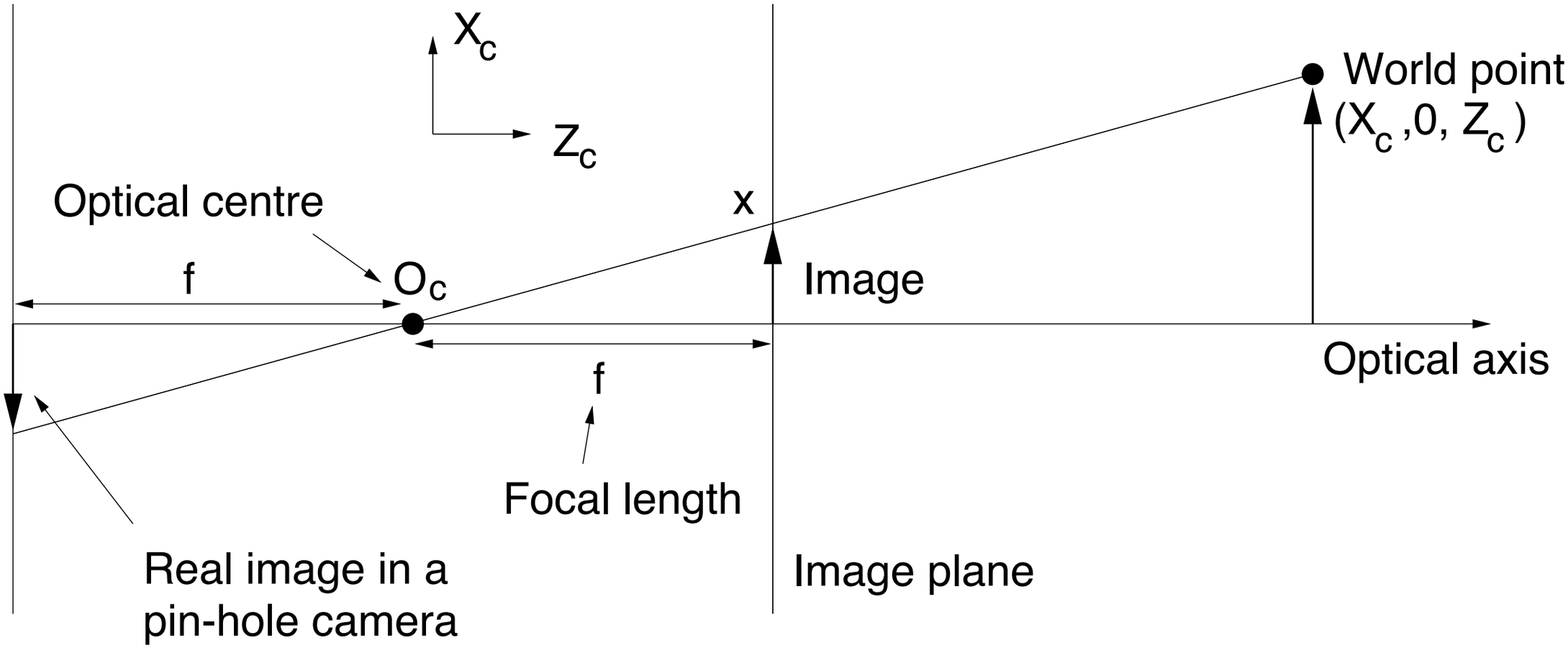
Lecture Overview

- Computer vision is a massive field… won't cover all today
- Camera models and calibration
- Random Sample Consensus (RANSAC)
- Interative Closest Point (ICP)

Camera Model

Perspective Projection

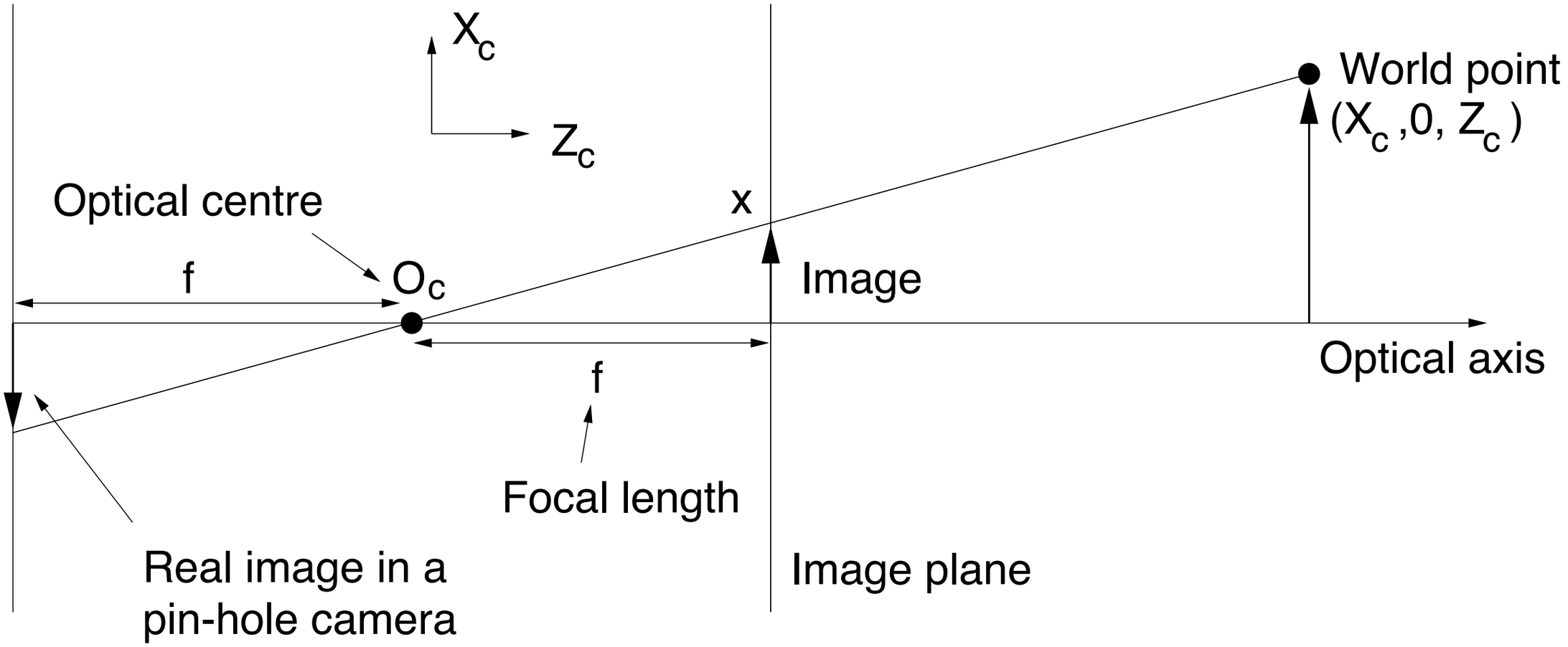
- Camera projects world points onto image plane



- Lens represented as a pinhole at Oc
- Same for the Y direction

Perspective Projection

- Camera projects world points onto image plane

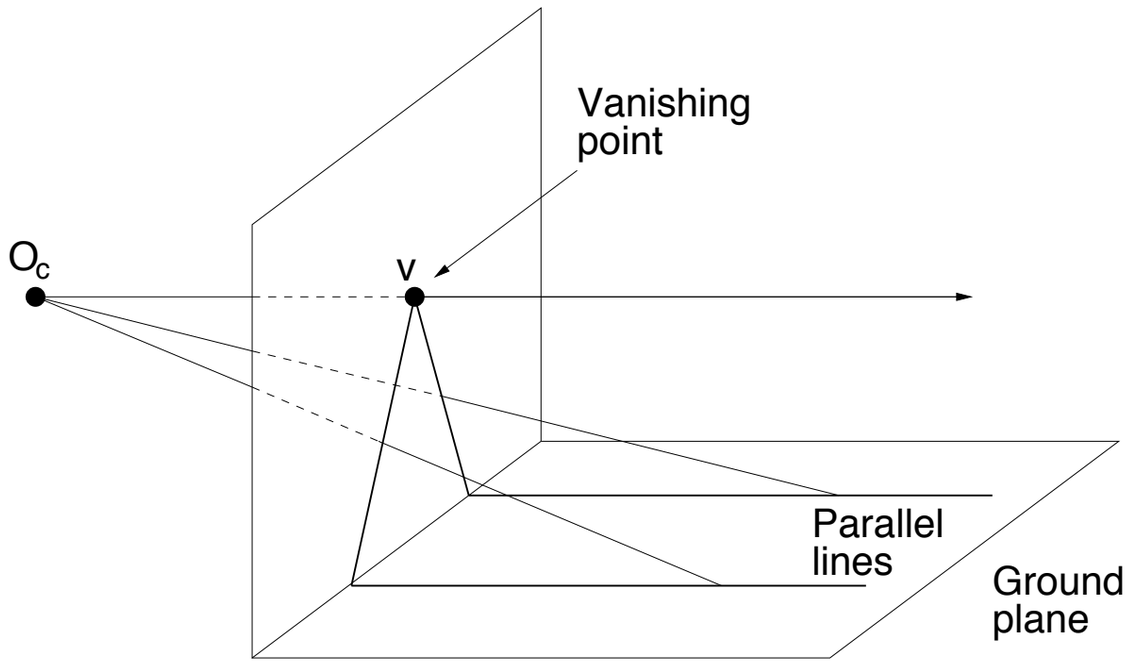


- By considering the similar triangles:

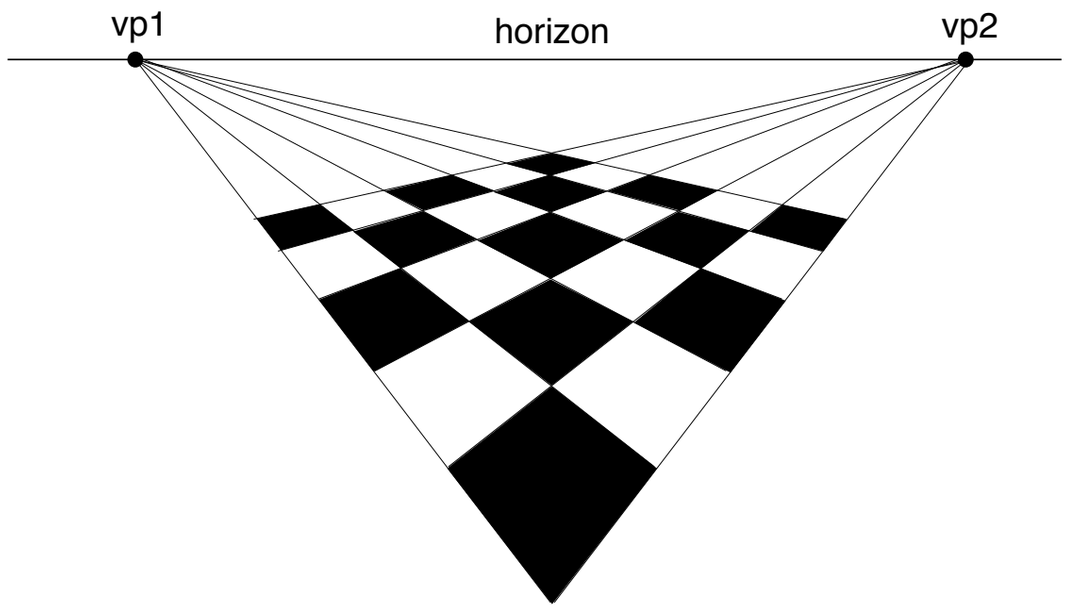
$$\frac{x}{f} = \frac{X_c}{Z_c} \qquad x = \frac{fX_c}{Z_c} \qquad \left(y = \frac{fY_c}{Z_c} \right)$$

Vanishing Points

- Perspective projection results in vanishing points



- Each set of parallel line corresponds to a vanishing point



Vanishing Points

- Consider a 3D line in space

$$\boldsymbol{X}_c = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + \lambda \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

- A point on the line is projected to the image point

$$\boldsymbol{x} = \left(f \frac{a_x + \lambda b_x}{a_z + \lambda b_z}, f \frac{a_y + \lambda b_y}{a_z + \lambda b_z} \right)$$

- As lambda goes to infinity we get the vanishing point

$$\boldsymbol{x}_v = \left(f \frac{b_x}{b_z}, f \frac{b_y}{b_z} \right)$$

- The vanishing point is independent of the line's position

Homogeneous Coordinates

- Projective geometry easier in homogeneous coordinates (imaging process becomes a linear matrix operation)
- Add an additional dimension and arbitrary scaling

$$\boldsymbol{X}_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \longrightarrow \tilde{\boldsymbol{X}}_c = \begin{bmatrix} \lambda X_c \\ \lambda Y_c \\ \lambda Z_c \\ \lambda \end{bmatrix}$$

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow \tilde{\boldsymbol{x}} = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix}$$

- As a convention, set scaling to one when possible

Perspective Projection

- Returning to our perspective projection we have

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda X_c \\ \lambda Y_c \\ \lambda Z_c \\ \lambda \end{bmatrix}$$

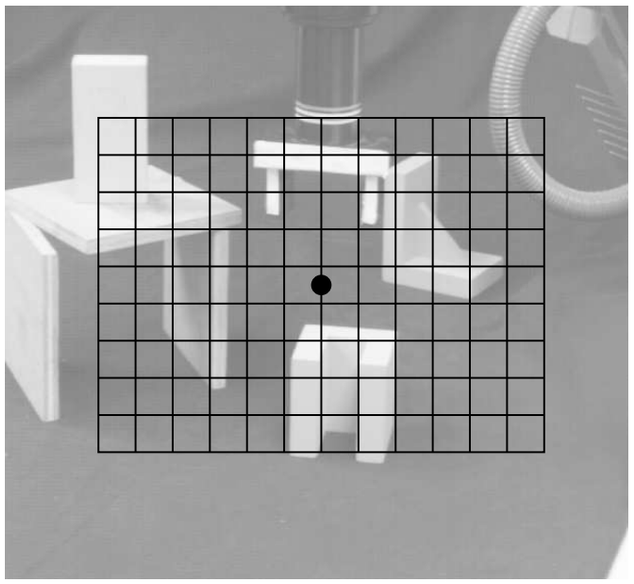
- Which corresponds to (as before)

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} f\lambda X_c \\ f\lambda Y_c \\ \lambda Z_c \end{bmatrix} \Longrightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} fX_c/Z_c \\ fY_c/Z_c \end{bmatrix}$$

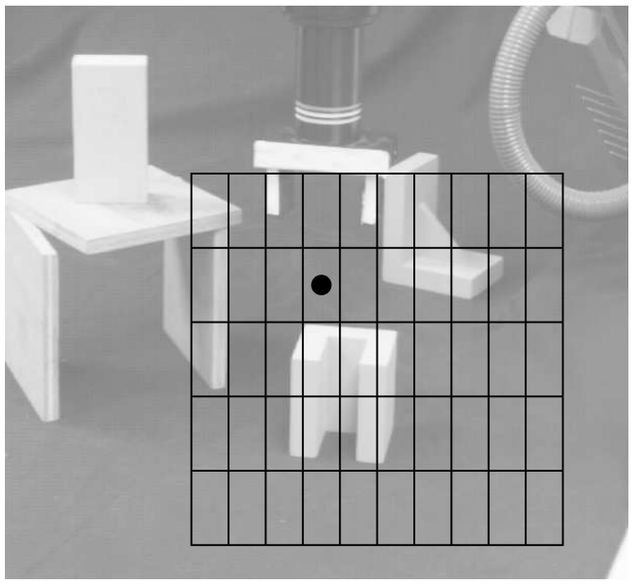
- Projection becomes a linear matrix operation $\tilde{\boldsymbol{x}} = P_p \tilde{\boldsymbol{X}}_{\boldsymbol{c}}$

Full Camera Model

- Full camera model maps world points to pixels
- Rigid body transform from world to camera frame
- Perspective projection onto image plane ✓
- Charge-coupled device (CCD) Imaging into pixel array



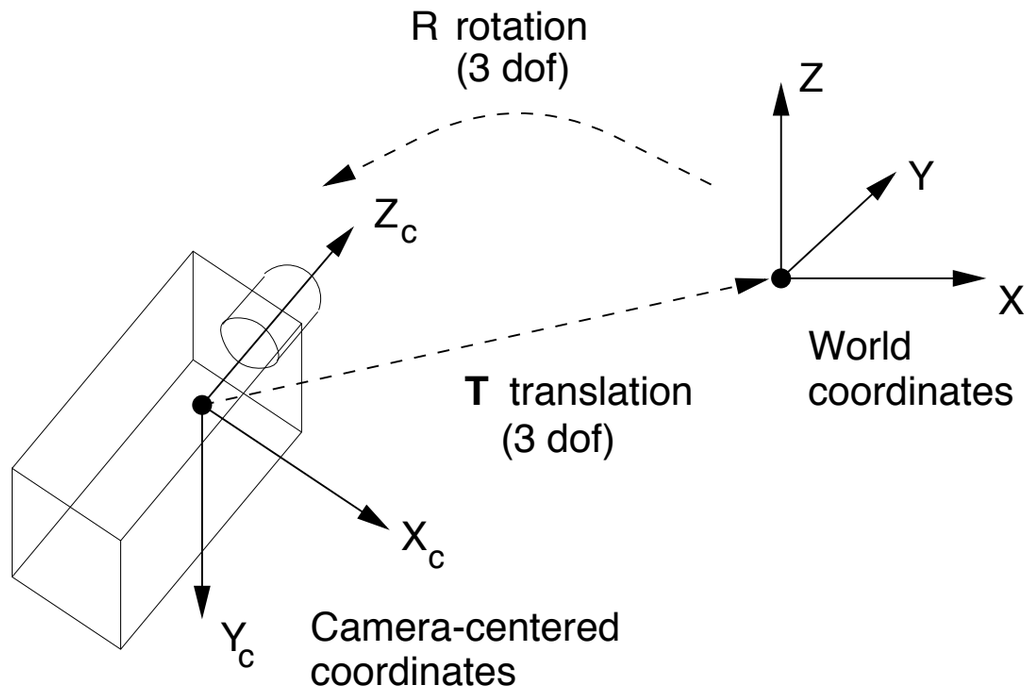
Array centered on optical axis
Square pixels



Array not centered on optical axis
Rectangular pixels

Rigid Body Transform

- Map world point into camera coordinate frame

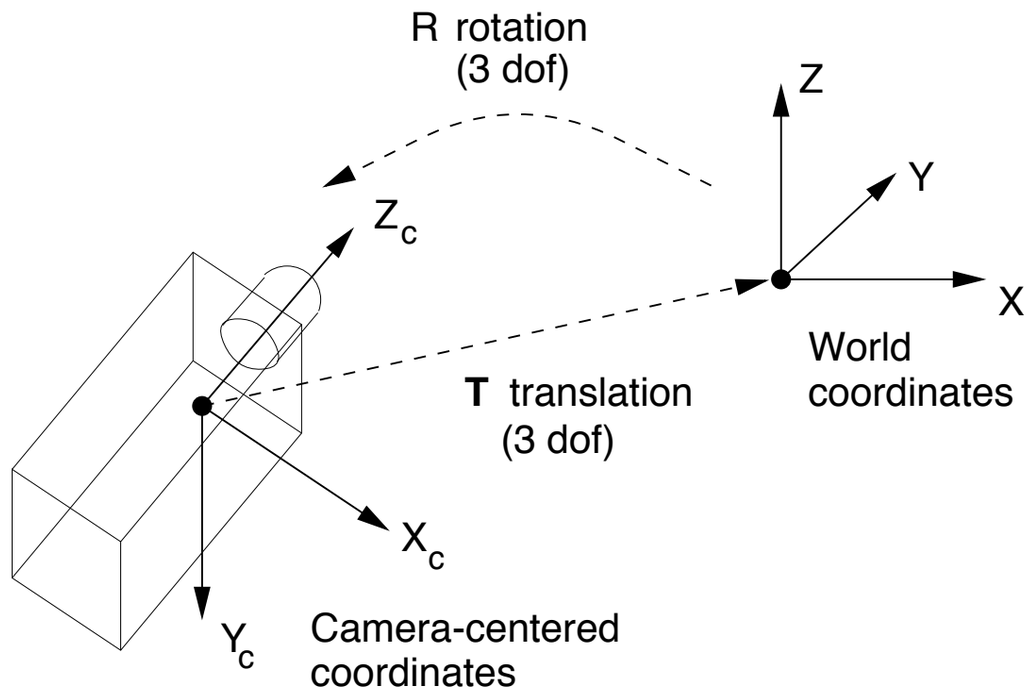


$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

$$\boldsymbol{X}_c = R\boldsymbol{X} + \boldsymbol{T}$$

Rigid Body Transform

- Map world point into camera coordinate frame

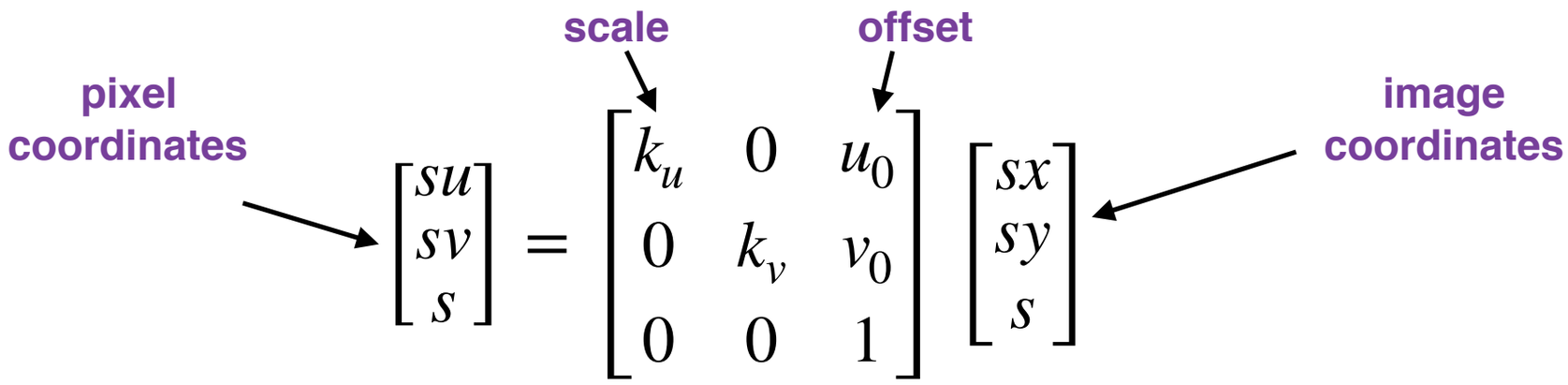


$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\tilde{\mathbf{X}}_c = P_r \tilde{\mathbf{X}}$$

Full Perspective Camera Model

- Imaging needs to scale and offset image for pixels



- Often combined with perspective projection

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} \qquad \alpha_u = k_u f$$

- Full transformation is thus given by

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Full Perspective Camera Model

- The perspective camera model

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

upper triangular

orthonormal

$$\tilde{\boldsymbol{w}} = P_{ps}\tilde{\boldsymbol{X}}$$

$$P_{ps} = \boldsymbol{K}[R \mid T]$$

camera calibration matrix

extrinsic calibration

- Considerable amount of structure in the matrix

Projective Camera Model

- Often more convenient to use projective camera model

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_x \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

note:scaling P does not matter

- Perspective camera model is a projective camera model

- Projective camera model is less constrained
  - Use projective model to perform initial calibration
  - Extract perspective camera parameter estimates afterwards

Calibration

- Use gradient descent to minimise projection error

$$\min_P \sum_i ((u_i - \hat{u}_i)^2 + (v_i - \hat{v}_i)^2)$$

$$\hat{u}_i = \frac{p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}}$$

$$\hat{v}_i = \frac{p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}}$$

- Remember that you can scale matrix arbitrarily

Matrix Decomposition

- Often want to determine extrinsic calibration parameters
  - Not technically perspective, but can extract estimate

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_x \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Matrix Decomposition

- Often want to determine extrinsic calibration parameters
  - Not technically perspective, but can extract estimate

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_x \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

3x3 RQ matrix decomposition $A = \boldsymbol{K}\boldsymbol{R}$

R: right triangular matrix $\boldsymbol{K}$

Q:orthonormal matrix $\boldsymbol{R}$

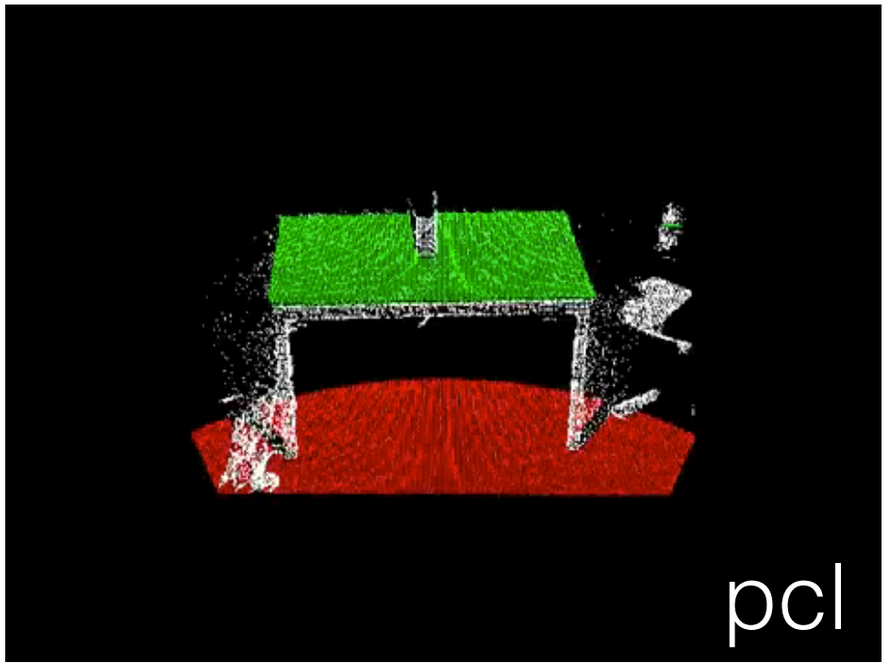
$$T = K^{-1} \begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix}$$

note:QR decomposition not same as RQ, order matters

Plane Estimation and RANSAC

Problem

- Laser scanners and RGB-D(epth) camera give point clouds
  - Each point corresponds to a 3D point on a surface
- Often want to detect certain structures in the scene
  - segment out the table from the tabletop setting



- Need to reliably estimate table plane parameters

Plane Estimation

- Estimate plane for a set of 3D points

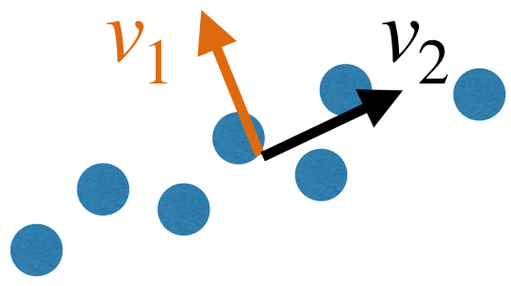
$$p_1, p_2, \ldots, p_n$$

- Need to estimate plane normal and offset
  - Compute covariance matrix

$$C = n^{-1} \sum_i (p_i - \bar{p})(p_i - \bar{p})^T$$

  - Normal given by eigenvector of C with smallest eigenvalue

$$Cv_i = \beta v_i$$

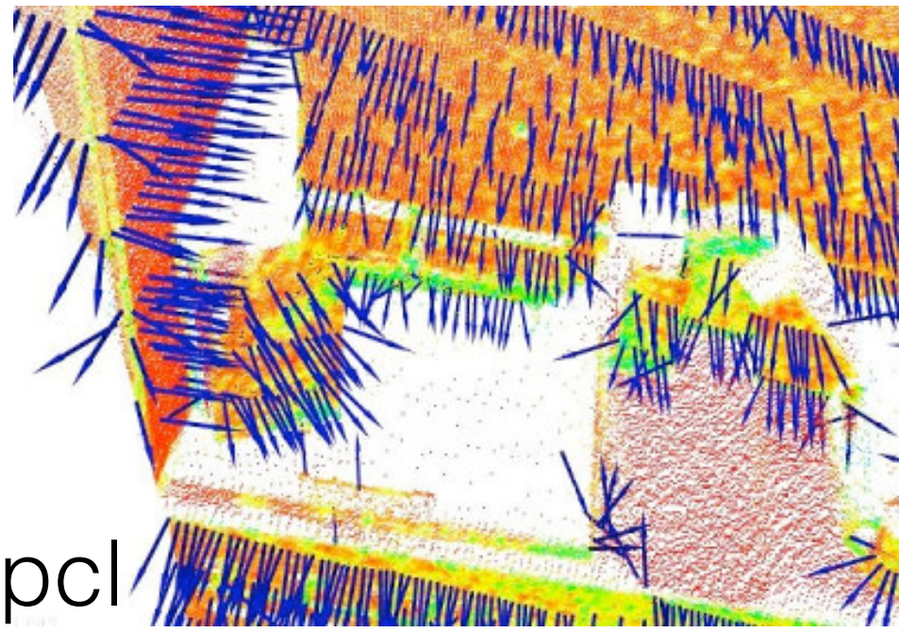


$$\beta_1 < \beta_2$$

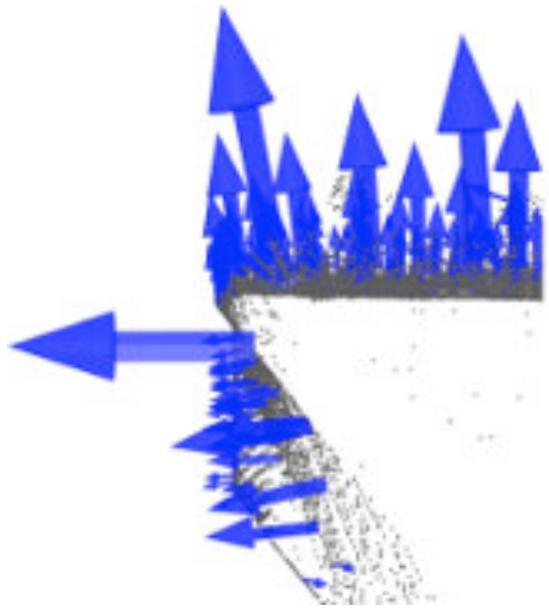
  - Offset is then given by $v_1^T \bar{p}$

Normal Estimation

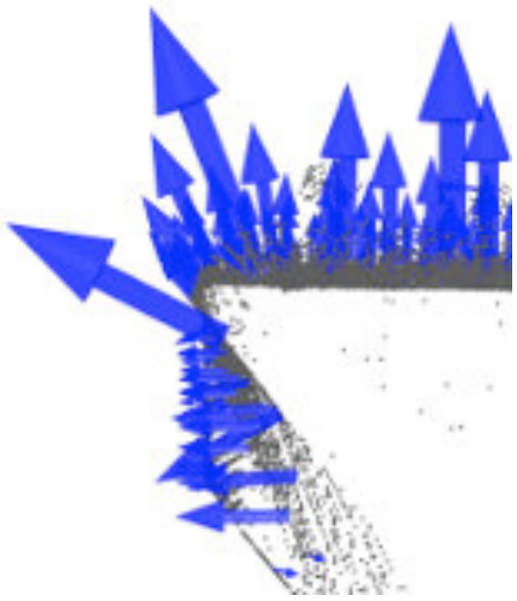
- Can use same approach to compute point normals
- Set of points given by local neighbourhood around point



Normals point towards camera



Small Neighbourhood

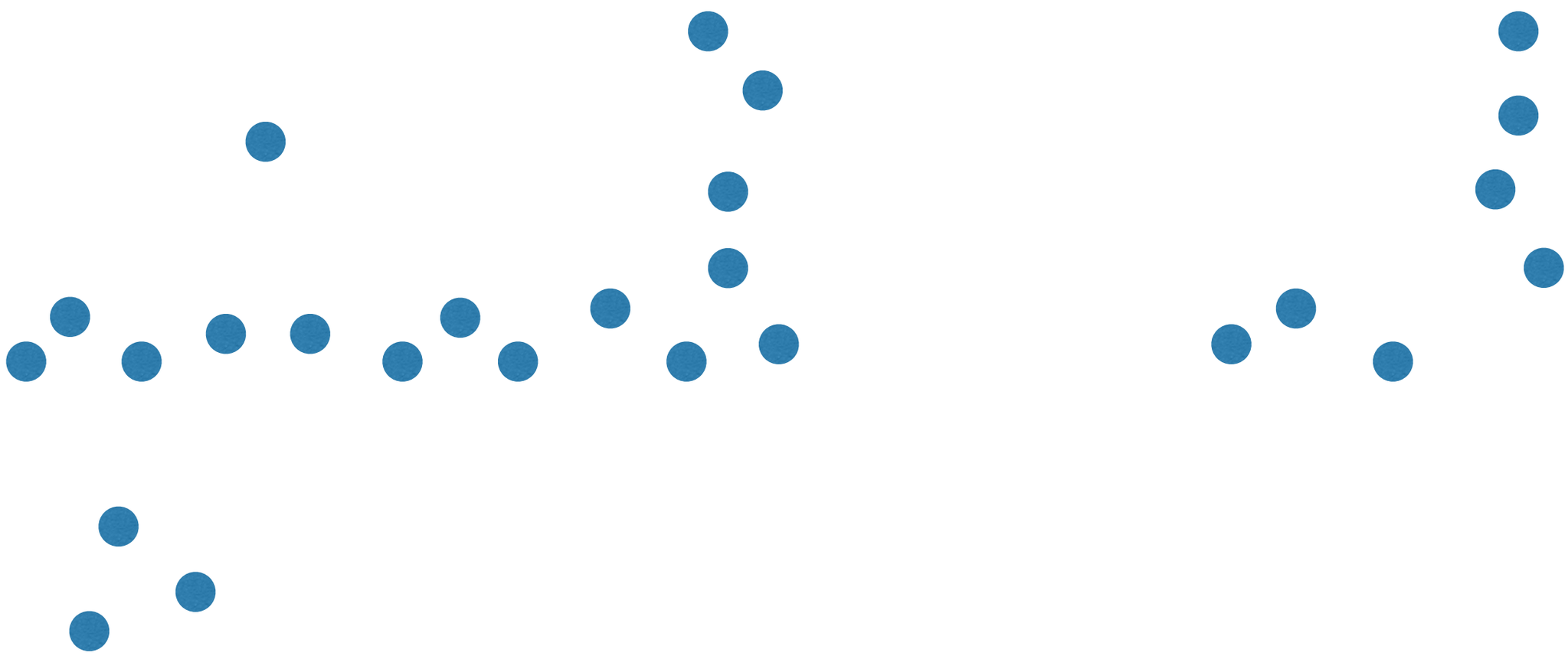


Large Neighbourhood

- Size of neighbourhood effects smoothness of normals
- How to select set of points for plane segmentation?

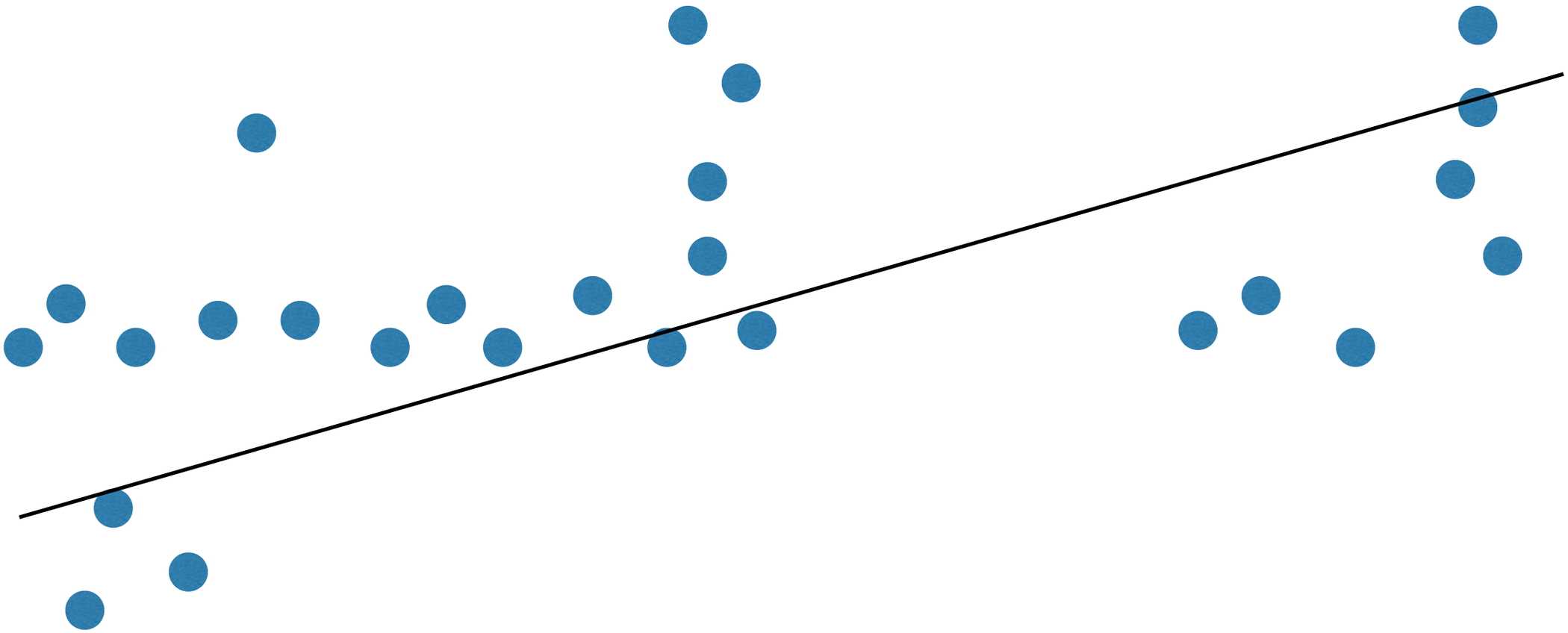
RANSAC

- Scene may have multiple planes and outliers



RANSAC

- Scene may have multiple planes and outliers



- Do not want to fit plane to ALL of the points in scene
- Large errors and overall a poor fit to scene

RANSAC

- Random Sample Consensus (RANSAC)

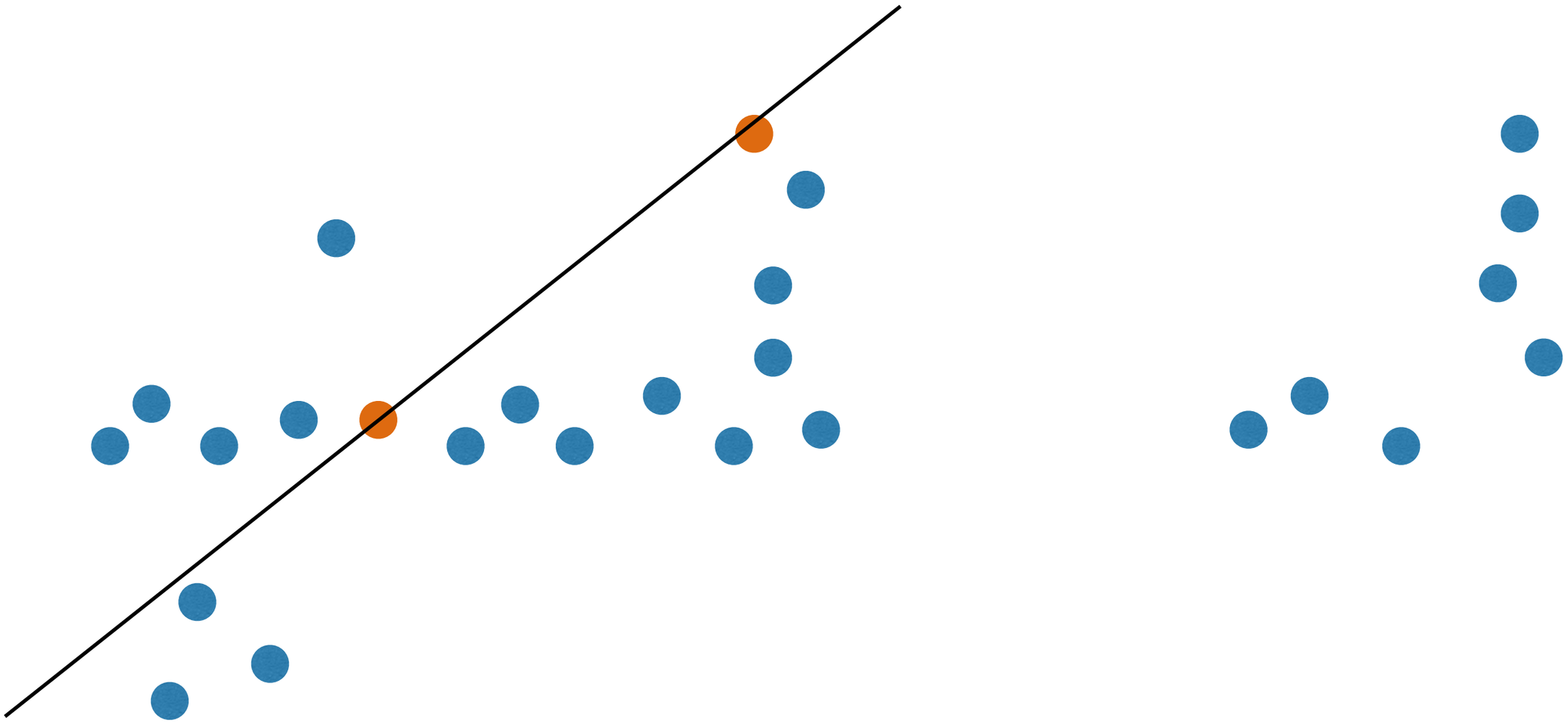
1. Randomly select a minimal set of points to fit model
2. Fit model to selected points
3. Determine set of inliers to trained model
4. If not enough inliers, return to first step
5. May re-estimate model using inliers
6. Compute quality of model (e.g., nr inliers or RMSE)
7. Update BestModel if new model is better
8. Repeat (k times, or anytime)

RANSAC

- Sample points

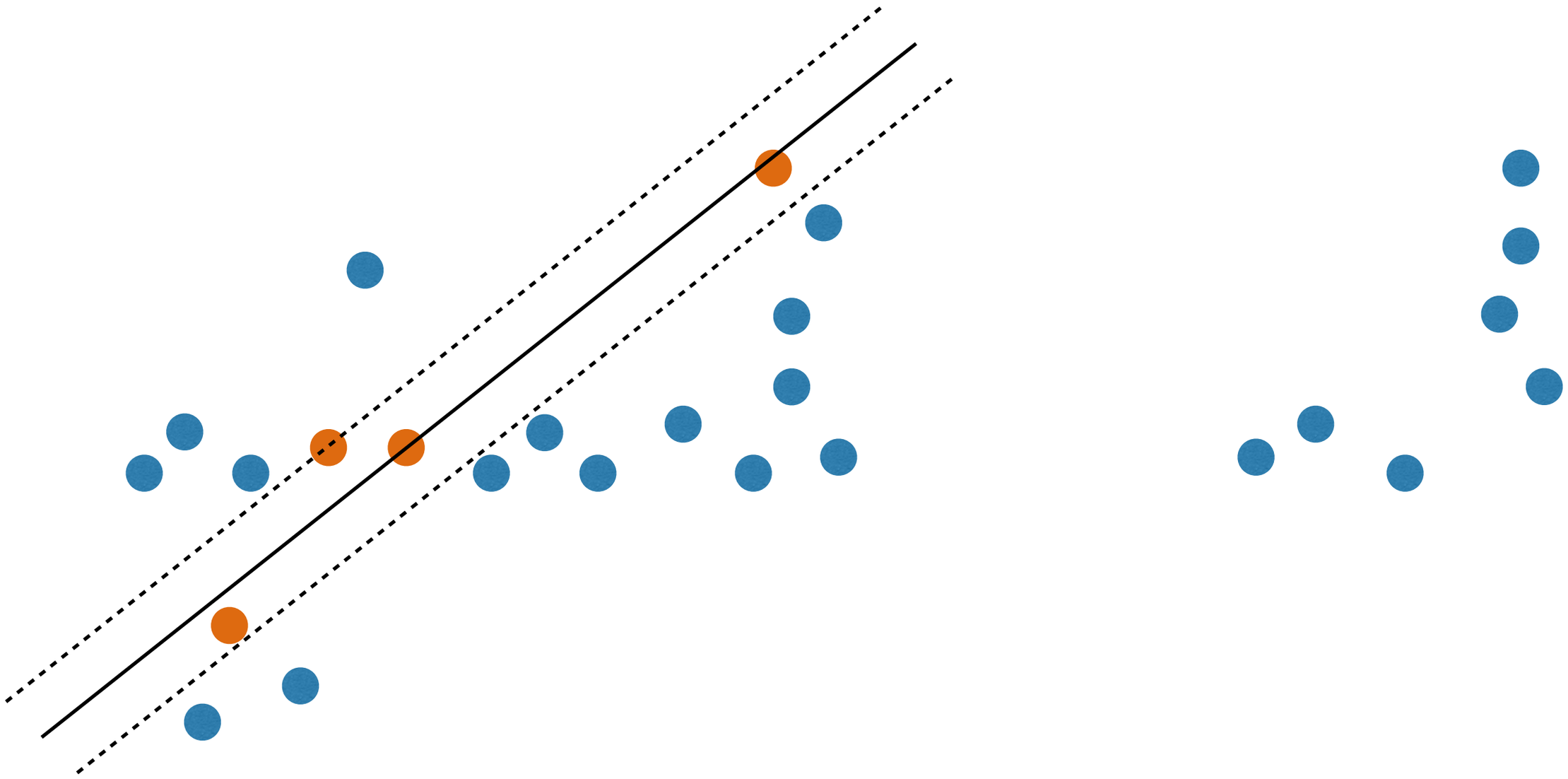
RANSAC

- Fit model



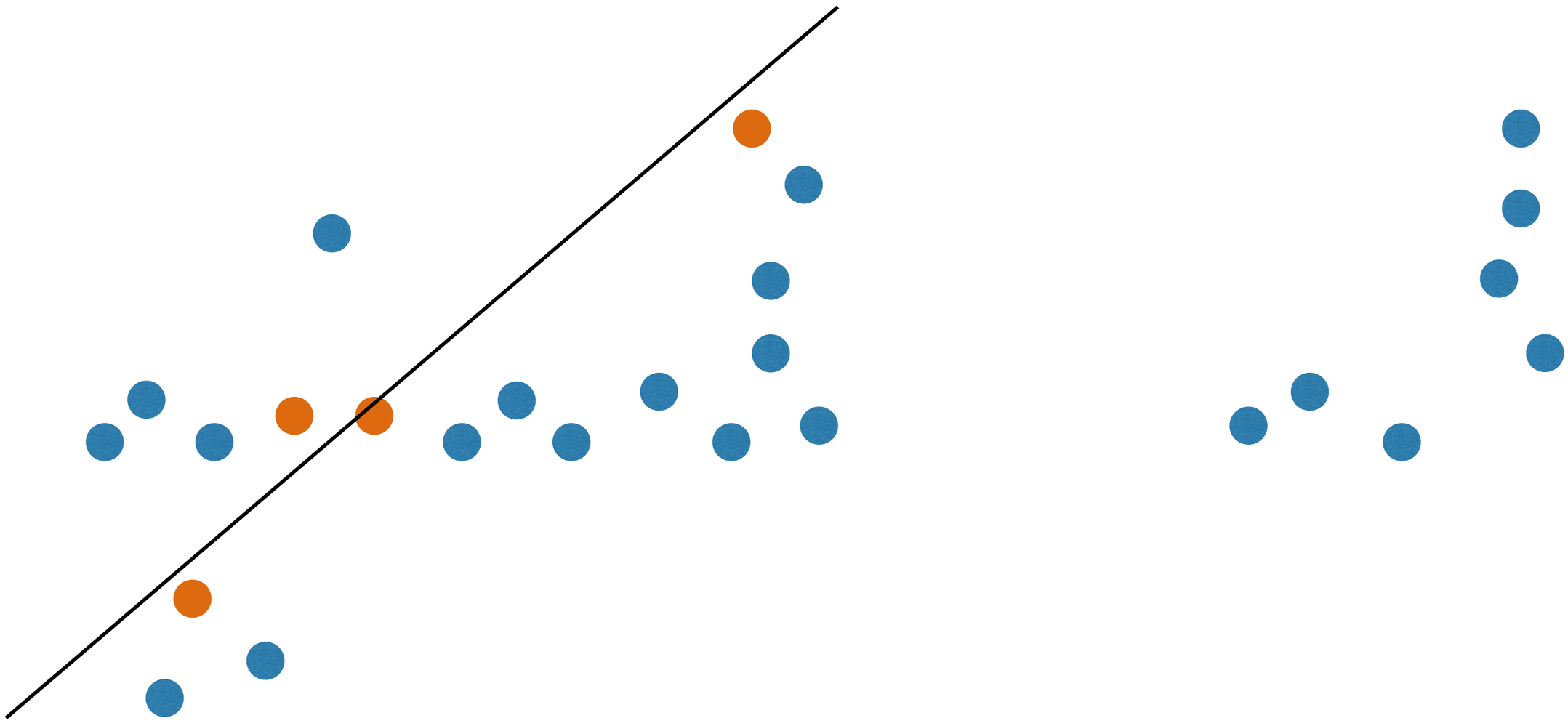
RANSAC

- Compute inliers



RANSAC

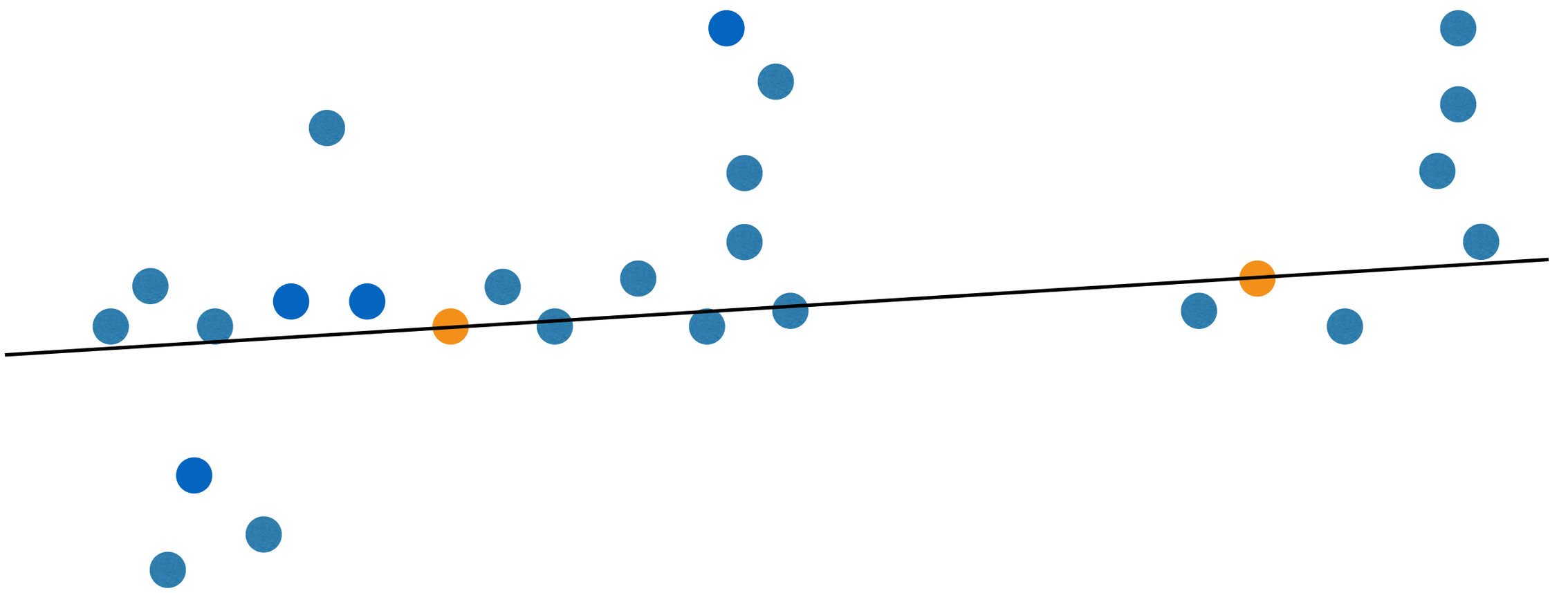
- Adjust model and compute quality



Inliers = 4
(new best)

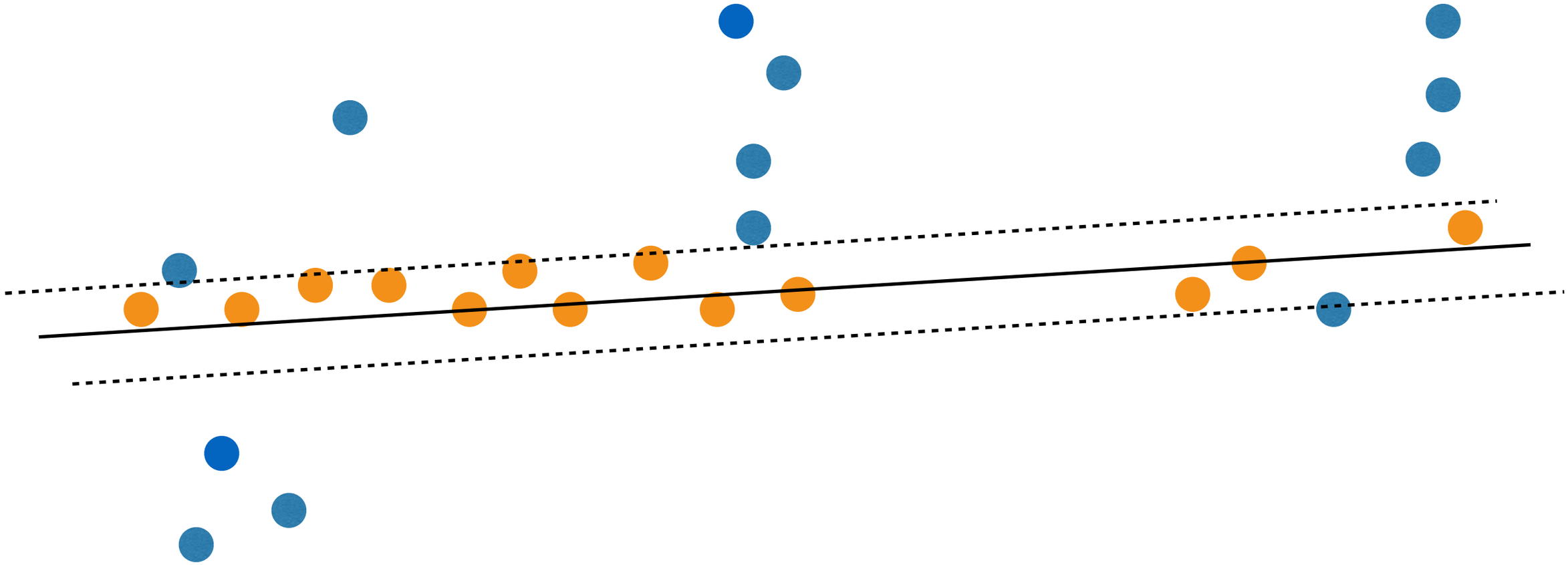
RANSAC

- Repeat



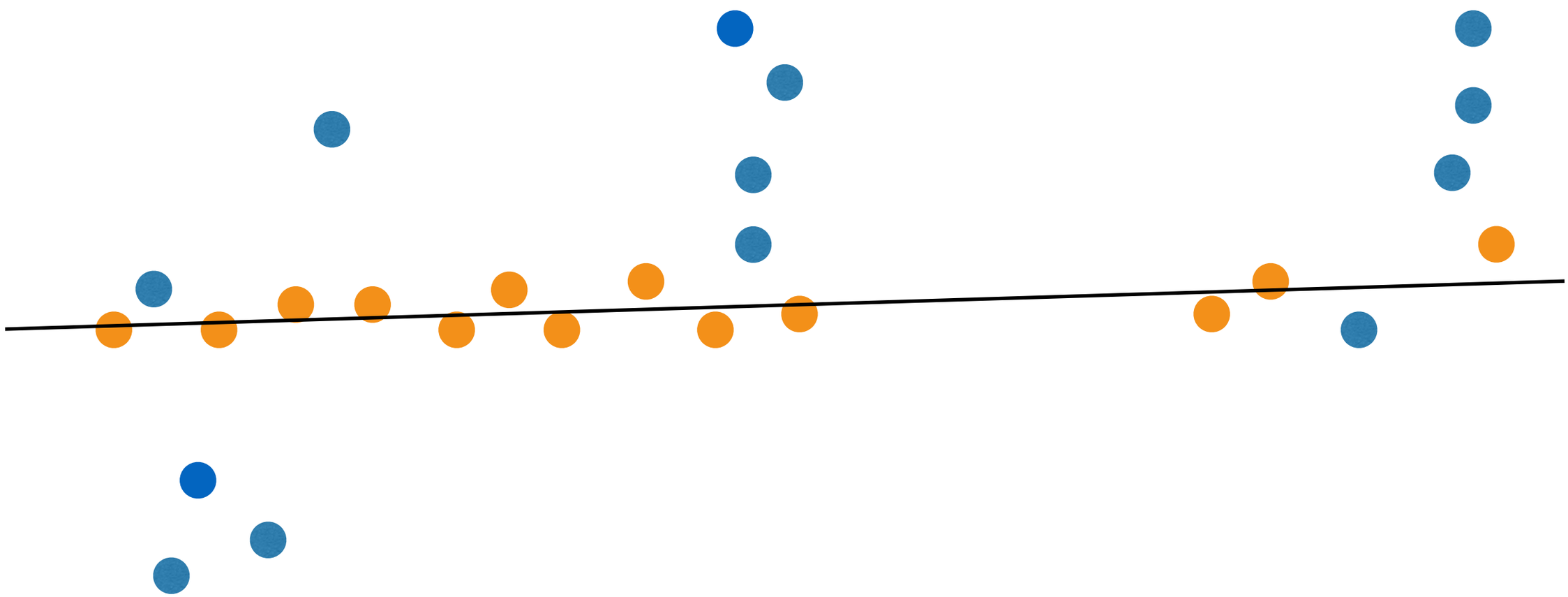
RANSAC

- Repeat



RANSAC

- Repeat



Inliers = 13
(new best)

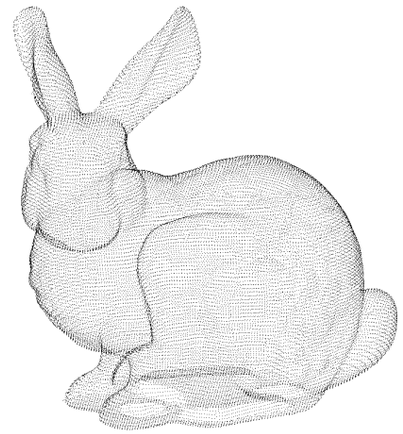
RANSAC

- For multiple planes, remove first one and repeat process

Iterative Closest Point

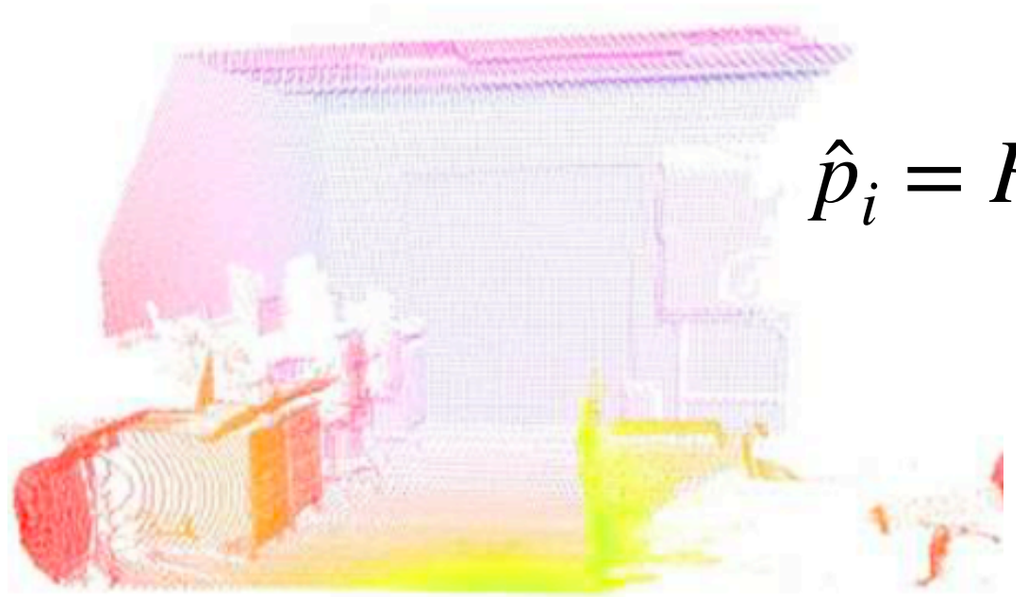
Registering Point Clouds

- Given a point cloud model and scene



p

R, T



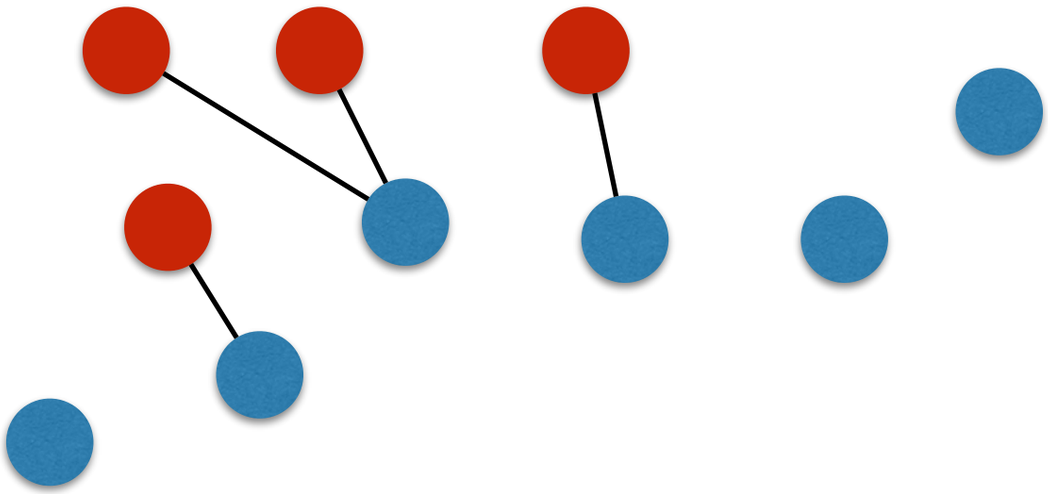
x

$$\hat{p}_i = Rp_i + T$$

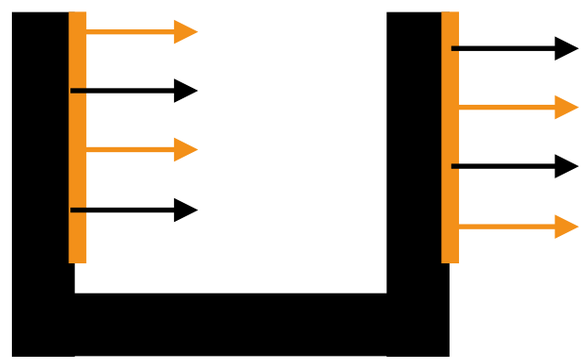
- Determine transform of model to align with the scene
- Assume a coarse initial alignment of model in scene
- Scene has many more points… find correspondences

Point Correspondances

- Given two sets of points, determine correspondences
- Use nearest points (within threshold)



  - Correspondences may not adhere to rigid transformation
- Consider compatible points
  - Color/hue within a certain threshold
  - Normals within a certain threshold



Computing the Transform

- Given a set of corresponding 3D points

$$x_1, x_2, \ldots, x_n \qquad p_1, p_2, \ldots, p_n$$

- Computed centered versions of the points

$$\tilde{x}_i = x_i - \mu_x \qquad \tilde{p}_i = p_i - \mu_p$$

$$\mu_x = n^{-1} \sum_i x_i \qquad \mu_p = n^{-1} \sum_i p_i$$

- Compute the cross-covariance matrix

$$W = \sum_i^n \tilde{x}_i \tilde{p}_i^T$$

Computing the Transform

- Apply singular value decomposition (SVD) to W

$$W = USV^T$$

diagonal singular values

- Transform is then given by

set singular values to one

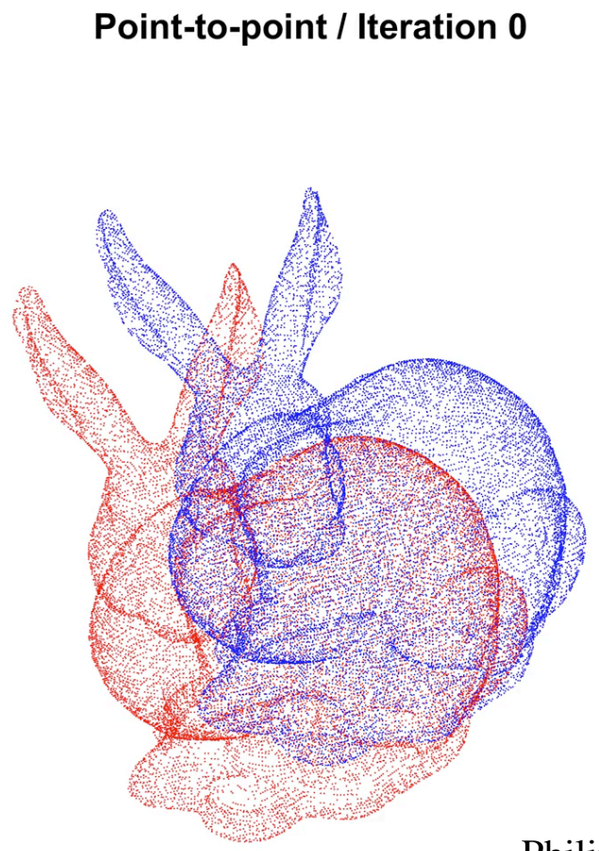
$$R = UV^T$$

$$T = \mu_x - R\mu_p$$

- Update the points given the new transformation

$$\hat{p}_i = Rp_i + T$$

- Iterative Closest Point (ICP)
  - Find correspondences
  - Compute transformation
  - Update the points to scene
  - Repeat until convergence or k steps



Philipp Glira

- Useful for adjusting pose of model to a scene
- Requires suitable initialisation (local solution)
- Beware of extruded shapes (self-similarity)

Questions?