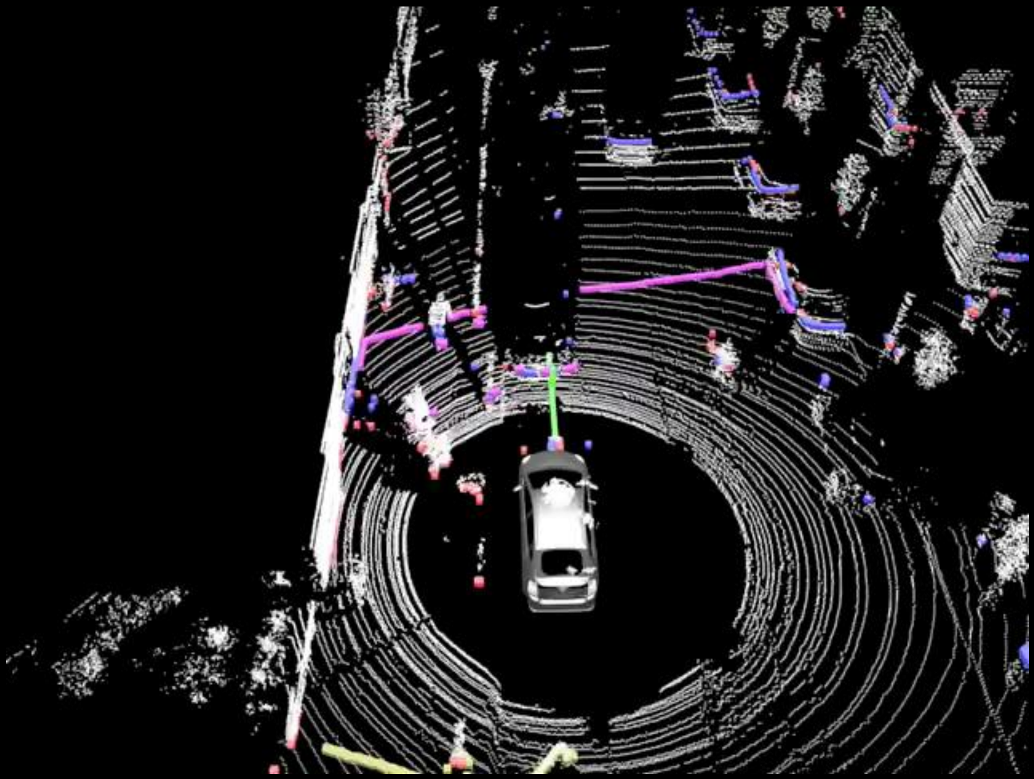
Least Squares and RANSAC

Using examples from pointclouds.org

Motivation

- Robot perception often uses point clouds
- Figuring our where the ground and objects are is essential for navigation
- We'll talk about two tools related to this: Least-Squares and RANSAC



Video from Matthew Johnson-Roberson

Least-Squares

Matrix Inversion

- Matrix inversion is a common way to solve problems in linear algebra
    - Used everywhere in robotics, from vision to kinematics
- $\mathbf{A}^{-1}$ is the inverse of $\mathbf{A}$ if

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

    - $\mathbf{A}$ must be square (n x n)
    - $\mathbf{A}$ must be *invertible…*

Matrix Invertability

- A square matrix is called singular if it is *not* invertible
- An *n* x *n* matrix $\mathbf{A}$ is invertible if (the statements below are equivalent – only need to check one)
  - $\mathbf{A}$ has rank *n*
    - Rank is the number of linearly independent columns
  - The determinant of $\mathbf{A}$ is not 0
    - The determinant can be viewed as how much the transformation described by the matrix scales an input
  - Many many other ways to check invertability….
- Rank, determinant, and inversion implementations are easy to find in Eigen

Using Matrix Inversion

- Probably the most common problem in linear algebra: Given a matrix $\mathbf{A}$ and vector b, and the following equation

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

solve for the vector x

- Use this to solve a system of linear equations. For example:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \end{aligned} \quad \Rightarrow \quad \begin{array}{cccc} \mathbf{A} & x & & b \\ \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} & \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} & = & \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \end{array}$$

Solving $\mathbf{A}x = b$

- If $\mathbf{A}$ is <u>*n* x *n*</u> and b is *n* x 1
  - Check rank or determinant of A to see if it is invertible.
  - If so, use the matrix inverse:

$$\mathbf{A}^{-1}\mathbf{A}x = \mathbf{A}^{-1}b$$
$$\mathbf{I}x = \mathbf{A}^{-1}b$$
$$x = \mathbf{A}^{-1}b$$

  - If not, no solution

- What if $\mathbf{A}$ is not square?

The Pseudo-inverse

- The Moore-Penrose Pseudo-inverse is defined as

$$\mathbf{A}^{+} = (\mathbf{A}^{T}\mathbf{A})^{-\mathbf{1}}\mathbf{A}^{T} \qquad (\textit{left}\text{ pseudo-inverse})$$

  - Has some of the properties of the inverse, most importantly:

$$\mathbf{A}^{+}\mathbf{A} = \mathbf{I}$$

  - Derivation:

$$\mathbf{I} = (\mathbf{A}^{T}\mathbf{A})^{-\mathbf{1}}(\mathbf{A}^{T}\mathbf{A})$$

$$\mathbf{I} = [(\mathbf{A}^{T}\mathbf{A})^{-\mathbf{1}}\mathbf{A}^{T}]\mathbf{A}$$

$$\mathbf{I} = \mathbf{A}^{+}\mathbf{A}$$

- The *right* pseudo-inverse is derived similarly to get $\mathbf{A}\mathbf{A}^{+} = \mathbf{I}$

$$\mathbf{A}^{+} = \mathbf{A}^{T}(\mathbf{A}\mathbf{A}^{T})^{-\mathbf{1}} \qquad (\textit{right}\text{ pseudo-inverse})$$

The Pseudo-inverse

$$\mathbf{A}^{+} = (\mathbf{A}^{T}\mathbf{A})^{-\mathbf{1}}\mathbf{A}^{T} \quad \text{(\textit{left} pseudo-inverse)}$$
$$\mathbf{A}^{+} = \mathbf{A}^{T}(\mathbf{A}\mathbf{A}^{T})^{-\mathbf{1}} \quad \text{(\textit{right} pseudo-inverse)}$$

- Works even when $\mathbf{A}$ is not square
- What about $(\mathbf{A}^{T}\mathbf{A})^{-\mathbf{1}}$ (left) or $(\mathbf{A}\mathbf{A}^{T})^{-\mathbf{1}}$ (right)?
  - $(\mathbf{A}^{T}\mathbf{A})$ or $(\mathbf{A}\mathbf{A}^{T})$ is automatically square
  - But we need to check if $(\mathbf{A}^{T}\mathbf{A})$ or $(\mathbf{A}\mathbf{A}^{T})$ is invertable

- If $\mathbf{A}$ is square and invertable, then $\mathbf{A}^{+} = \mathbf{A}^{-\mathbf{1}}$
  - We don’t loose any generality by always using the pseudo-inverse

The Pseudo-inverse

- Can use the pseudo-inverse like an inverse to solve $\mathbf{A}x = b$ when $\mathbf{A}$ is m x n and b is m x 1 :

$$\mathbf{A}^{+}\mathbf{A}x = \mathbf{A}^{+}b$$
$$\mathbf{I}x = \mathbf{A}^{+}b$$
$$x = \mathbf{A}^{+}b$$

- This is known as the least-squares solution
- Remember that $(\mathbf{A}^{T}\mathbf{A})^{-\mathbf{1}}$ (left) or $(\mathbf{A}\mathbf{A}^{T})^{-\mathbf{1}}$ (right) must be invertable

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

- What does this mean for solving linear systems of equations represented by $\mathbf{A}$ (m x n) and b (m x 1)?
  - m is the number of equations
  - n is the number of unknowns (x is n x 1)

- If m = n
  - $x = \mathbf{A}^{+} b$ is the *exact* solution to the system of equations
- If m < n (*underdetermined*; many solutions are possible)
  - $x = \mathbf{A}^{+} b$ outputs an $\boldsymbol{x}$ that minimizes $||\boldsymbol{x}||_2$
- If m > n (*overdetermined*; no exact solution in general)
  - $x = \mathbf{A}^{+} b$ outputs an $\boldsymbol{x}$ that minimizes the sum of squared errors

Eigen example

- In math, the method is clear, but how to do it in code?
- Eigen has many ways to compute a least-squares solution, let's compare a few

```cpp
void printSolutions(const Eigen::MatrixXd& A, const Eigen::VectorXd& b)
{
    //solve Ax=b using left-pseudoinverse with .inverse() (problem if A.transpose()*A is singular)
    Eigen::VectorXd x1a = (A.transpose()*A).inverse()*A.transpose()*b;
    std::cout << "Using left pseduoinverse with .inverse() x =" << std::endl << x1a
                                                                 << std::endl << std::endl;
    std::cout << "Error magnitude: " << (A*x1a-b).norm() << std::endl;
    std::cout << std::endl;

    //solve Ax=b using right-pseudoinverse with .inverse() (problem if A*A.transpose() is singular)
    Eigen::VectorXd x1b = A.transpose()*(A*A.transpose()).inverse()*b;
    std::cout << "Using right pseduoinverse with .inverse() x =" << std::endl << x1b
                                                                 << std::endl << std::endl;
    std::cout << "Error magnitude: " << (A*x1b-b).norm() << std::endl;
    std::cout << std::endl;

    //alternatively, you could use QR decomposition (more numerically stable)
    Eigen::VectorXd x2 = A.colPivHouseholderQr().solve(b);
    std::cout << "Using QR decomposition x =" << std::endl << x2
                                                           << std::endl << std::endl;
    std::cout << "Error magnitude: " << (A*x2-b).norm()<< std::endl;
}
```

Test Exactly Constrained Ax=b

```
Eigen::VectorXd b(4);
b << 1,1,1,1;

//exactly constrained
Eigen::MatrixXd A(4,4);
A << 1.2, 2, 3, 1.2,
     2.3, 4, 2, 0.9,
     6, 4.1, 2.3, 2.3,
     2, 1.2, 0.5, 2.5;

int rank = A.colPivHouseholderQr().rank();
std::cout << "Exactly Constrained (rank " << rank
          <<")" <<std::endl<<std::endl;

printSolutions(A,b);
```

Output:

```
Exactly Constrained (rank 4)

Using left pseudoinverse with .inverse() x =
-0.172115
 0.206089
0.0970424
 0.419361

Error magnitude: 3.80727e-15

Using right pseudoinverse with .inverse() x =
-0.172115
 0.206089
0.0970424
 0.419361

Error magnitude: 4.84444e-15

Using QR decomposition x =
-0.172115
 0.206089
0.0970424
 0.419361

Error magnitude: 2.48253e-16
```

Test Overconstrained Ax=b

```
Eigen::VectorXd b(4);
b << 1,1,1,1;

//overconstrained
Eigen::MatrixXd A(4,3);
A << 1.2, 2, 3,
     2.3, 4, 2,
     6, 4.1, 2.3,
     2, 1.2, 0.5;

int rank = A.colPivHouseholderQr().rank();
std::cout << "Overconstrained (rank " << rank
          <<")" <<std::endl<<std::endl;

printSolutions(A,b);
```

- Which pseudo-inverse will do better here?

Output:

```
Overconstrained (rank 3)

Using left pseudoinverse with .inverse() x =
0.0479331
 0.108243
 0.220953

Error magnitude: 0.708657

Using right pseudoinverse with .inverse() x =
0.0390625
 0.306641
0.0117188

Error magnitude: 0.881806

Using QR decomposition x =
0.0479331
 0.108243
 0.220953

Error magnitude: 0.708657
```

Test Underconstrained Ax=b

```
Eigen::VectorXd b(4);
b << 1,1,1,1;

//underconstrained
Eigen::MatrixXd A(4,5);
A << 1.2, 2, 3, 2.1, 2,
     2.3, 4, 2, 3.1, 1.1,
     6, -2.1, 2.3, 2.9, 0.9,
     -2, 1, 2.6, 1.9, 0.8;

int rank = A.colPivHouseholderQr().rank();
std::cout << "Underconstrained (rank " << rank
          <<")" <<std::endl<<std::endl;
printSolutions(A,b);
```

- Which pseudo-inverse will do better here?

Output:

```
Underconstrained (rank 4)

Using left pseudoinverse with .inverse() x =
  0.375
   0.25
-0.3125
   0.25
      0

Error magnitude: 2.2694

Using right pseudoinverse with .inverse() x =
-0.0297742
-0.0163173
  0.191671
  0.246306
-0.0119457

Error magnitude: 2.60607e-15

Using QR decomposition x =
-0.0325217
-0.0191953
   0.18072
  0.254884
         0

Error magnitude: 2.71948e-16
```

Test singular A

```
Eigen::VectorXd b(4);
b << 1,1,1,1;


//A is singular
Eigen::MatrixXd A(4,4);
A.setIdentity();
A(0,0) = 0;

int rank = A.colPivHouseholderQr().rank();
std::cout << “Singular (rank " << rank
          <<")" <<std::endl<<std::endl;
printSolutions(A,b);
```

Output:

```
Singular (rank 3)

Using left pseudoinverse with .inverse() x =
nan
  1
  1
  1

Error magnitude: nan

Using right pseudoinverse with .inverse() x =
nan
nan
nan
nan

Error magnitude: nan

Using QR decomposition x =
0
1
1
1

Error magnitude: 1
```

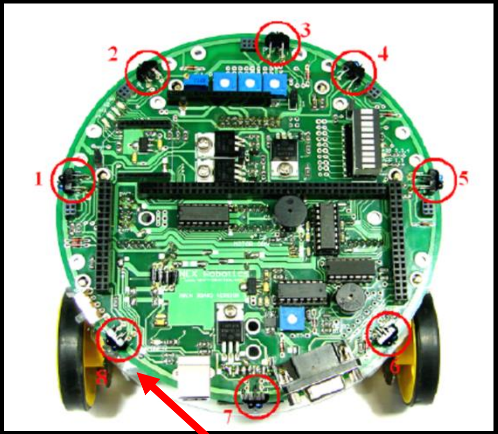
Maybe QR decomposition is *too* stable?

Lesson: Check rank before solving Ax = b!

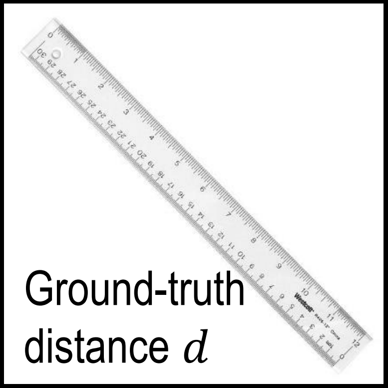
Example: Calibration using Least-Squares

- Suppose you have a IR range sensor that reports a distance x, but the sensor is noisy
- You'd like to calibrate this sensor using a ruler as ground-truth
- You measure distance to an object from a series of positions and record the measurements as v
- The corresponding ground-truth distances are d

Fire Bird V Mobile Robot



Measures distance v



- Goal: Find x_1 and x_2 for the function $x_1 v + x_2 = d$ that minimize the sum of squared errors.
  - This is the "Linear Least-Squares" problem

Example: Calibration using Least-Squares

- How do we turn this into an $\mathbf{A}x = b$ problem?
  - Let's start by thinking about a single datapoint: v_1, d_1
  - We want an equation of the form $x_1 v_1 + x_2 = d_1$
  - We rewrite it as

$$[v_1 \ 1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = d_1$$

  - Let x be the vector of the unknowns, so $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. Then,

$$[v_1 \ 1]x = d_1$$

  - Then define $\mathbf{A} = [v_1 \ 1]$ and $b = d_1$ and we have

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

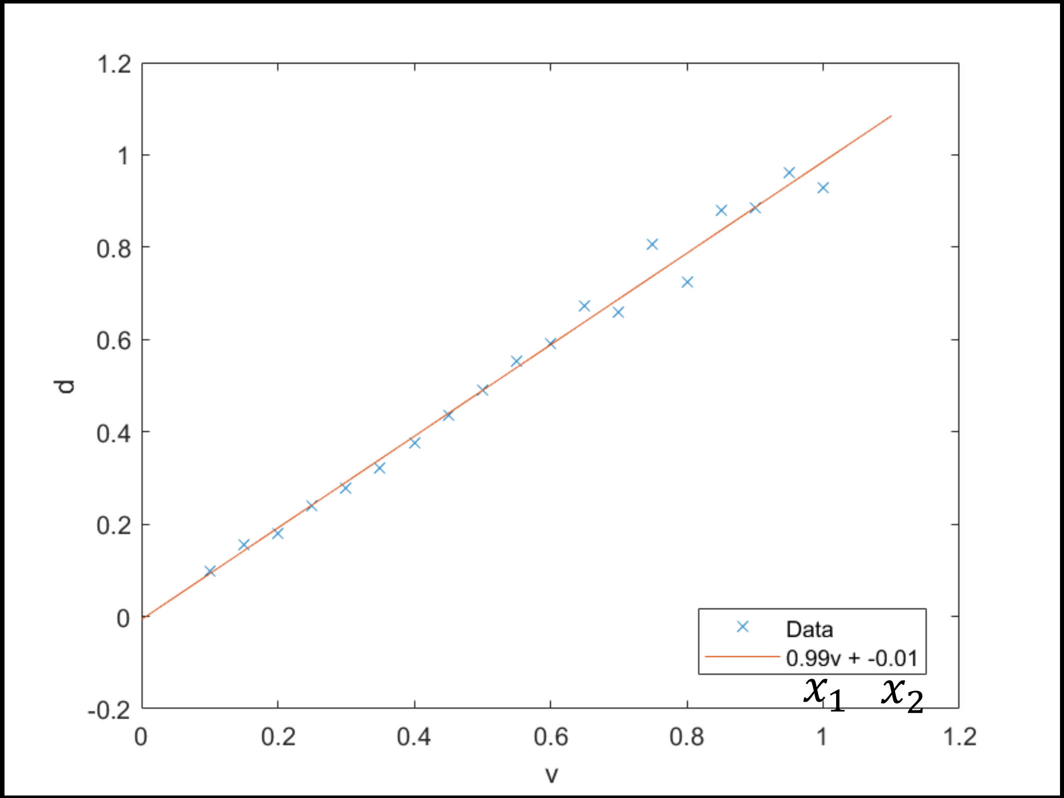
Example: Calibration using Least-Squares

- What about multiple datapoints?

$$\begin{array}{c}[v_1\ 1]x = d_1 \\ [v_2\ 1]x = d_2 \\ [v_3\ 1]x = d_3 \\ \vdots\end{array} \quad \Rightarrow \quad \underbrace{\begin{bmatrix} v_1 & 1 \\ v_2 & 1 \\ v_3 & 1 \\ & \vdots \end{bmatrix}}_{\mathbf{A}} x = \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \end{bmatrix}}_{b} \quad \Rightarrow \quad \mathbf{A}x = b$$

Example: Calibration using Least-Squares

Solve $\mathbf{A}x = b$ for x



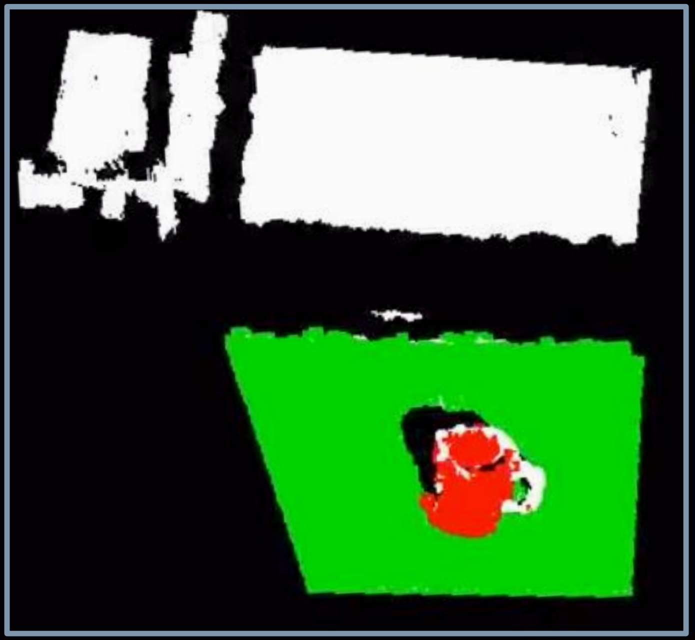
- Suppose the IR sensor outputs $v = 0.9\mathrm{m}$
- Then the true distance is estimated as $0.99 * 0.9\mathrm{m} - 0.01 = 0.88\mathrm{m}$

RANSAC

Fitting Models to Points

- Want to fit a parametrized model to a set of points, e.g.
  - Fit a plane
  - Fit a cylinder
- Problem: outliers (e.g. noise)
  - Don't know which points to fit to!

3D points from laser scan data:



Cylinder
Table
Other stuff

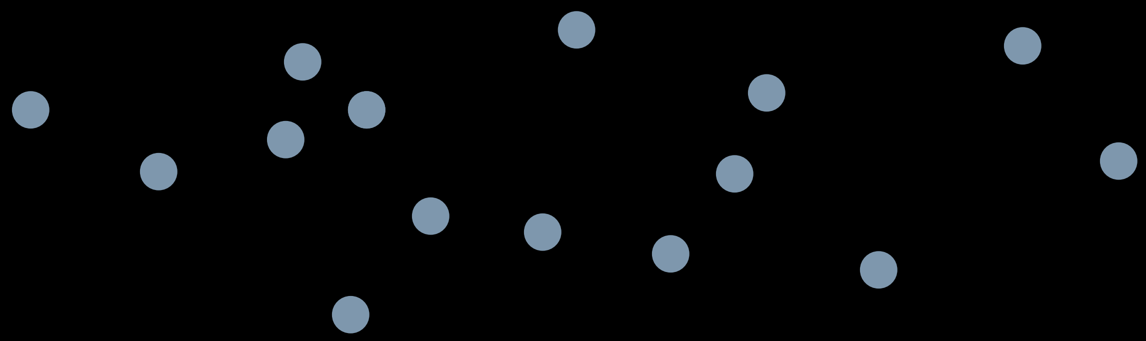
RANSAC Algorithm Sketch

- RANdom SAmple Consensus (RANSAC) samples models and returns the one with the best fit

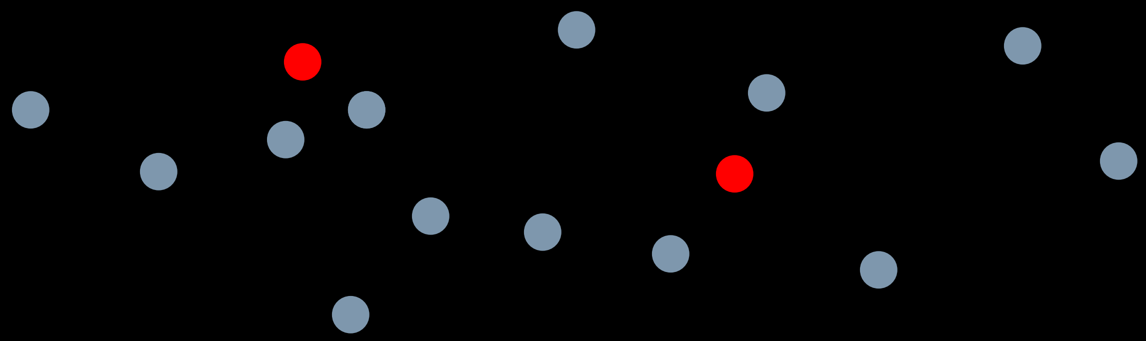
```
Input: Set of Points P, model type
Output: Model parameters
```

```
For some number of iterations
  1. Pick a random subset of points
  2. Fit the model to these random points
  3. Compute how many other points are close to the
model and how far they are
  4. If this is the best model so far, save it
Return best model found
```

RANSAC Line fitting example

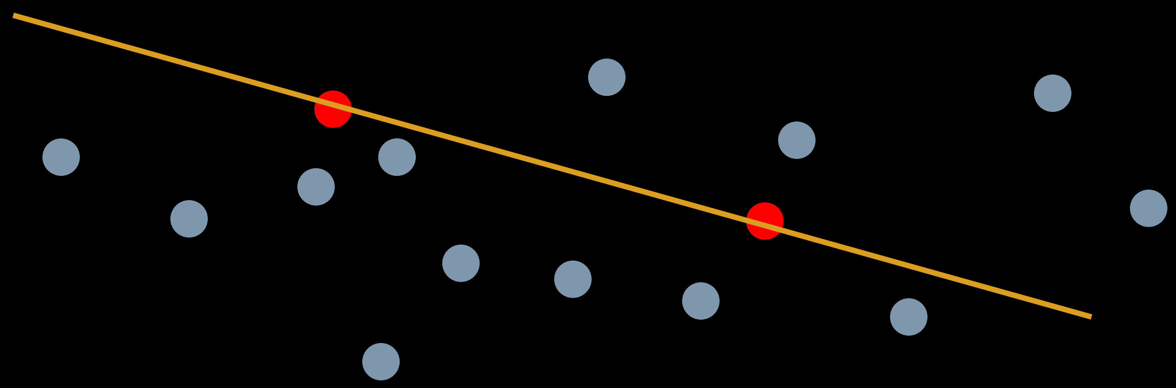


RANSAC Line fitting example



Pick R (hypothetical inliers)

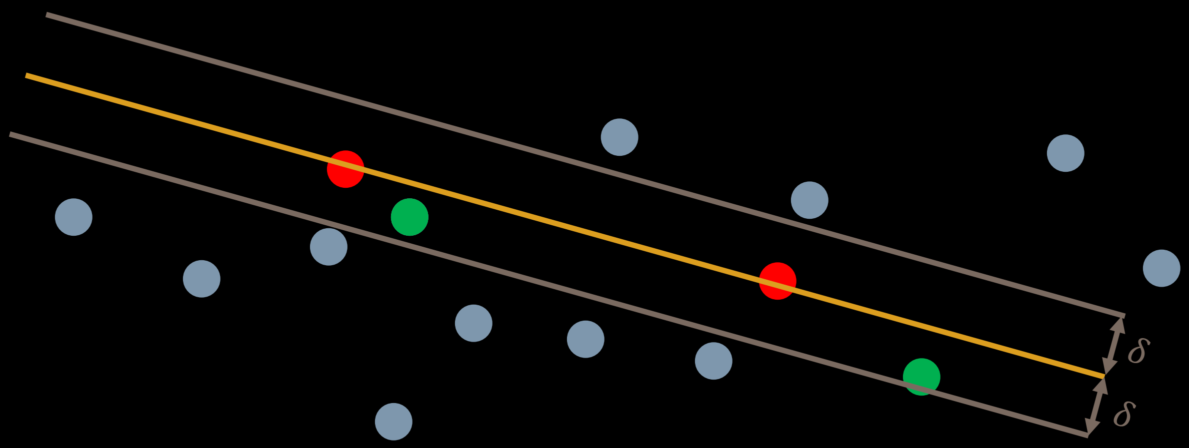
RANSAC Line fitting example



Pick R (hypothetical inliers)

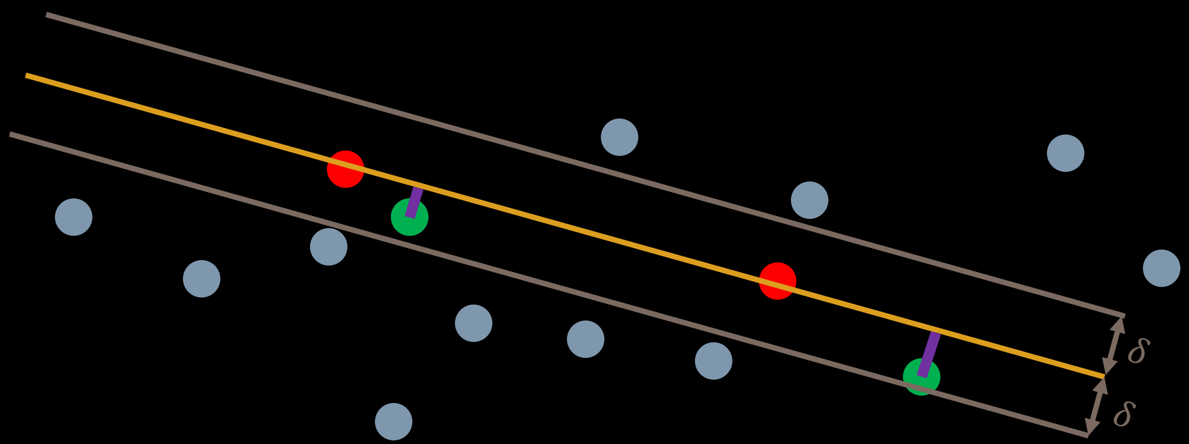
Fit Model to R

RANSAC Line fitting example



Pick R (hypothetical inliers)
Fit Model to R
Find C (consensus set)

RANSAC Line fitting example



Pick R (hypothetical inliers)

Fit Model to R

Find C (consensus set)

Compute Error of Model on $C \cup R$

RANSAC Algorithm

```
Input: Set of Points P, model type, K: # of iterations, δ: threshold for
inliers, N: minimum number of consensus points required

Output: Model parameters θ
```

$e_{best} \leftarrow \infty$

For $i \in \{1,2,\ldots,K\}$

  Pick a random subset $R \subset P$ //R is the set of hypothetical inliers (enough to fit model)

  $\theta \leftarrow$ Fit(model, R) //θ are the model parameters

  $C \leftarrow \{\emptyset\}$ //C is the consensus set

  For $p \in P \backslash R$ //For all points that weren't used yet

    If Error(p, model(θ)) < δ //Check if p is close to the model prediction

      $C \leftarrow C \cup p$ //Add p to the consensus set

  If $|C| > N$ //If we have enough consensus points

    $\theta \leftarrow$ Fit(model, $R \cup C$) //Re-fit the model parameters

    $e_{new} \leftarrow$ Error($R \cup C$, model(θ)) //Get new error

    If $e_{new} < e_{best}$ //If this is the best model so far, save it

      $e_{best} \leftarrow e_{new}$

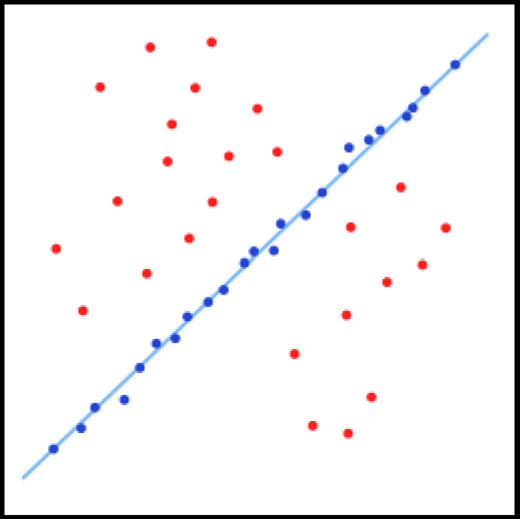
      $\theta_{best} \leftarrow \theta$

return θ_{best}

RANSAC Example

- Line fitting with extreme noise:

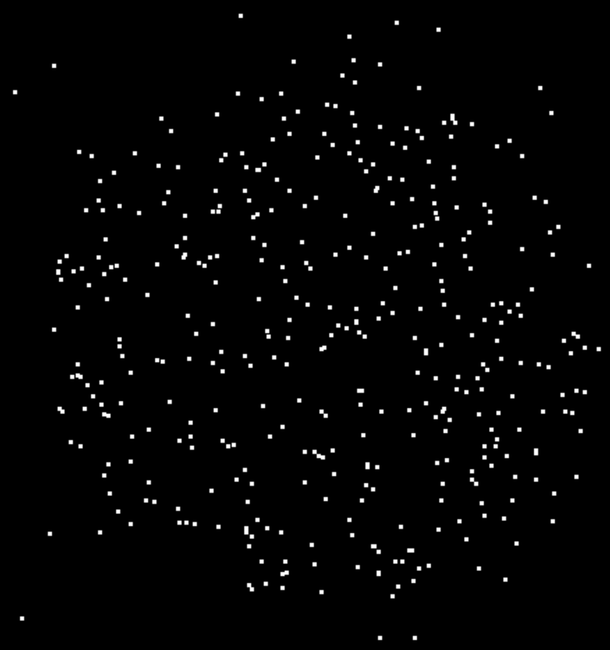
Input data



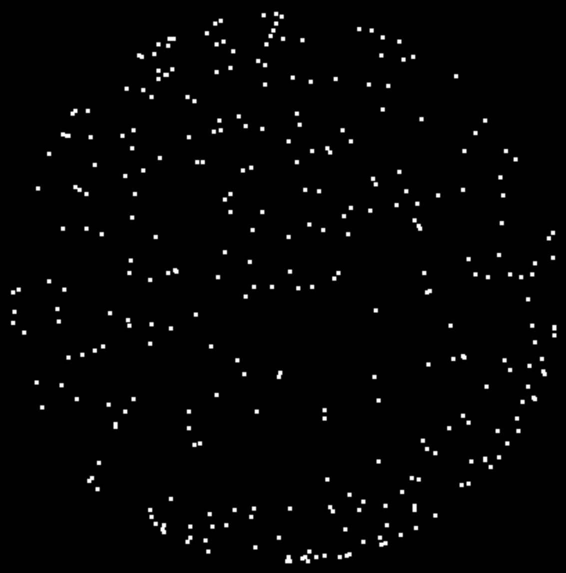
RANSAC output model with inliers in blue

RANSAC Example

- Fitting a sphere surface

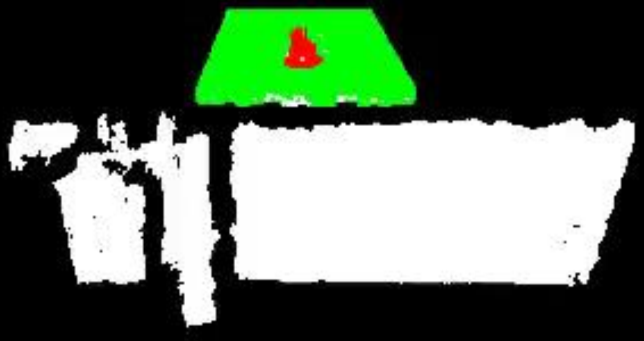


Input data



Inliers of RANSAC output model

RANSAC Example



RANSAC Advantages and Disadvantages

- Advantages
  - Fitting is robust to extreme noise in the data
  - Model type can be anything, as long a there is a `Fit(model,data)` function

- Disadvantages
  - Solution may not be optimal if number of iterations is too small
  - Thresholds (δ and N) are problem-specific

- Time vs. accuracy trade-off
  - More iterations increase computation time but make it more likely a good model will be produced

- The `Fit(model,data)` function should be fast!
  - Need to evaluate it at least once per iteration

Summary

- Least-squares is a way to solve Ax = b problems
  - Uses the pseudoinverse
  - Meaning of the result depends on if the system is over/under/exactly constrained

- RANSAC is a way to fit models to data
  - Works well with noise, can use any parameterized model (e.g. plane, cylinder)
  - Disadvantage: Need to set problem-specific thresholds

Homework

- Homework 4 is out soon
- No class Monday - have a good break!