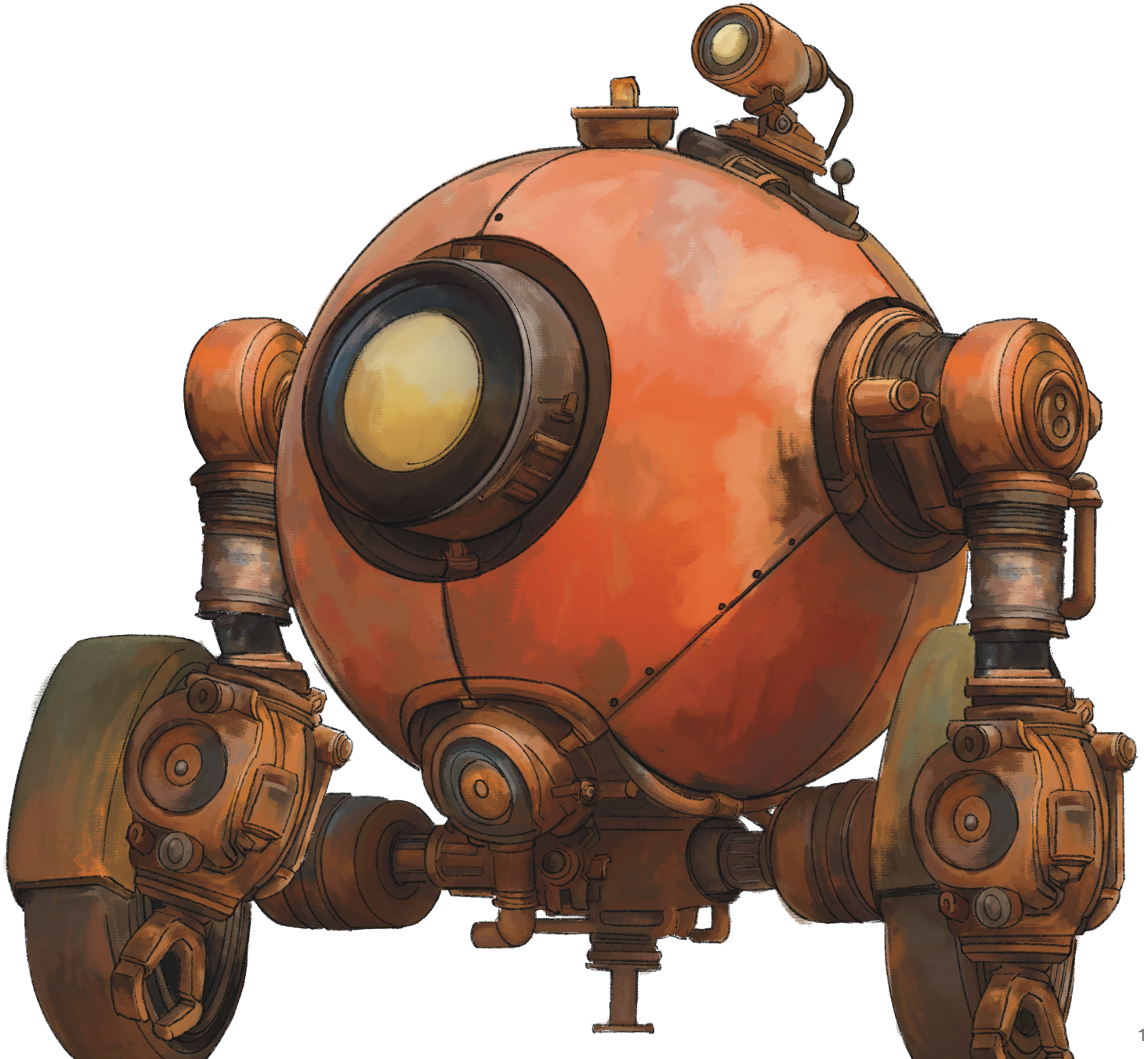
CS 3630, Fall 2026

Lecture 16:

A Logistics Robot:

Planning & Learning

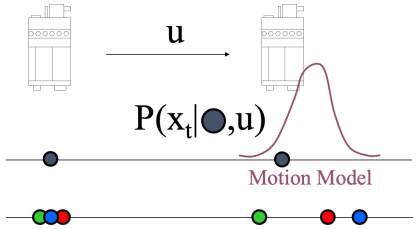


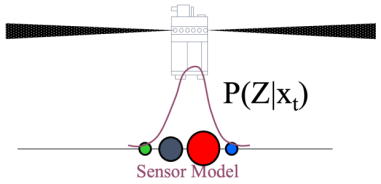

Recap

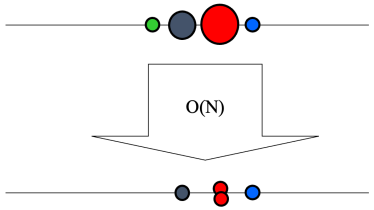
Recap: Bayes filtering

Prediction

The motion model propagates our belief after an action.







Measurement

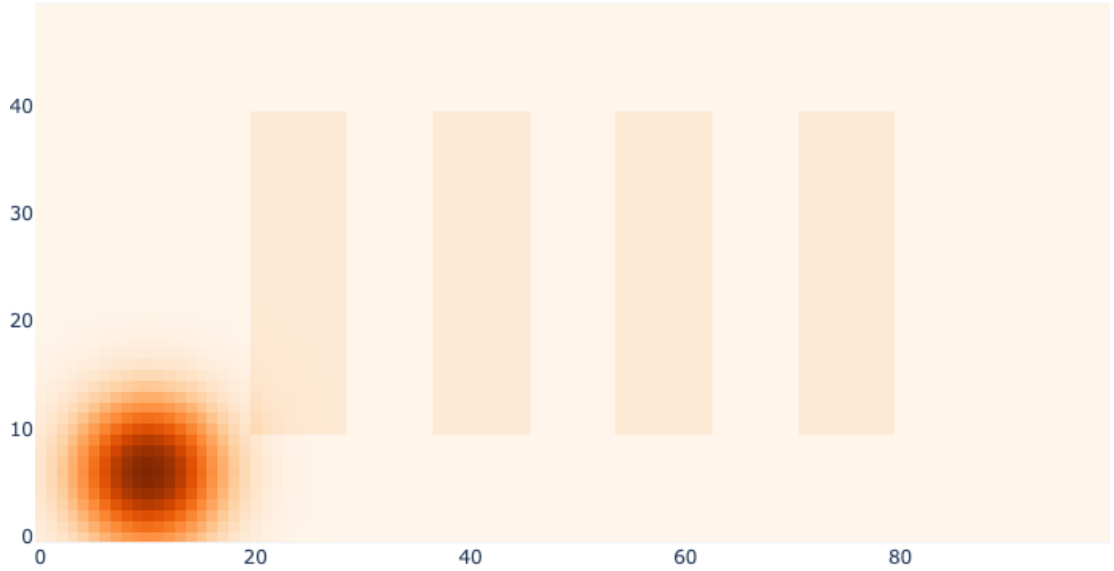
The sensor likelihood reweights the predicted belief.

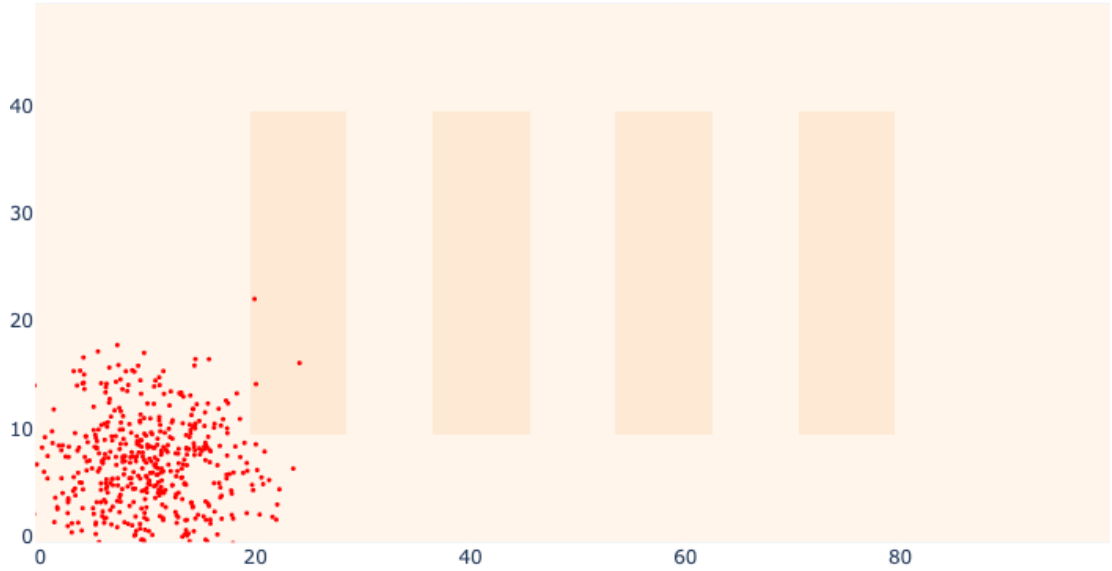
```
# Conceptual Bayes filter
belief = predict(belief, u, model)
belief *= likelihood(z, states)
belief /= belief.sum()
```

The models tell us how actions and measurements relate to state.

Recap: a belief can have several modes

A grid stores probability at fixed cells. Particles store sampled states and weights.





A single point estimate can hide ambiguity that matters for choosing an action.

Today: planning and learning

4.5 Planning for Logistics

Value iteration on a warehouse grid, then acting with uncertain state.

4.6 Some System Identification

Learning the sensor and motion models used by the robot.

How should the robot act, and where do its models come from?





Planning for Logistics
Section 4.5

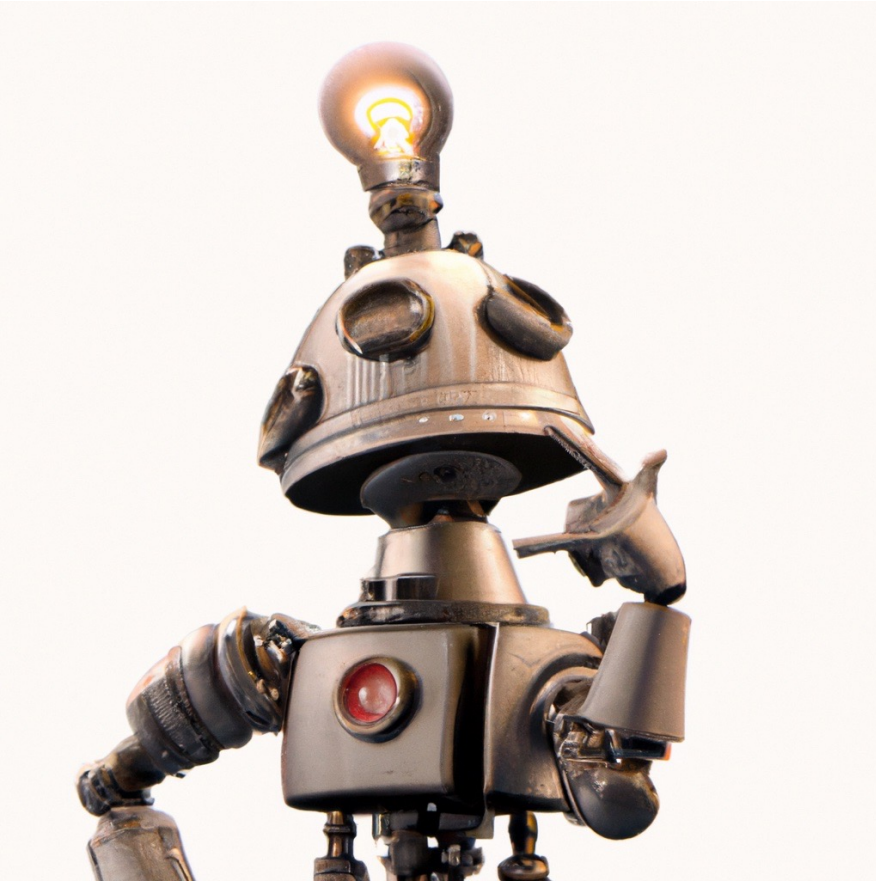
A warehouse grid as an MDP

State

A cell (i, j) in a 50×100 grid gives 5,000 states.

Actions

Move left, right, up, or down. The notebook starts with deterministic motion.



A policy selects an action for each state.

Recall: Value Iteration

$$V^{k+1}(x) = \max_a Q\left(x, a; V^k\right)$$

$$Q(x, a; V) = \overline{R}(x, a) + \gamma \sum_{x'} P(x'|x, a) V(x')$$

```
def update_V(x, V):
    Q_x = [Q_value(x, a, V)
           for a in range(4)]
    return max(Q_x)
```

Each sweep asks every cell which action currently looks best.



A dense transition table is *very* expensive

100 million entries

5,000 states × 4 actions × 5,000 possible successors.

Local transitions

Most probabilities are zero. Store sparse transitions or calculate Q directly.

```
n_states = 50 * 100
n_actions = 4

entries = (n_states * n_actions * n_states)
# 100_000_000

# One deterministic successor:
x_next = transition(x, a)
```

The notebook computes Q as a function instead of storing the full table.



Deterministic motion on the grid

The action chooses one neighboring cell. At the map boundary, the robot stays put.

```
i, j = x
if a == 0: next = (i, j-1) if j > 0  else x  # LEFT
if a == 1: next = (i, j+1) if j < 99 else x  # RIGHT
if a == 2: next = (i-1, j) if i > 0  else x  # UP
if a == 3: next = (i+1, j) if i < 49 else x  # DOWN
```

With one successor, the expected future value reduces to V[next].



Rewards and proximity penalties

Goal: (35, 50). The robot earns +100 on arrival and pays for proximity to obstacles.

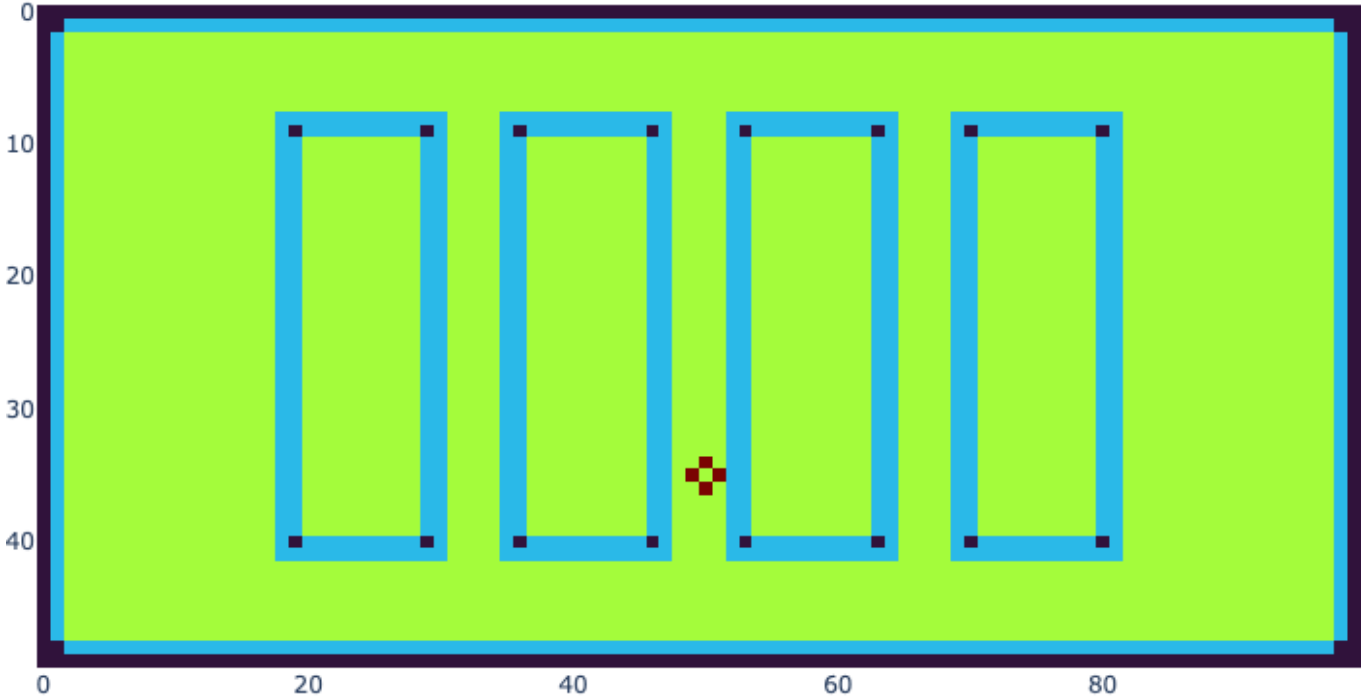
```
if x == goal: return 0
if logistics.base_map[i, j]: return 0
reward = 100 if next == goal else 0
r, c = next
if logistics.proximity_map_on[i, j]: reward -= 50
if logistics.proximity_map_on[r, c]: reward -= 50
return reward + 0.975 * V[r, c]
```

Two −50 terms: one for the current cell and one for the next cell.



Value iteration in action

```
V_k = np.zeros((50, 100))
error = float("inf")
while error > 1e-5:
    V_plus = np.empty((50, 100))
    for i, j in states:
        V_plus[i, j] = update_V((i, j), V_k)
    error = np.max(
        np.abs(V_plus - V_k))
    V_k = V_plus
```



Stopping condition

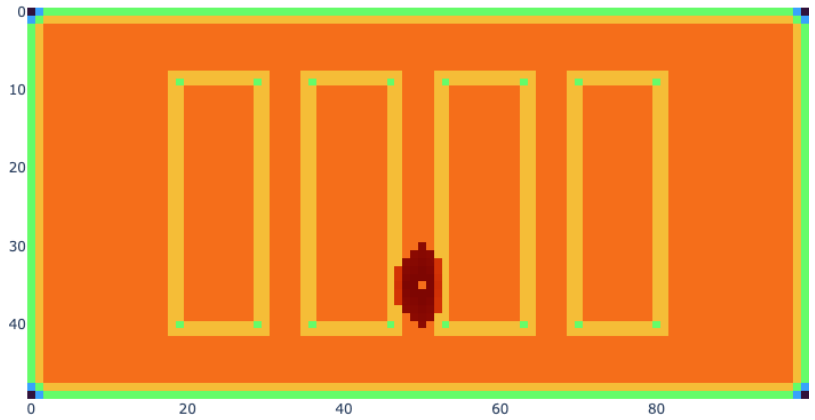
Stop when the largest absolute change is at most 10^{-5}.

One old value image

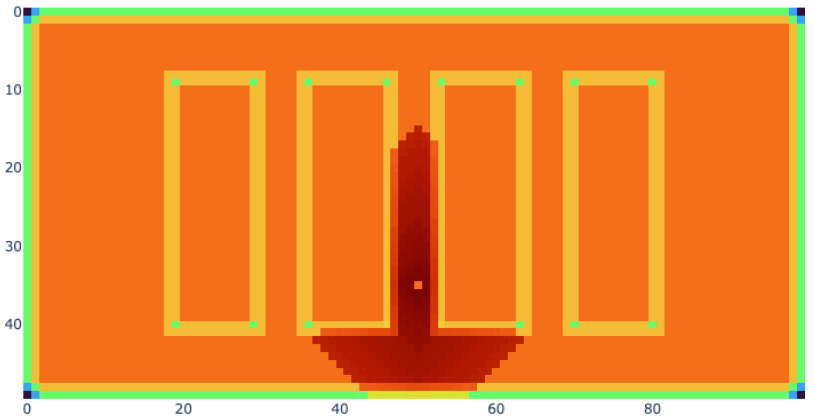
All cells read the same V_k during a sweep.

Value propagates one transition per sweep

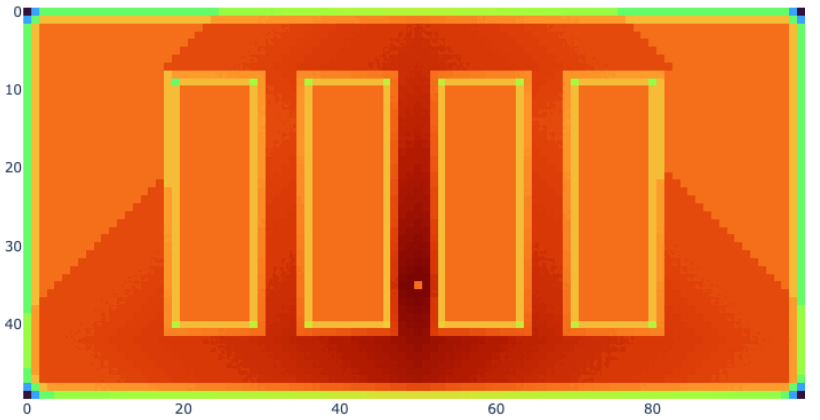
Sweep 5



Sweep 20



Sweep 60



Neighbors learn about the goal first. Their neighbors learn on the next sweep.

Discussion: why can the deterministic example stop after a finite number of sweeps?

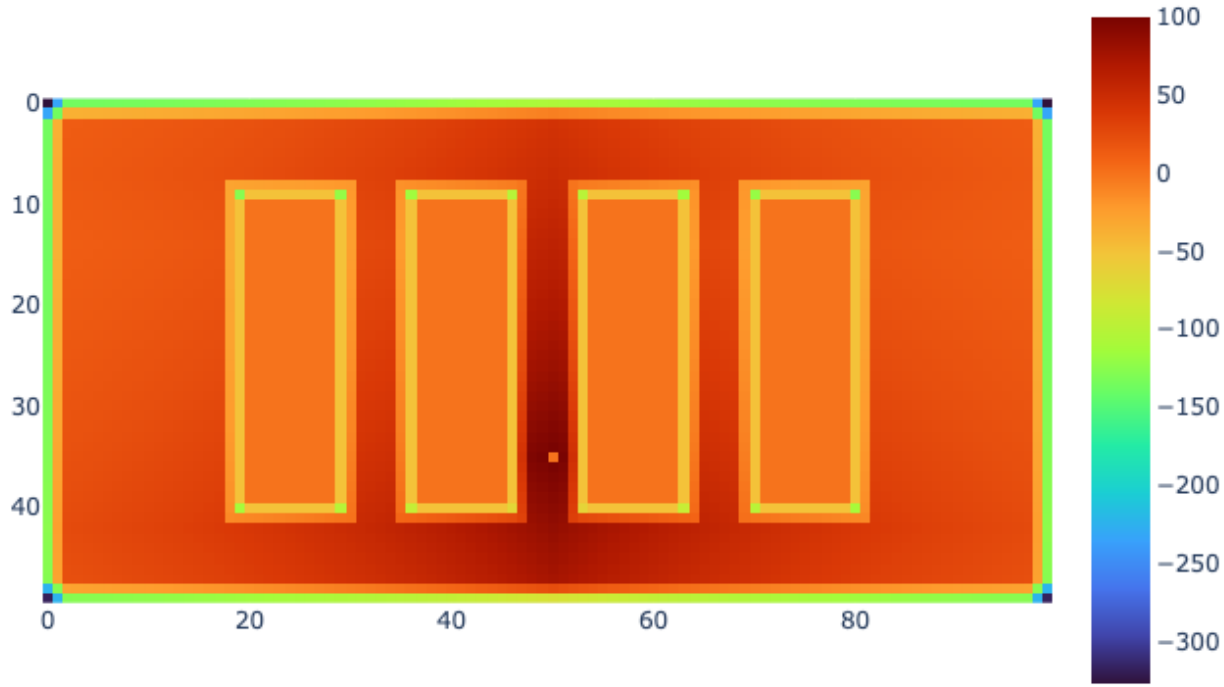
The converged value function

High value near the goal

Discounting reduces the value of a reward that takes longer to reach.

Low value near obstacles

Proximity penalties make risky cells less attractive.



Notebook output: after 86 iterations, error = 0.0.

Extracting a policy

Greedy action

Choose the action with the largest Q value under the converged V.

The role of discounting

If only a goal bonus matters, fewer steps give a larger discounted return.

```
def policy(x, V):
    q = [Q_value(x, a, V)
         for a in range(4)]
    return int(np.argmax(q))

# Arrival reward after d steps:
# return = 100 * gamma**(d - 1)
```

The immediate reward matters, especially near the goal and obstacles.



Closing the loop: Sense-Think-Act

Estimate the state

Use sensor data and the previous action to update the belief.

Execute the policy

Select an action, execute it, and process the next measurement.

```
# Conceptual control loop
while running:
    z = sense()
    belief = bayes_filter(
        belief, u_previous, z)
    x_hat = estimate(belief)
    u = policy(x_hat, V)
    execute(u)
    u_previous = u
```

What if the posterior has two plausible locations that require different actions?




Some System Identification
Section 4.6

Learning the robot's models

Sensor model

How does a known state predict a measurement?

Motion model

How do the current state and control predict the next state?



Fit the linear mapping, then estimate noise from the residuals.

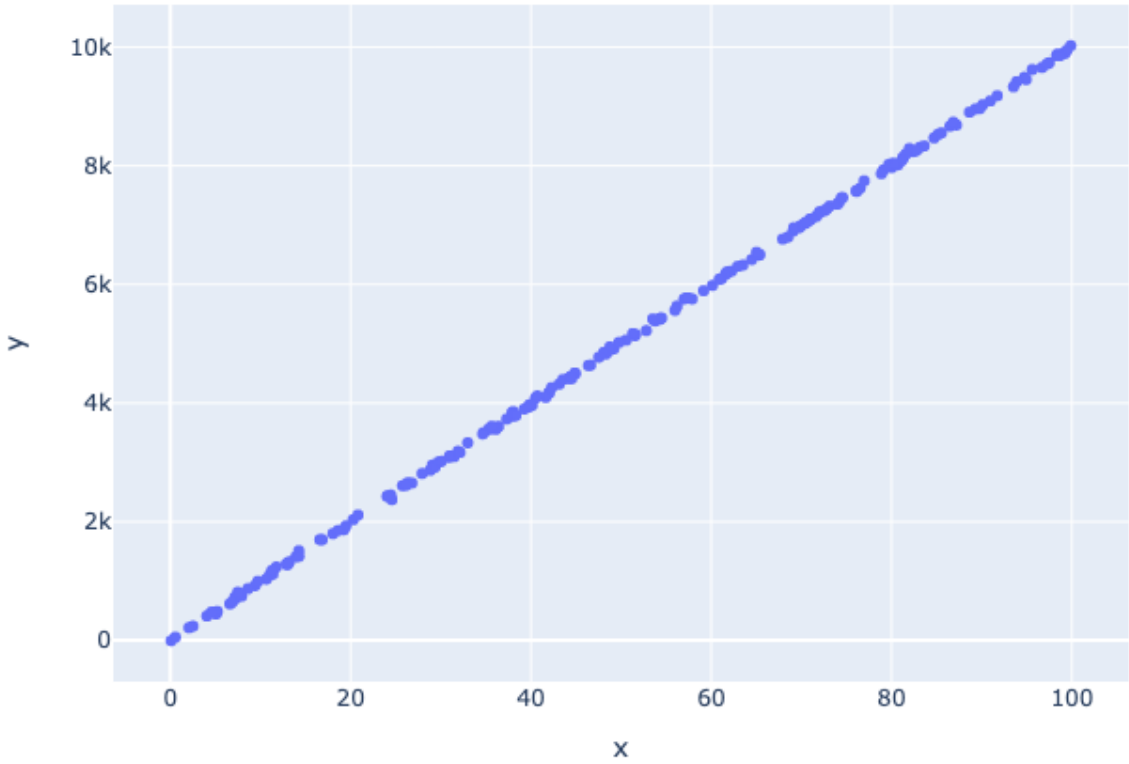
A scalar sensor model

$$z_k = Cx_k + n_k$$

Ground truth x_k is known. Noise n_k has mean zero and variance $R = \sigma^2$.

Notebook simulation

200 positions across 100 m. $C = 100$ cm/m and $\sigma = 30$ cm.



The slope converts meters to centimeters.

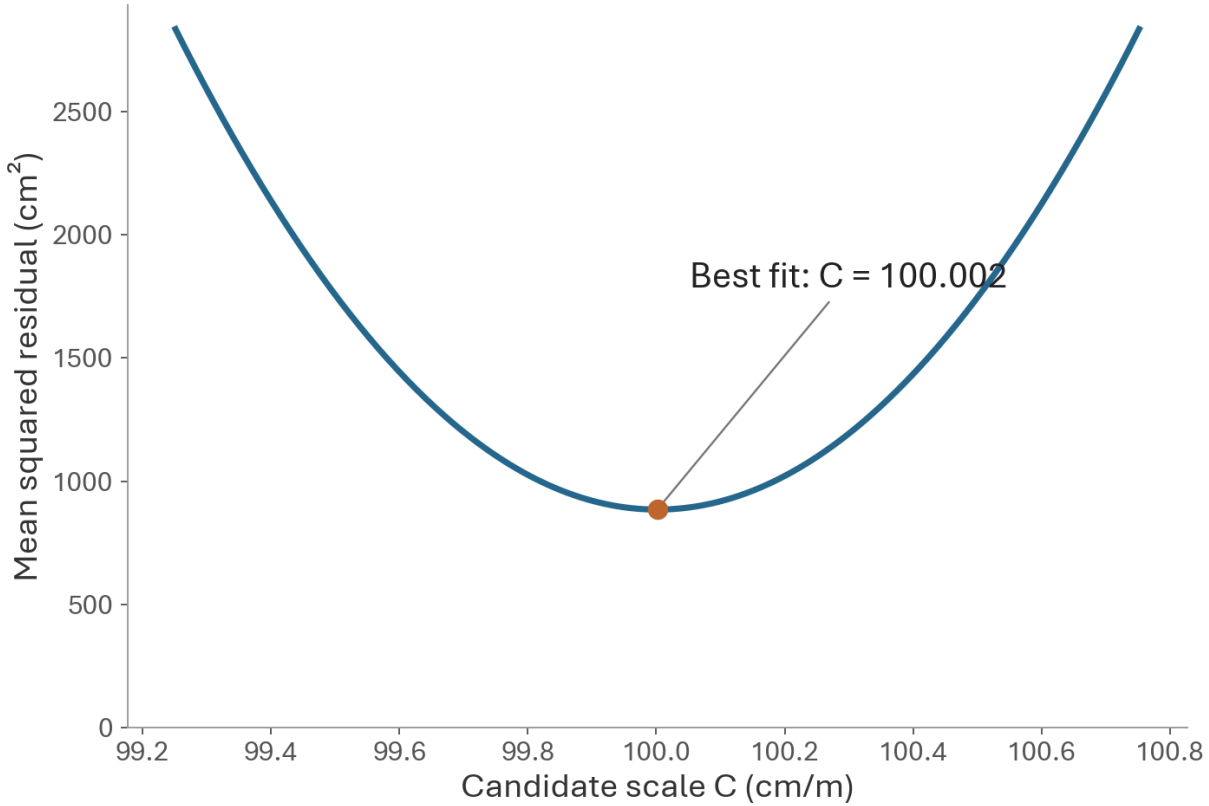
Maximum likelihood becomes least squares

A score for each scale

Each candidate C predicts a line. Minimize $\sum_k (C x_k - z_k)^2$ to find its best fit.

The best-fitting scale

The minimum of the curve gives the least-squares estimate.




```
C_hat = np.sum(X * Z) / np.sum(X * X)
```

Gaussian noise makes least squares a maximum-likelihood fit. Simulation of the notebook model.

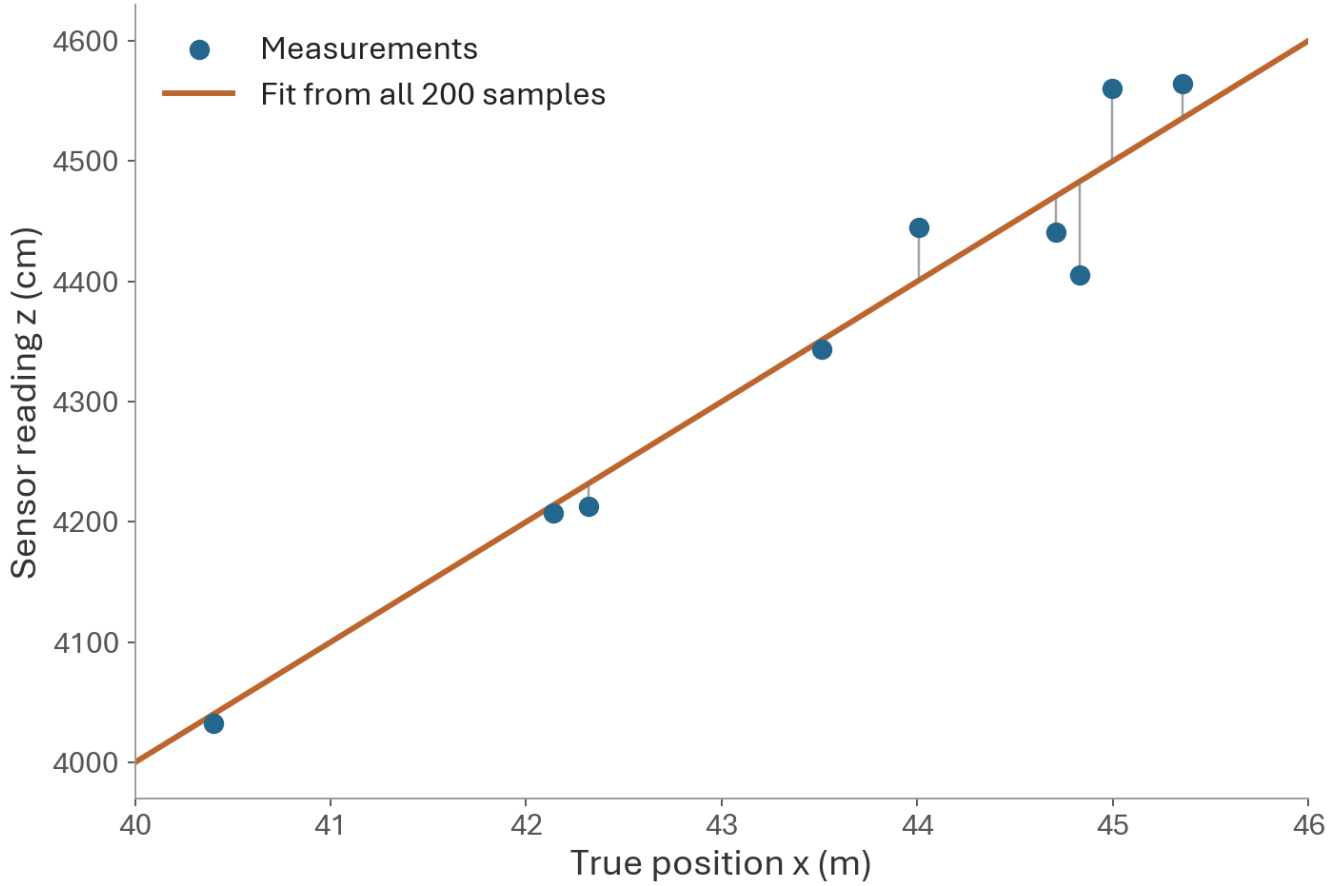
The fitted scale predicts the measurements

Fit all 200 samples

Estimated $C = 100.002$ cm/m.
True $C = 100$ cm/m.

Zoom into a short interval

Dots show sensor readings. The vertical gaps are the residuals left after fitting.



A zoom of the new notebook-model simulation makes the 30 cm sensor noise visible.

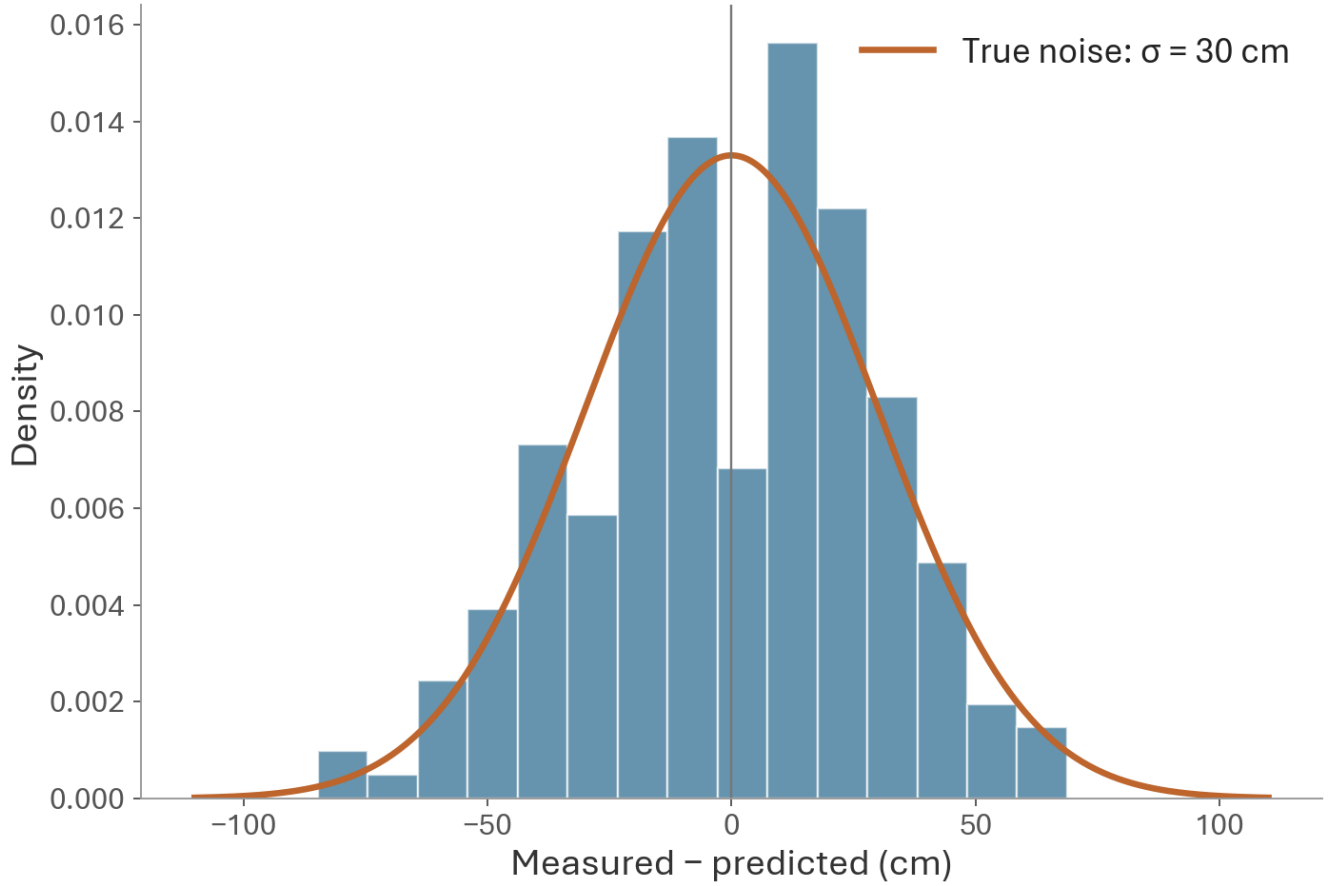
Residuals reveal the sensor noise

Width measures uncertainty

Estimated $\sigma = 29.82$ cm.
True $\sigma = 30$ cm.

One fitted parameter

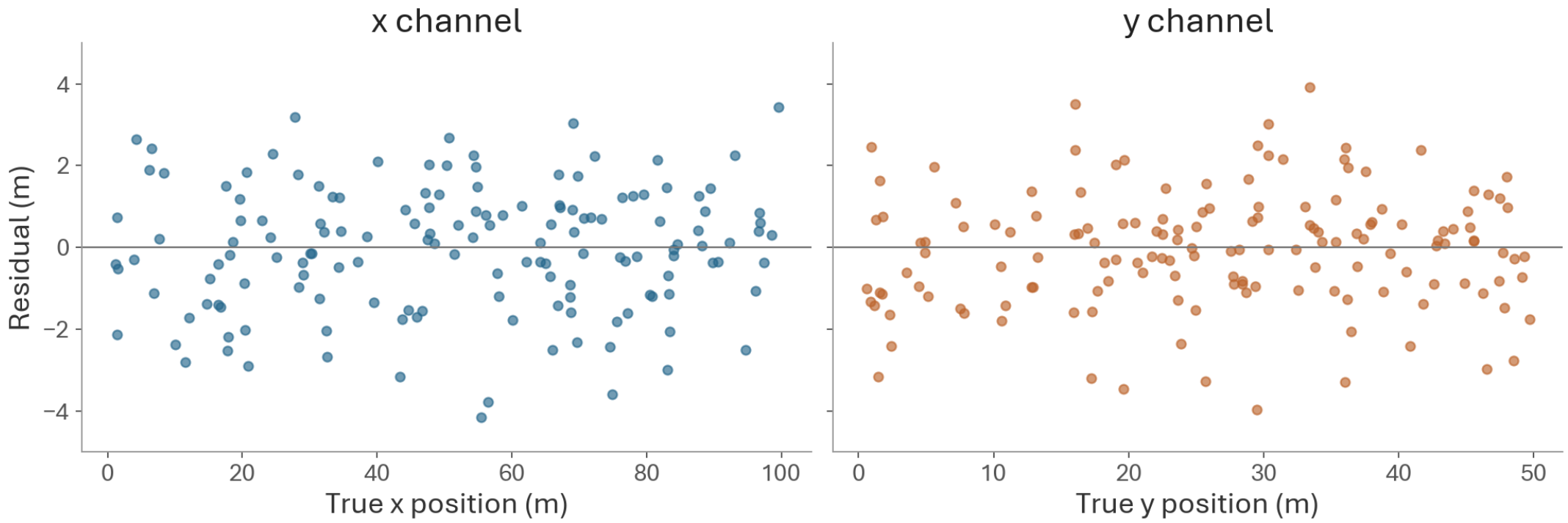
The unbiased variance divides squared residuals by $N - 1$.
Maximum likelihood divides by N.



Histogram of measured − predicted values. Orange curve: the noise used in the simulation.

Independent coordinate calibration

When each sensor channel measures only its matching coordinate, fit the two scales separately.



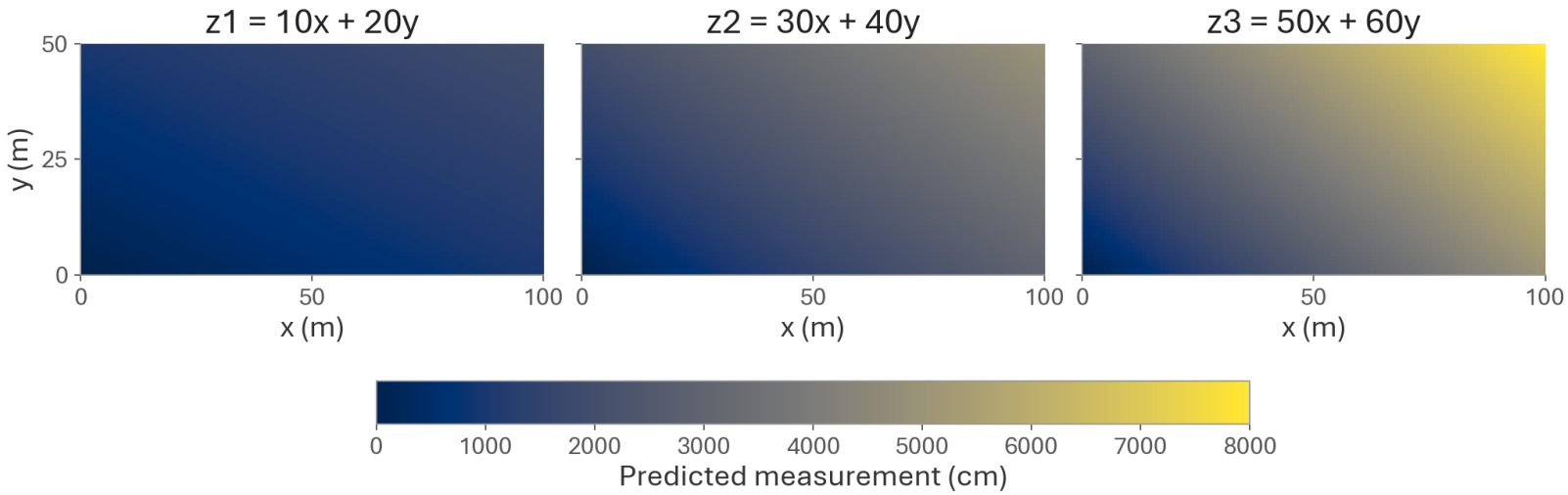
Illustrative GPS-like simulation: independent 1.5 m noise in x and y.

Multivariate linear regression

$$z_k = Hx_k + n_k \tag{4.63}$$

$x_k \in \mathbb{R}^n, \qquad z_k \in \mathbb{R}^m, \qquad H \in \mathbb{R}^{m \times n}$

Each row of H predicts one measurement from the same input state.



The notebook uses two state coordinates and three measurement channels.

The negative log likelihood

Gaussian measurement noise gives a weighted squared-error objective.

$$\hat{H} = \arg\min_{H} \sum_{k} \frac{1}{2} (Hx_k - z_k)^{\mathrm{T}} R^{-1} (Hx_k - z_k) \tag{4.64}$$

Assume isotropic noise, as in the book: $R = \sigma^2 I$

$$\hat{H} = \arg\min_{H} \frac{1}{2\sigma^2} \sum_{k} \sum_{i} (H_i x_k - z_{ki})^2 \tag{4.65}$$

```
# Sum over training samples and measurement coordinates
error = H @ X - Z
J = np.sum(error**2) / (2 * sigma**2)
```

Each row predicts one output. The same variance is used for every output coordinate.



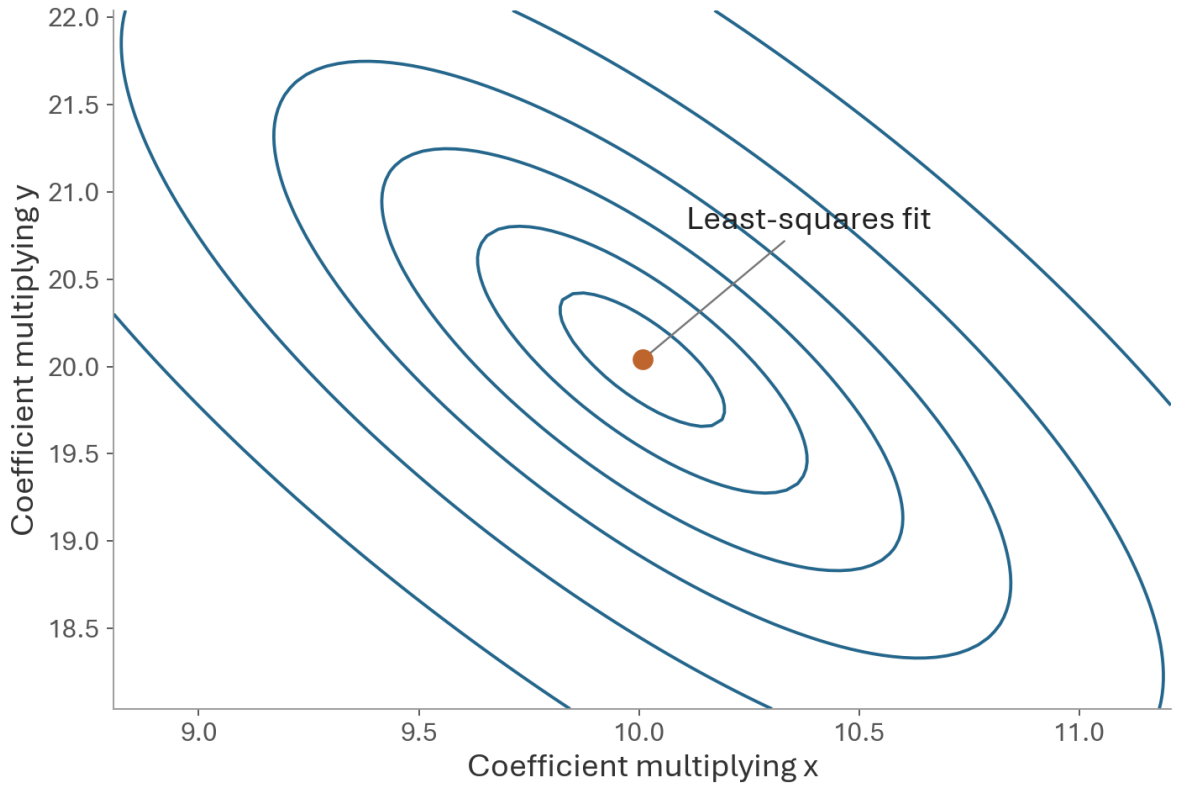
Two inputs, one output coordinate

$$x_k = \begin{bmatrix} x_{k1} \\ x_{k2} \end{bmatrix}, \qquad H_i = [a \quad b]$$

$$e_k = ax_{k1} + bx_{k2} - z_{ki}$$

$$J_i(a,b) = \frac{1}{2\sigma^2} \sum_k {e_k}^2$$

Fit a and b using all training examples for this output.



The first notebook channel has coefficients 10 and 20. Contours show equal squared error.

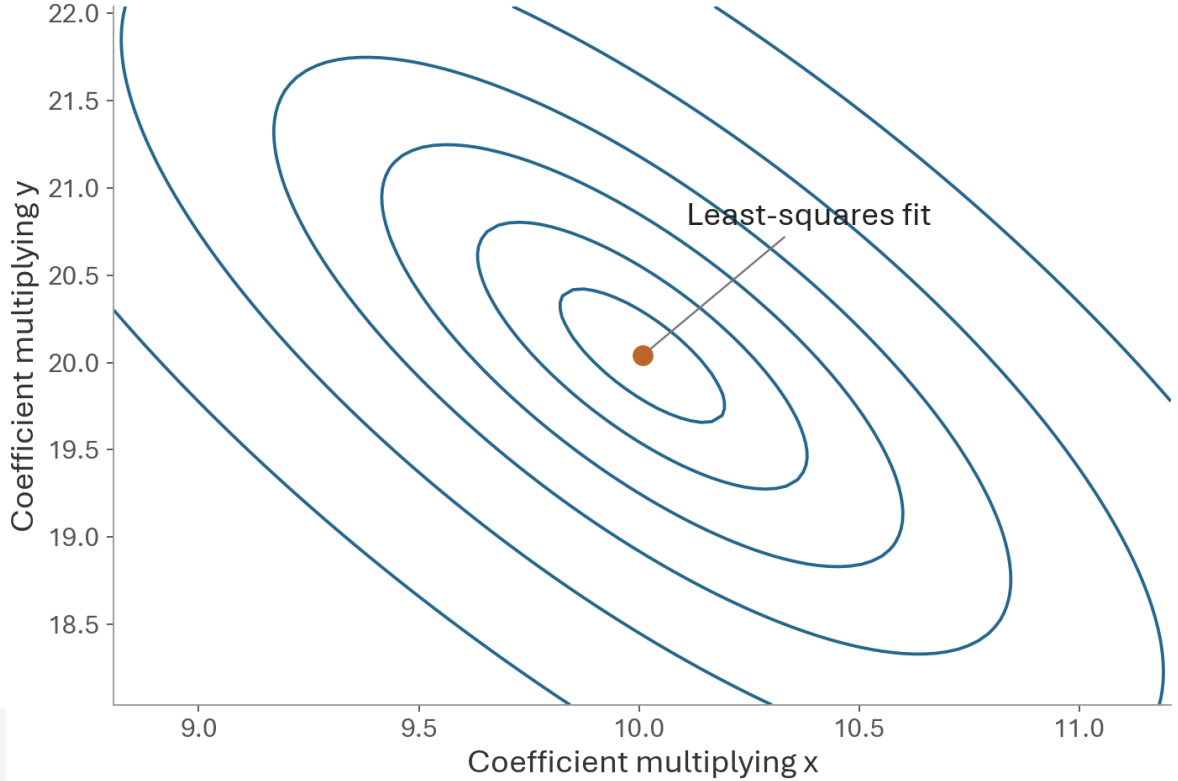
Partial derivatives for the two coefficients

$$e_k = ax_{k1} + bx_{k2} - z_{ki}$$

Apply the chain rule to each squared error:

$$\frac{\partial J_i}{\partial a} = \frac{1}{2\sigma^2}\sum_k 2e_k x_{k1} = \frac{1}{\sigma^2}\sum_k e_k x_{k1}$$

$$\frac{\partial J_i}{\partial b} = \frac{1}{2\sigma^2}\sum_k 2e_k x_{k2} = \frac{1}{\sigma^2}\sum_k e_k x_{k2}$$




```
e = a * X[0] + b * X[1] - Z[i]
dJ_da = np.sum(e * X[0]) / sigma**2
dJ_db = np.sum(e * X[1]) / sigma**2
```

The factor 2 from differentiating the square cancels the factor 1/2.

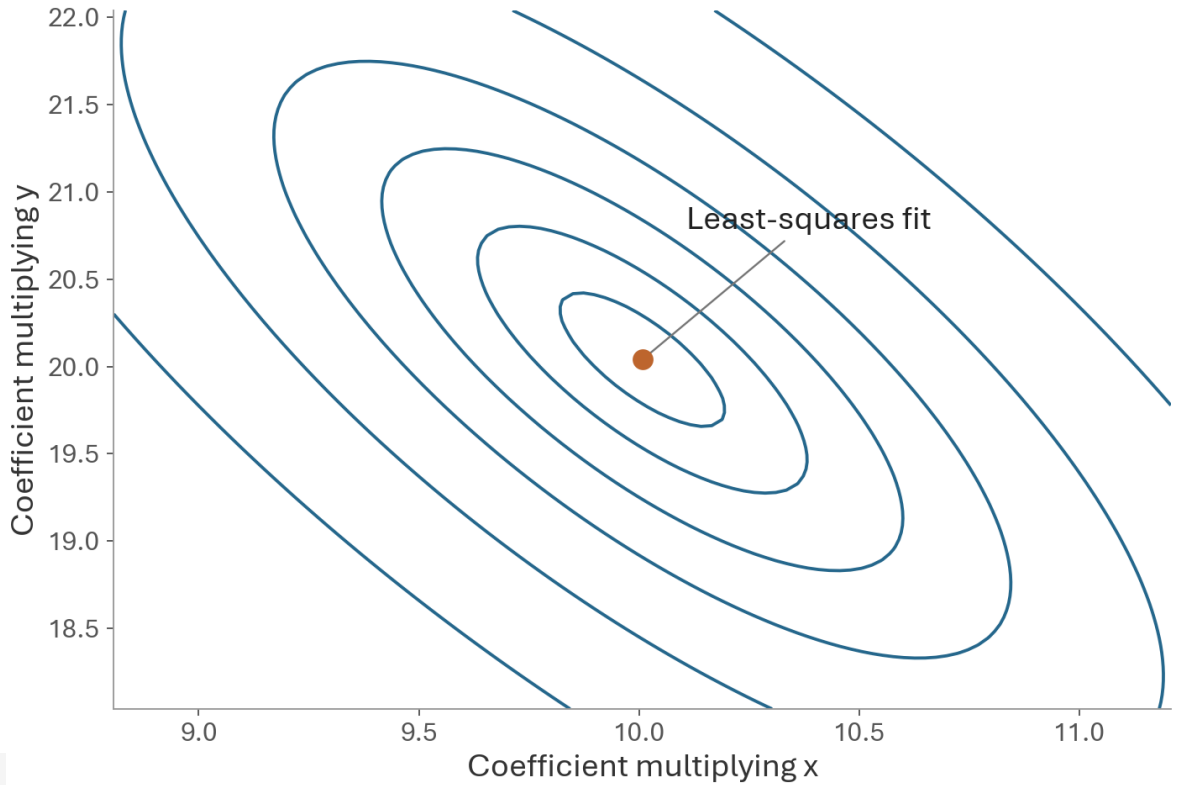
The zero-gradient condition

$$\begin{bmatrix} \frac{\partial J_i}{\partial a} \\ \frac{\partial J_i}{\partial b} \end{bmatrix} = \frac{1}{\sigma^2} \sum_k \begin{bmatrix} x_{k1} \\ x_{k2} \end{bmatrix} e_k = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Write the same condition for any number of input coordinates:

$$\sum_k x_k \left({x_k}^{\mathrm{T}} {\hat{H}_i}^{\mathrm{T}} - z_{ki} \right) = 0$$

$$H_i x_k = {x_k}^{\mathrm{T}} {H_i}^{\mathrm{T}}, \qquad {H_i}^{\mathrm{T}} = \begin{bmatrix} a \\ b \end{bmatrix}$$




```
gradient = X @ error_i / sigma**2
# At the fitted coefficients: gradient == 0
```

At the least-squares solution, the input-weighted residuals sum to zero.

The solution for all output coordinates

$$\hat{H}^{\mathrm{T}} = \left(\sum_k x_k {x_k}^{\mathrm{T}}\right)^{-1} \sum_k x_k {z_k}^{\mathrm{T}} \tag{4.68}$$

$$\hat{H}^{\mathrm{T}} = \left(XX^{\mathrm{T}}\right)^{-1} XZ^{\mathrm{T}}$$

$$X \in \mathbb{R}^{n \times N}, \qquad Z \in \mathbb{R}^{m \times N}, \qquad \hat{H}^{\mathrm{T}} \in \mathbb{R}^{n \times m}$$

```
# Notebook implementation
H_transpose = np.linalg.inv(X @ X.T) @ (X @ Z.T)
H_optimal = H_transpose.T

# Equivalent least-squares solve (avoids the explicit inverse)
H_optimal = np.linalg.lstsq(X.T, Z.T, rcond=None)[0].T
```

Each output fits one row of H using the same input samples.



Noise variance estimation

$$\widehat{\sigma^2} = \frac{1}{mN - mn} \sum_k \sum_i \left(\hat{H}_i x_k - z_{ki} \right)^2 \tag{4.69}$$

Count the scalar residuals, then subtract the fitted coefficients.

$$mN - mn = 3{\times}200 - 3{\times}2 = 600 - 6 = 594$$

```
residual = H_optimal @ X - Z
sigma_hat = np.sqrt(
    np.sum(residual**2) / (3*N - 6))
```

This estimates variance without bias. Maximum likelihood divides by the number of scalar residuals.



Multivariate ML code example

```
H = np.array([[10, 20],
              [30, 40],
              [50, 60]])
R = 30**2
N = 200
X = rng.uniform(low=[0, 0],
                high=[100, 50],
                size=(N, 2)).T
Z = rng.normal(H @ X, np.sqrt(R))
```

```
H_transpose = np.linalg.inv(X @ X.T) @ (X @ Z.T)
H_optimal = H_transpose.T
```

Stored notebook estimate

$$\hat{H} = \begin{bmatrix} 10.10 & 19.86 \\ 30.06 & 39.85 \\ 50.03 & 59.92 \end{bmatrix}$$

Estimated standard deviation

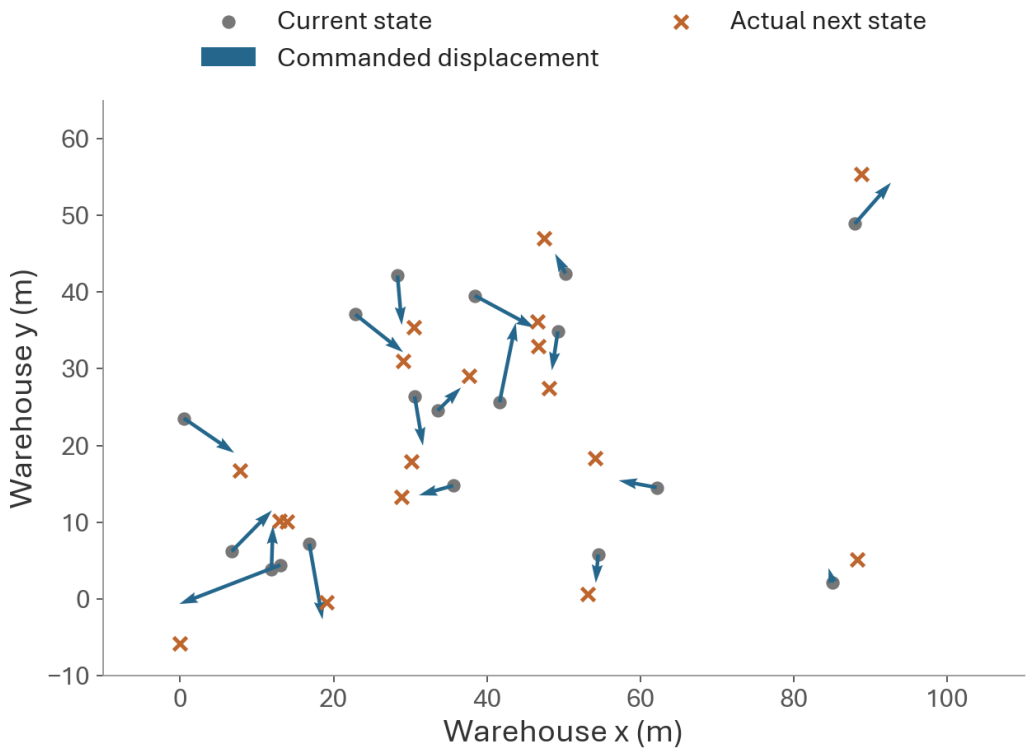
$$\hat{\sigma} \approx 29.94 \text{ cm}$$

The estimates are close to the coefficients and 30 cm noise used to generate the data.



Learning a linear motion model

$$x_{k+1} = Ax_k + Bu_k + w_k$$



Fit [A B] using stacked [state; control] inputs. These are 18 simulated transitions.

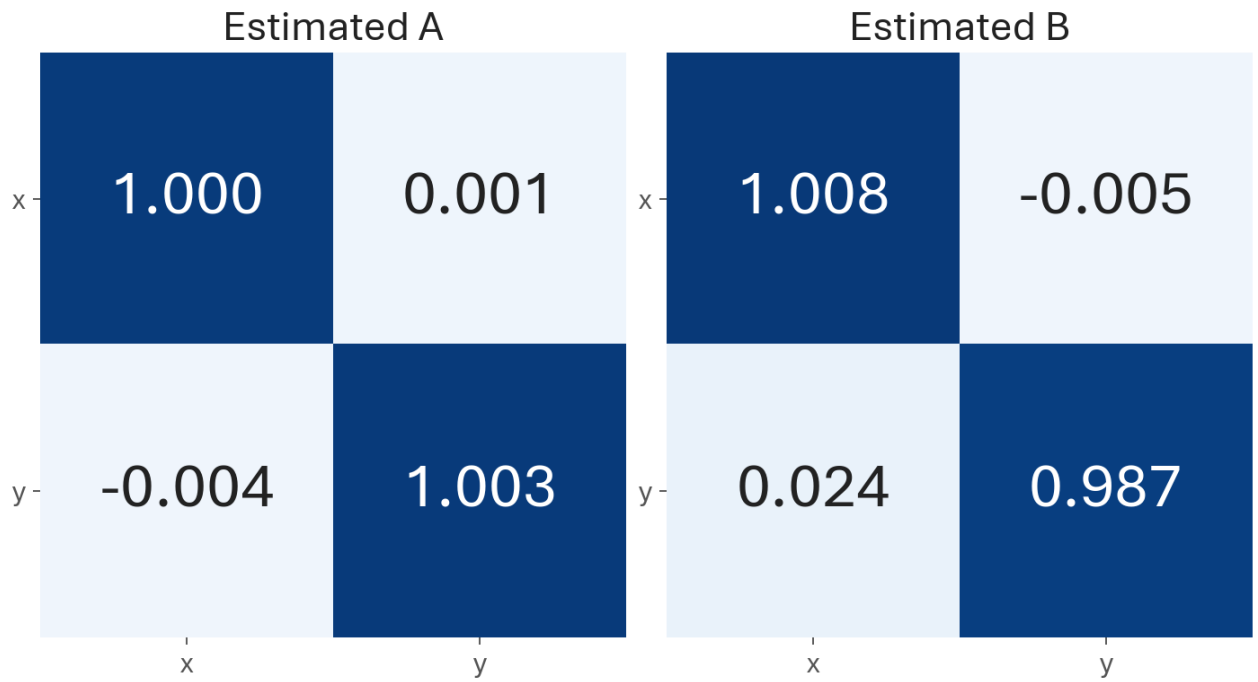
Recovering A and B

The diagonal dominates

Both fitted matrices are close to the identity matrices used to simulate the robot.

Small cross terms

An x command has little estimated effect on y, and vice versa.

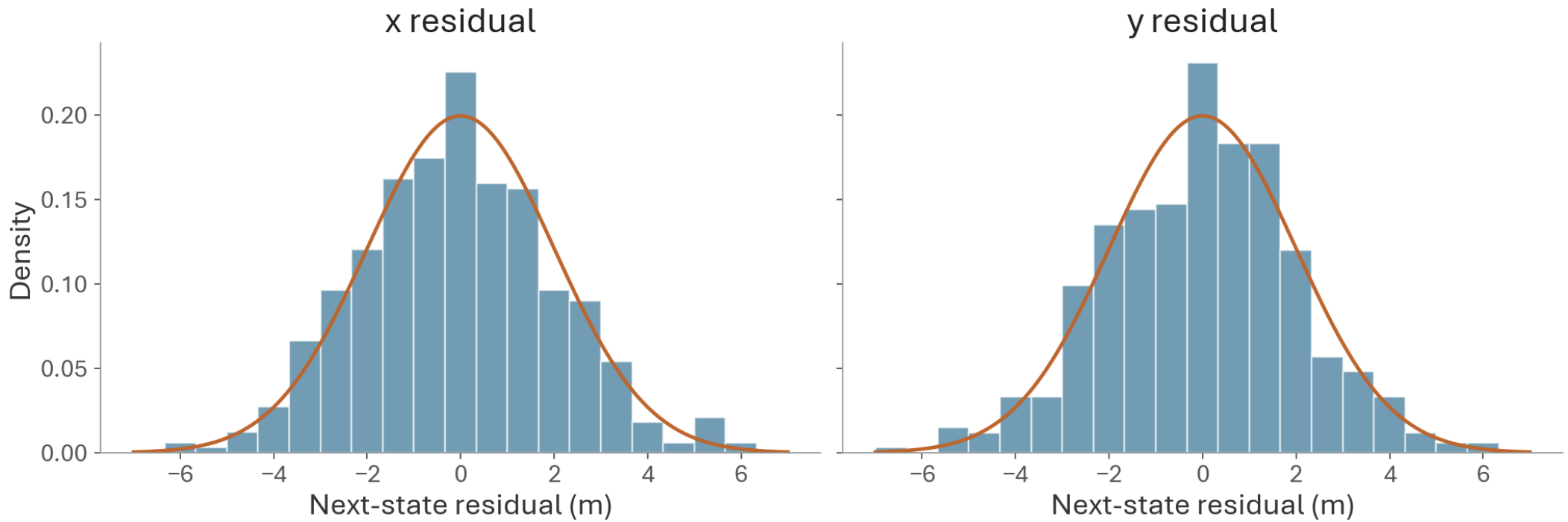



```
XU = np.vstack([X, U])   # Fit next state from state + control
```

Numbers are fitted coefficients from the new notebook-model simulation.

Process-noise estimates match the simulation

The unexplained motion has estimated $\sigma = 2.07$ m. The simulation used $\sigma = 2$ m.



Two coordinates × 500 transitions, minus eight fitted coefficients: denominator $2N - 8$.

Planning and learning: the main ideas

Planning

Value iteration propagates rewards through the grid. The policy maximizes Q.

Uncertainty

Motion noise changes expected outcomes. Beliefs also capture uncertainty about state.

Learning

Regression estimates linear models from rich data.

