KF (cont), Local Planners

Fast Robots, ECE4160/5160, MAE 4190/5190

E. Farrell Helbling, 3/10/26
Slides adapted from Prof. Kirstin Petersen

Class Action Items

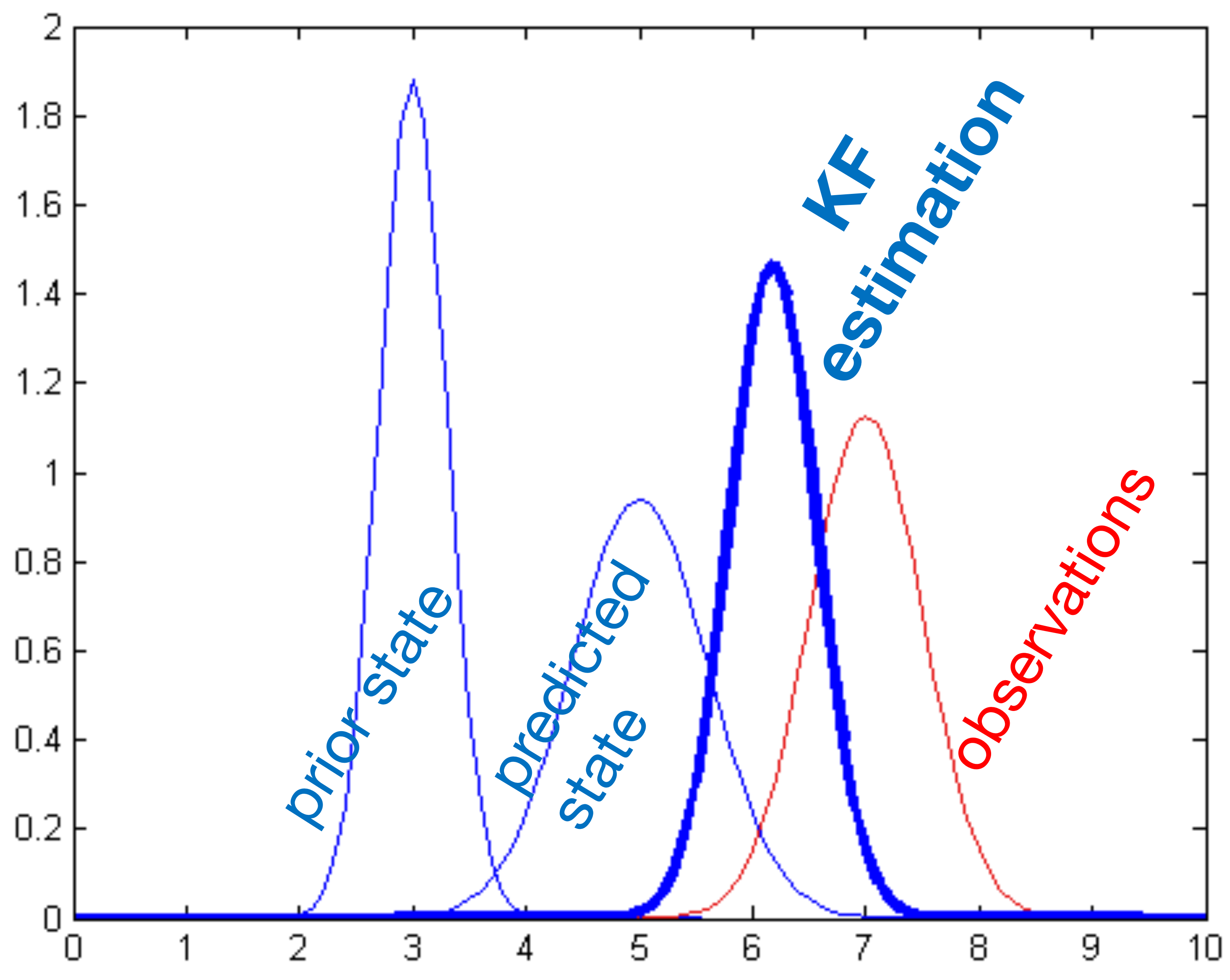
- Lab 6 begins this week, looking at orientation control of your robot.
  - Good example from last year: Evan Leong
- Please start working with a lab partner. We are entering the robot skills part of this class, it is highly likely that your robot will break. When this happens, borrow your friends robot. Get the other working in the meantime so you always have a working robot.
- Midsemester evaluations, please fill them out! I was not notified that you already got the emails. You get a point towards your final grade if you fill them out.
- Continuing with Kalman Filtering today for your Lab 7, local planners

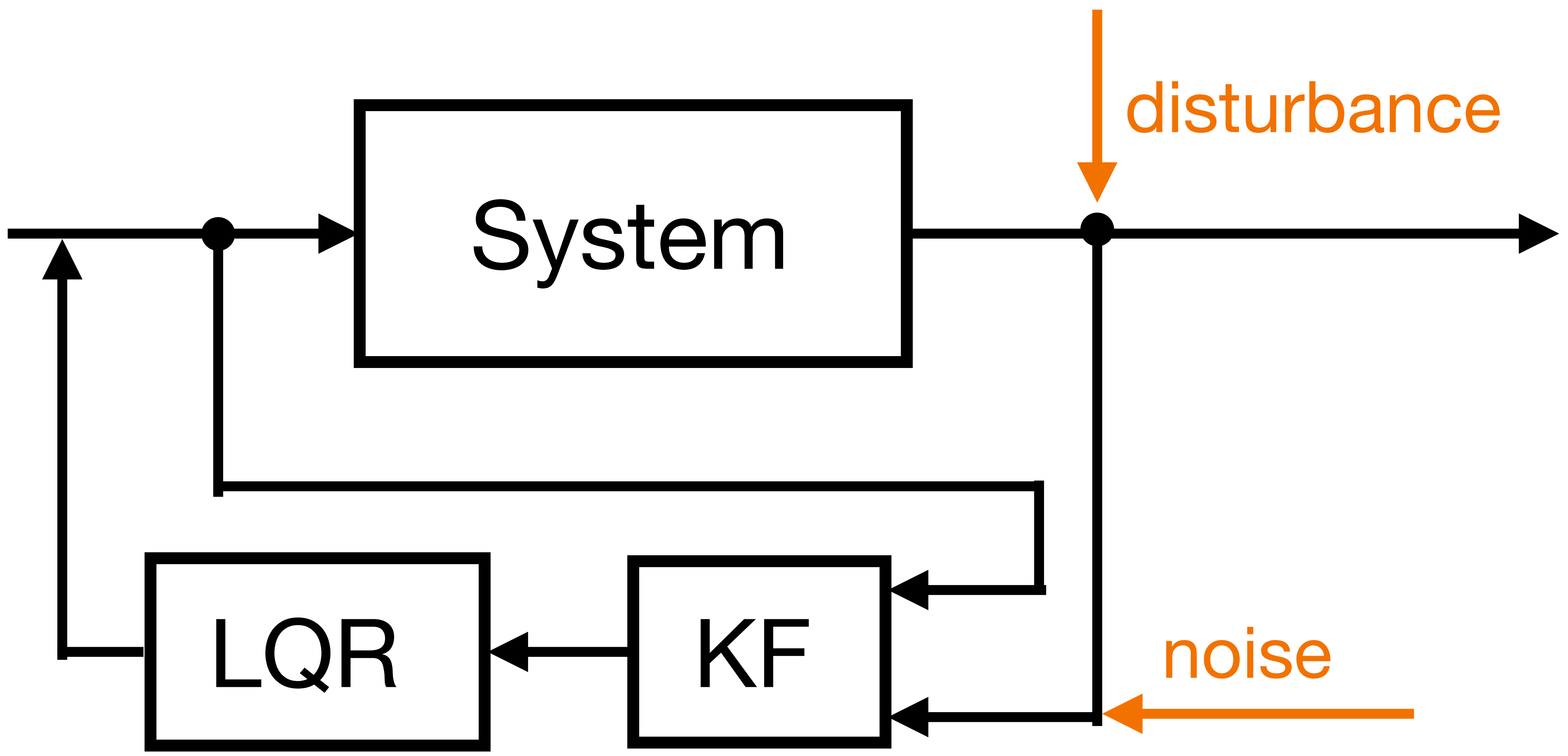
Class Late Policy

- Everyone has 2 one-week extensions that they can use towards any 2 labs throughout the semester (excluding the final lab, which will be due during Finals Week).
- If work is submitted after this one-week grace period, we will assess a late penalty of 1 point per day.
- If you have already used both extensions for prior labs and a third lab will be submitted late, we will apply the late penalty.
- We will not apply the late penalty before the two one-week extensions have been used.

Kalman Filter

- Incorporate uncertainty to get better estimates based on both inputs and observations, assuming that posterior and prior beliefs are Gaussian





Kalman Filter

Kalman Filter ($\mu(t-1)$, $\Sigma(t-1)$, $u(t)$, $z(t)$)

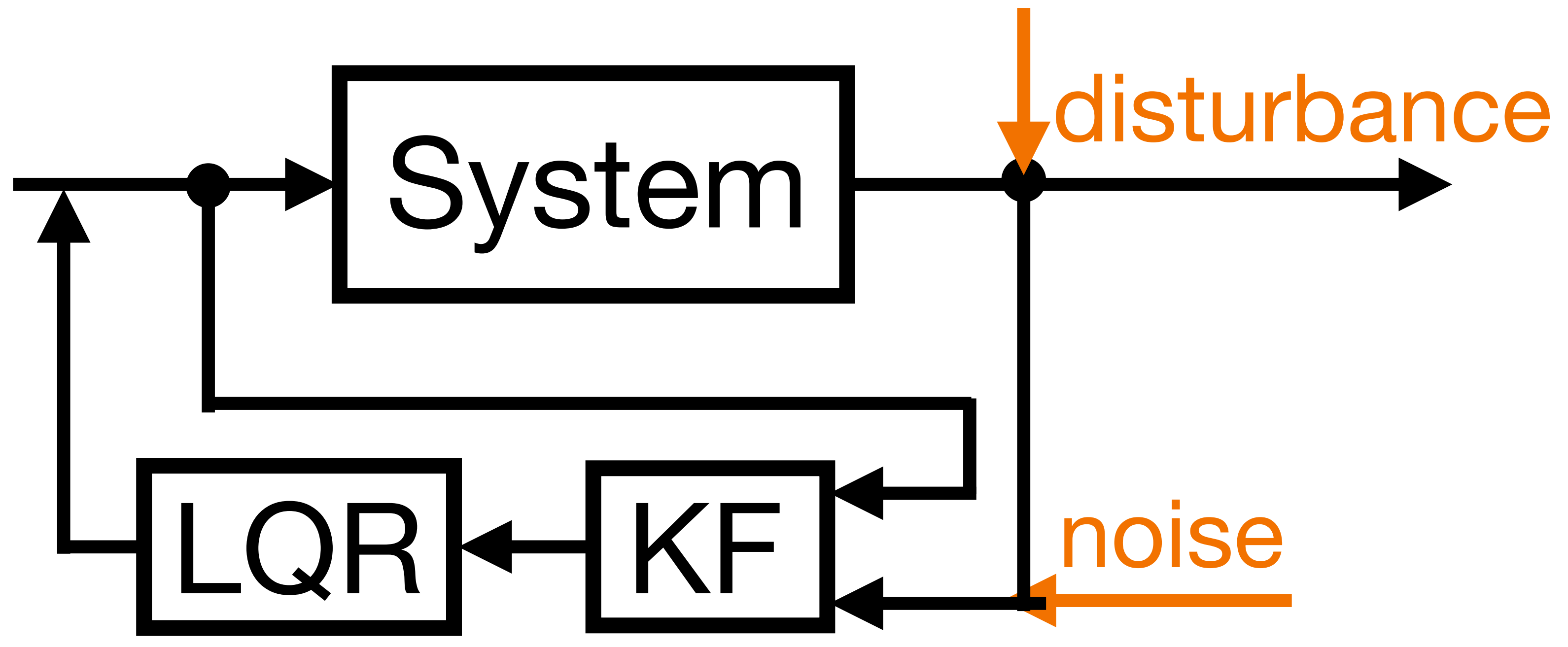
prediction

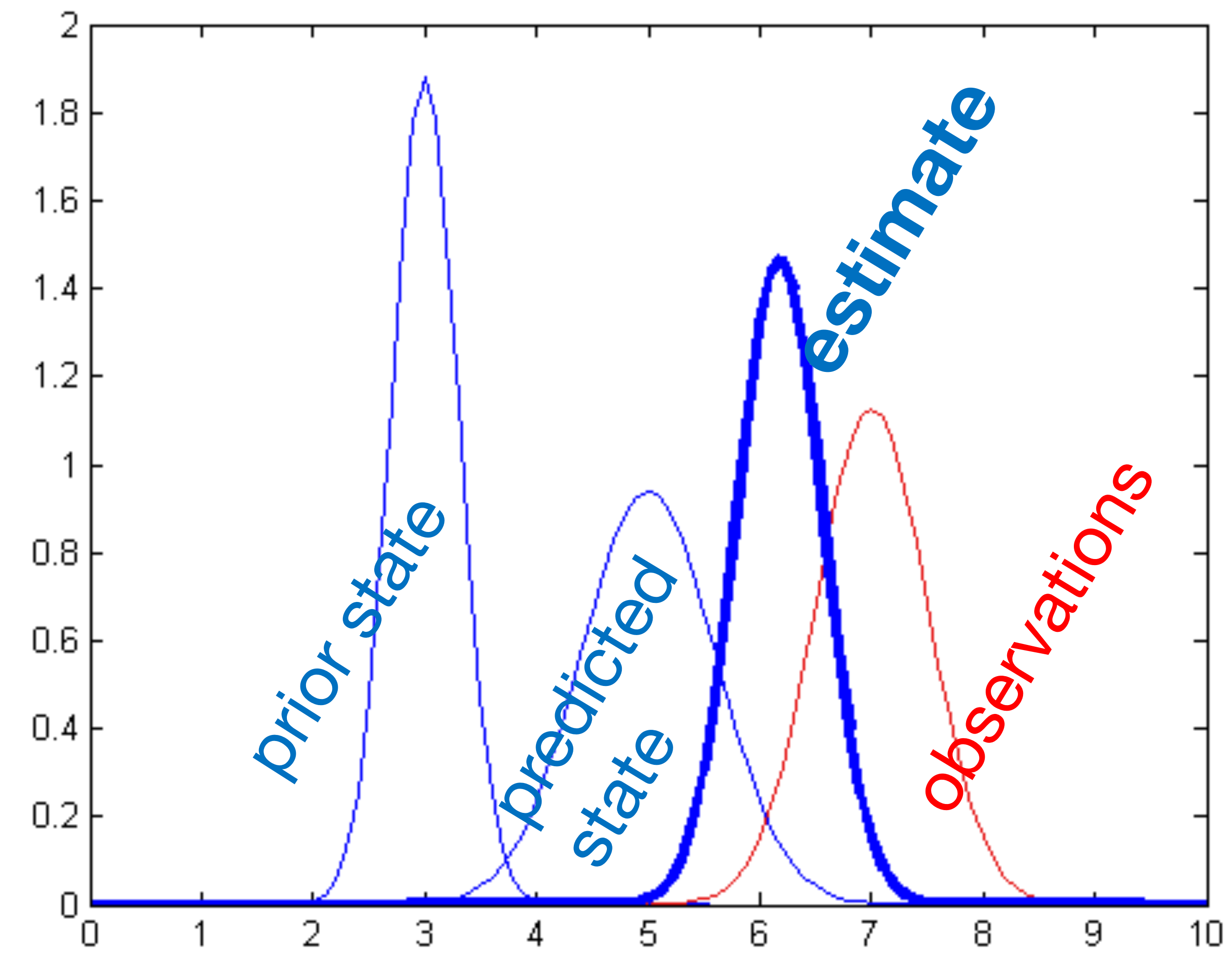
update

6. Return $\mu(t)$ and $\Sigma(t)$

State estimate: $\mu(t)$

State uncertainty: $\Sigma(t)$





Example process and measurement noise covariance matrices:

$$\Sigma_u = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}, \Sigma_z = \sigma_3^2$$

Lab 7: Kalman Filter

- Define A, B, and C matrices
  - System ID on step response

Lab 7: Kalman Filter

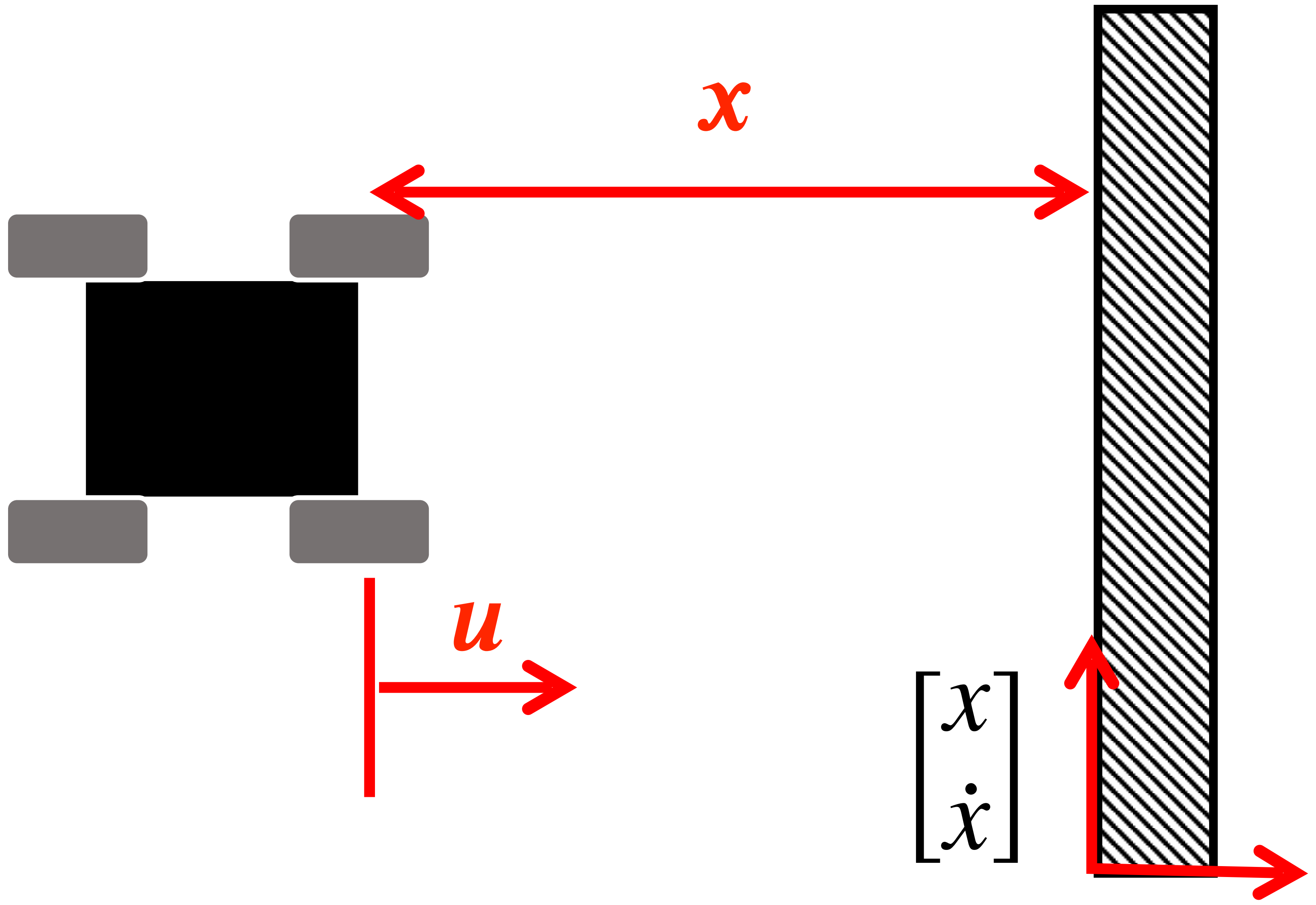
$$F = ma = m\ddot{x}$$

$$F = u - d\dot{x}$$

$$m\ddot{x} = u - d\dot{x}$$

$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

What are d and m?



State space equations

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{d}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u$$

$$C = [-1 \quad 0]$$

Lab 7: Kalman Filter

$$F = ma = m\ddot{x}$$

$$F = u - d\dot{x}$$

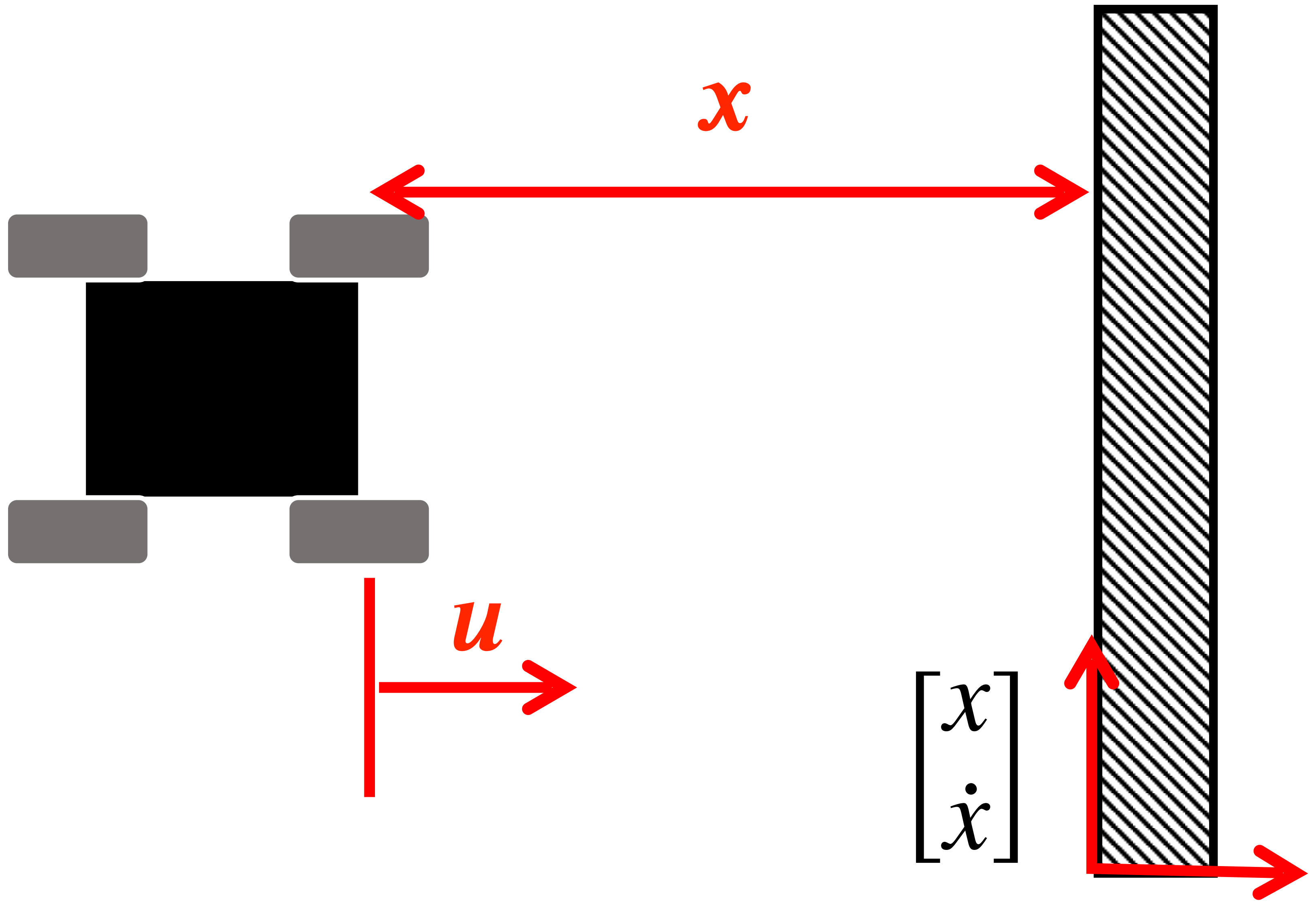
$$m\ddot{x} = u - d\dot{x}$$

$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

What are d and m?

At constant speed, we can find d:

$$0 = \frac{u}{m} - \frac{d}{m}\dot{x} \qquad d = \frac{u}{\dot{x}}$$



State space equations

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{d}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u$$

$$C = \begin{bmatrix} -1 & 0 \end{bmatrix}$$

Lab 7: Kalman Filter

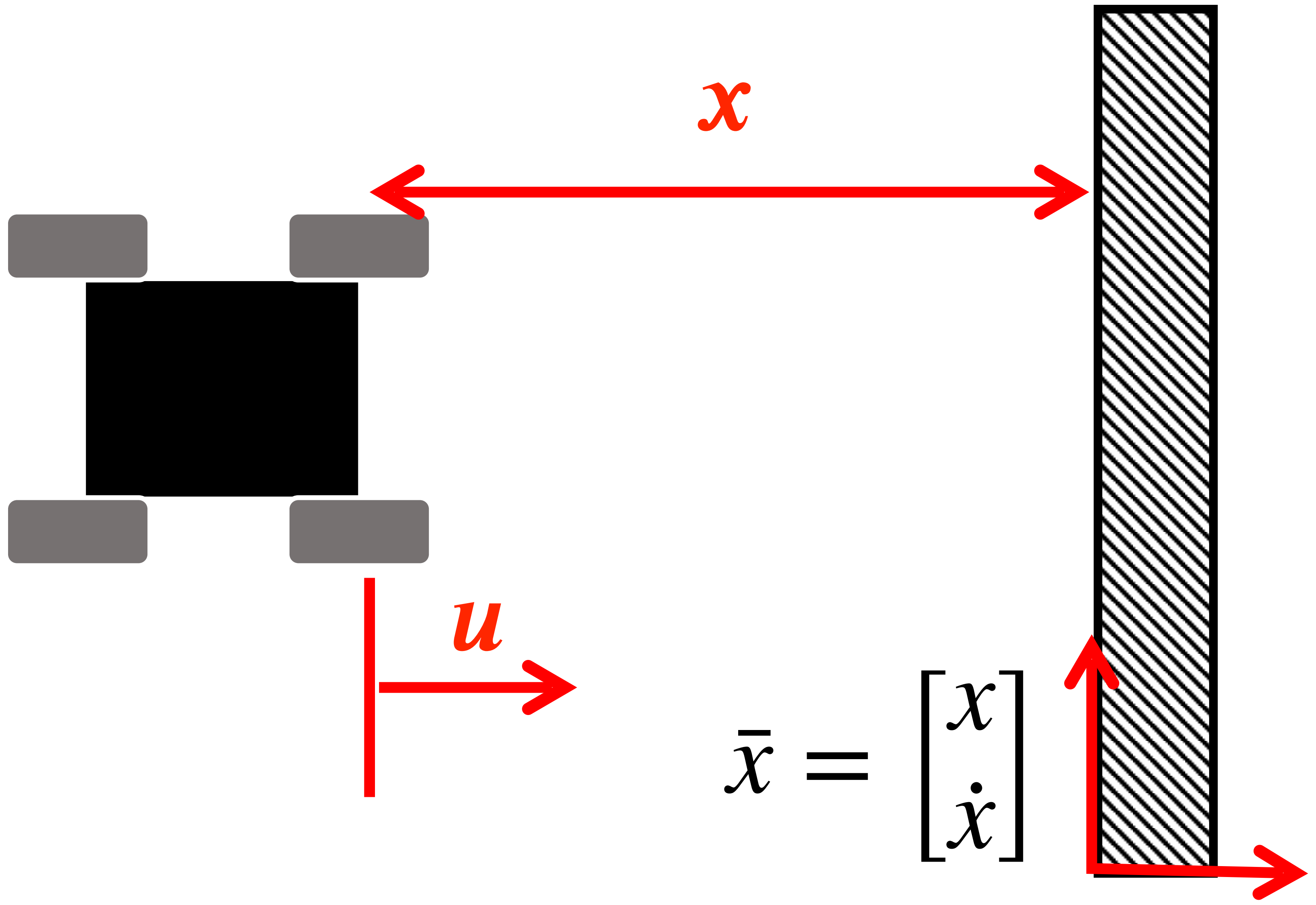
$$F = ma = m\ddot{x}$$

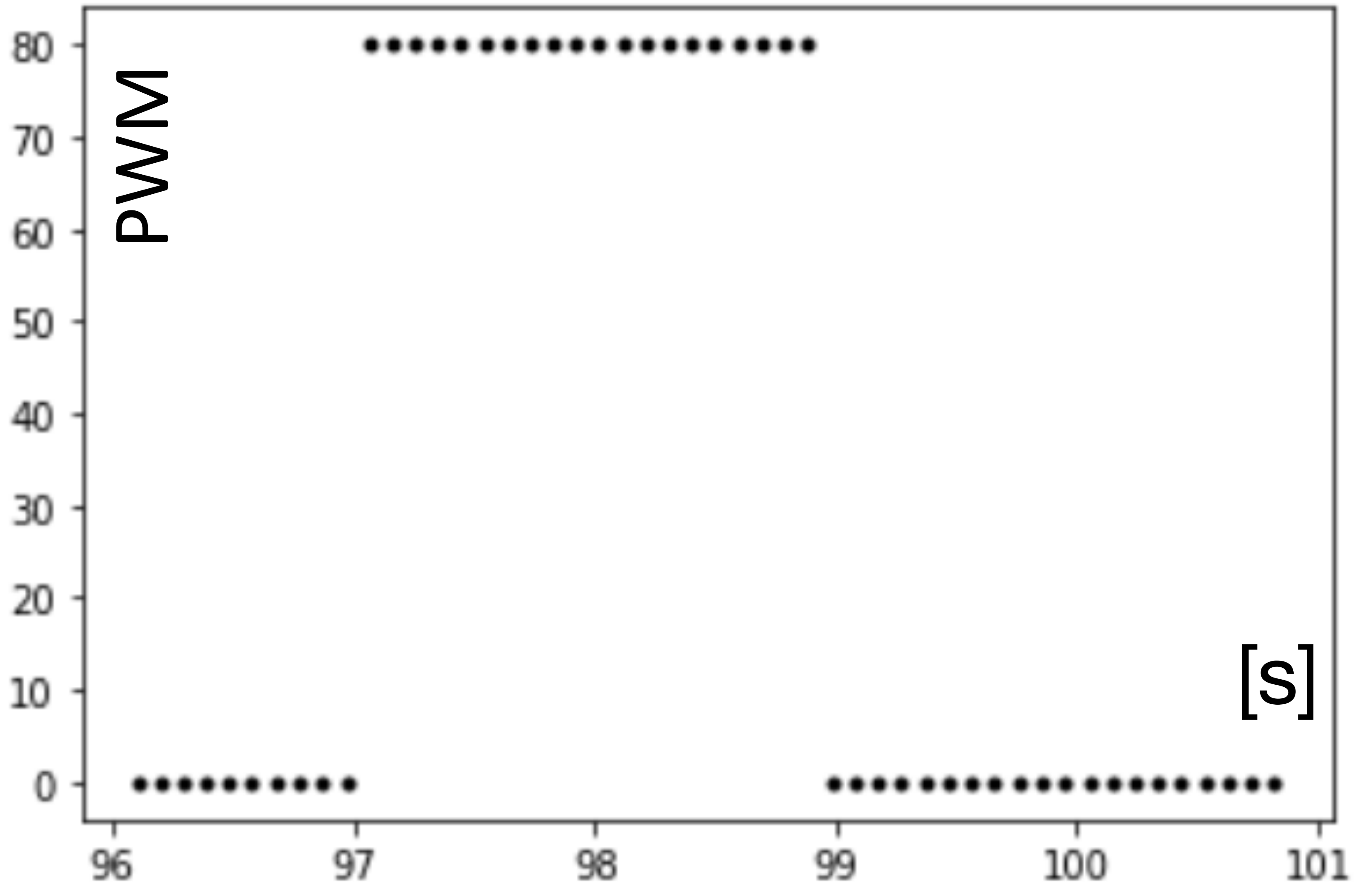
$$F = u - d\dot{x}$$

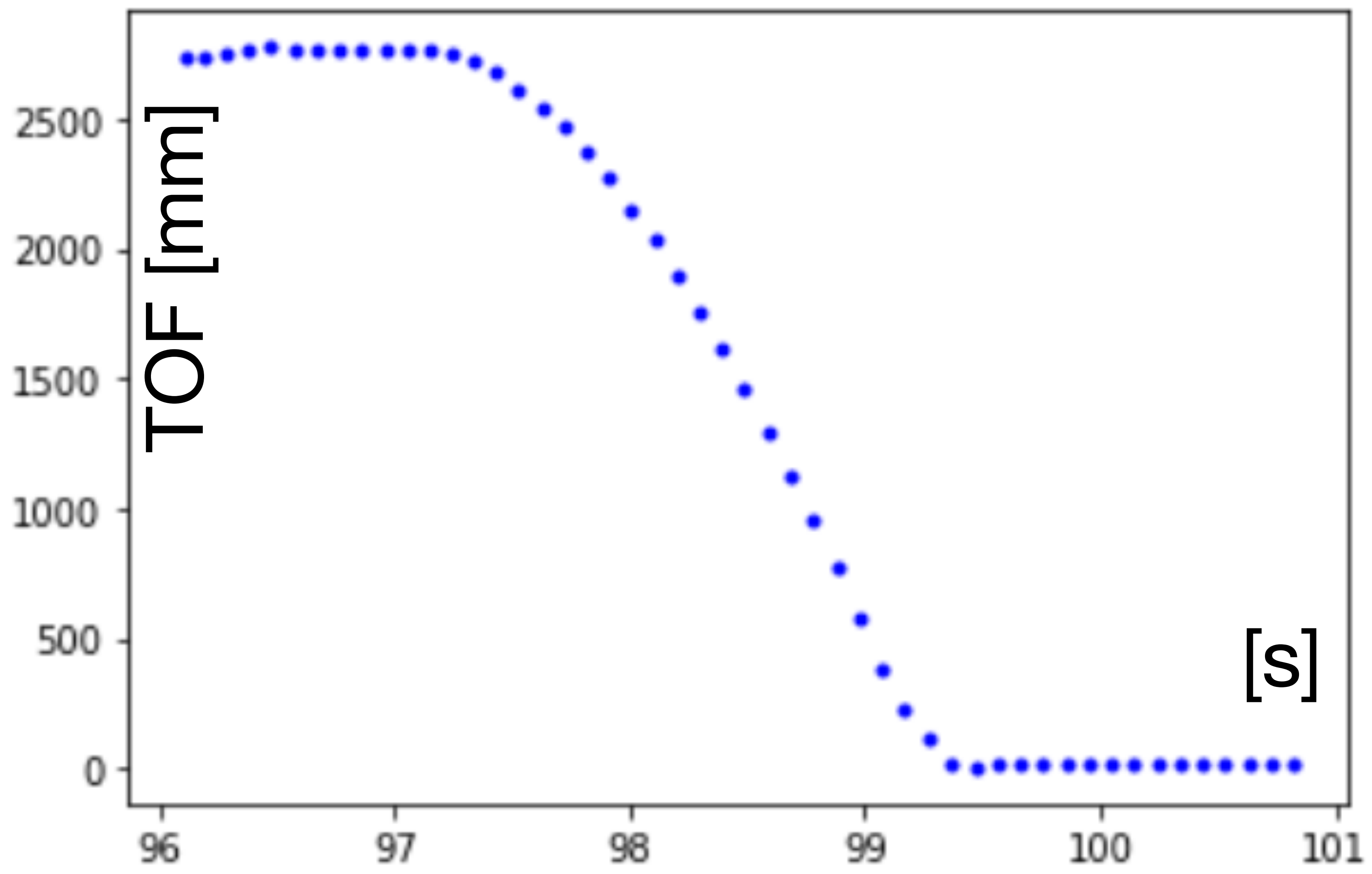
$$m\ddot{x} = u - d\dot{x}$$

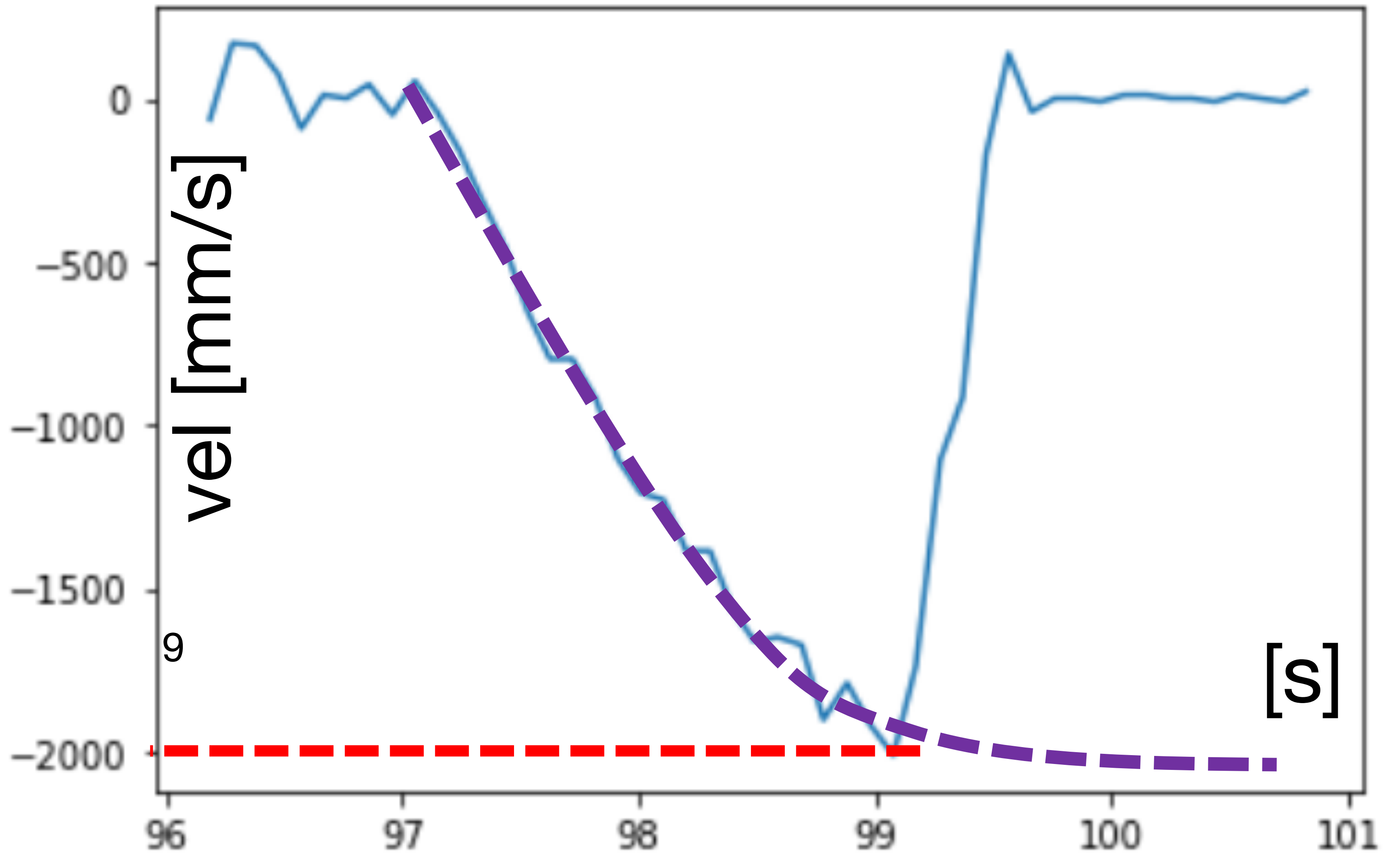
$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

What are d and m?









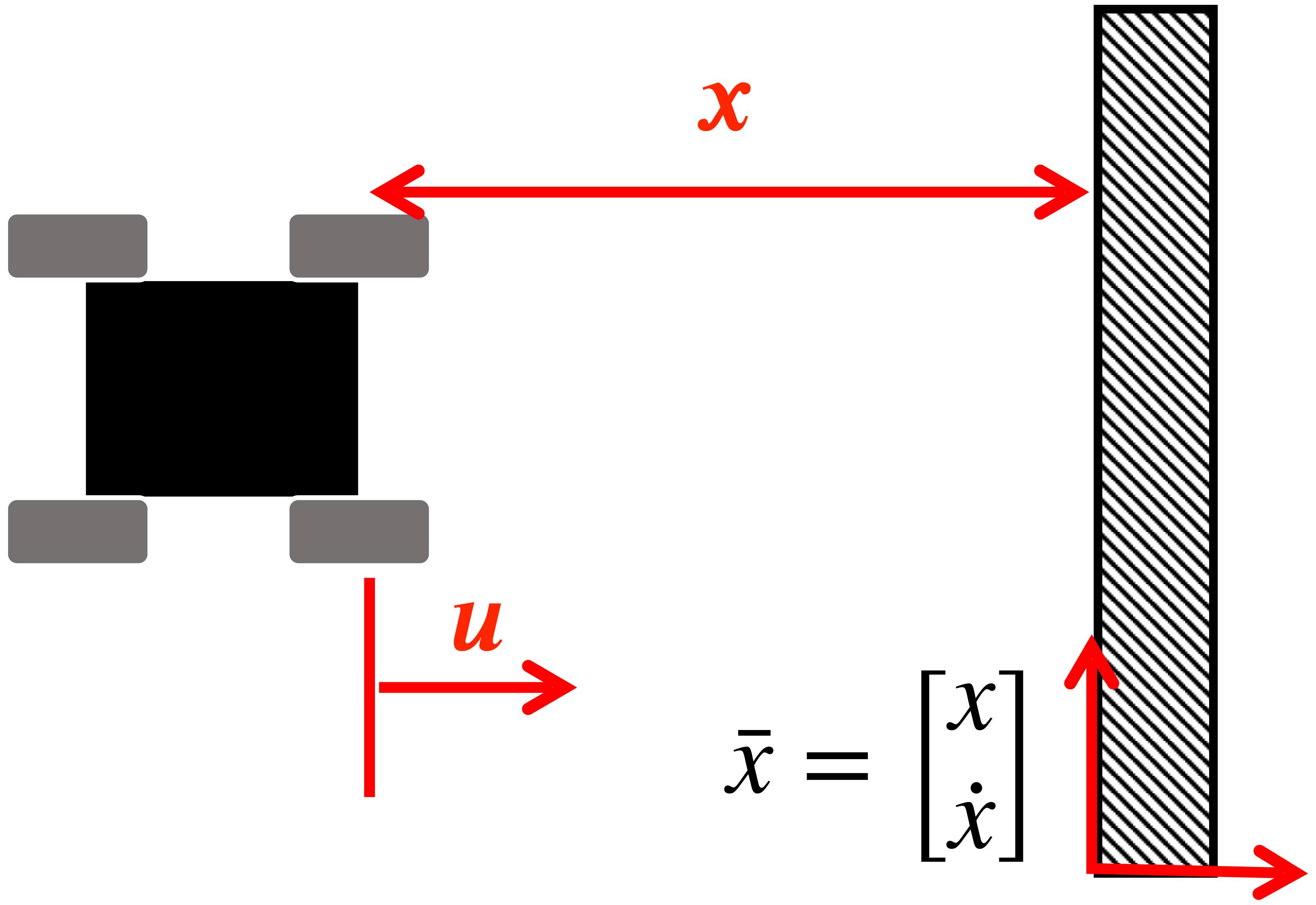
Lab 7: Kalman Filter

$$F = ma = m\ddot{x}$$

$$F = u - d\dot{x}$$

$$m\ddot{x} = u - d\dot{x}$$

$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$



$$\bar{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

What are d and m?

At constant speed, we can find d:

$$0 = \frac{u}{m} - \frac{d}{m}\dot{x} \qquad d = \frac{u}{\dot{x}}$$

(assume u=1 for now)

$$d \approx \frac{1}{2000\text{mm/s}}$$

State space equations

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{d}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u$$

$$C = \begin{bmatrix} -1 & 0 \end{bmatrix}$$

Lab 7: Kalman Filter

$$F = ma = m\ddot{x}$$

$$F = u - d\dot{x}$$

$$m\ddot{x} = u - d\dot{x}$$

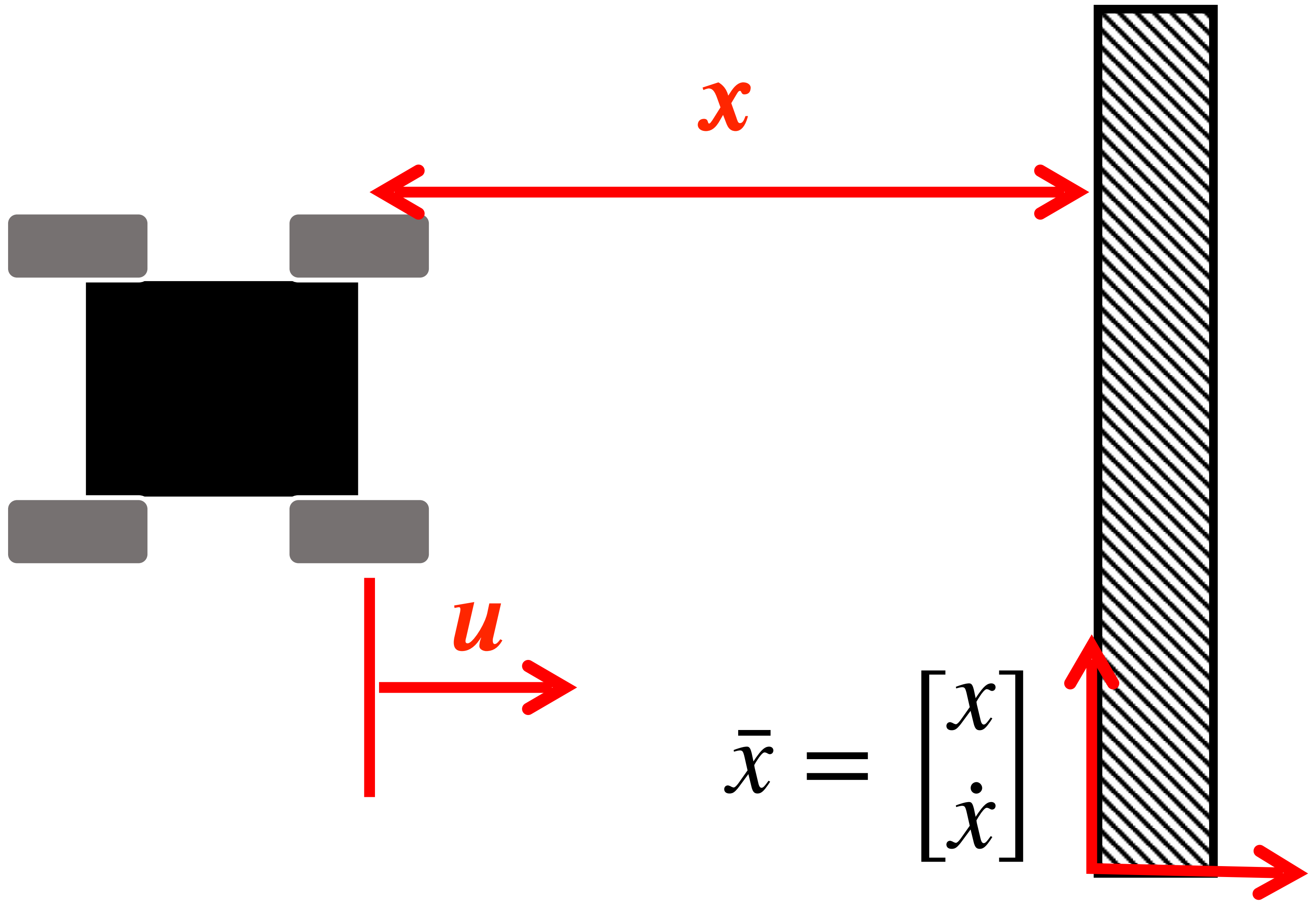
$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

1st order system:

$$\frac{dy(t)}{dt} + \frac{1}{\tau}y(t) = x(t)$$

Unit step response solution:

$$y(t) = 1 - e^{-\frac{t}{\tau}}$$



$$\bar{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

What are d and m?

Use the rise time to determine m

$$\dot{v} = \frac{u}{m} - \frac{d}{m}v$$

$$v = 1 - e^{-\frac{d}{m}t_{0.9}} \qquad \ln(1 - v) = -\frac{d}{m}t_{0.9}$$

$$m = \frac{-dt_{0.9}}{\ln(1 - 0.9)}$$

State space equations

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{d}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u$$

$$C = \begin{bmatrix} -1 & 0 \end{bmatrix}$$

Lab 7: Kalman Filter

$$F = ma = m\ddot{x}$$

$$F = u - d\dot{x}$$

$$m\ddot{x} = u - d\dot{x}$$

$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

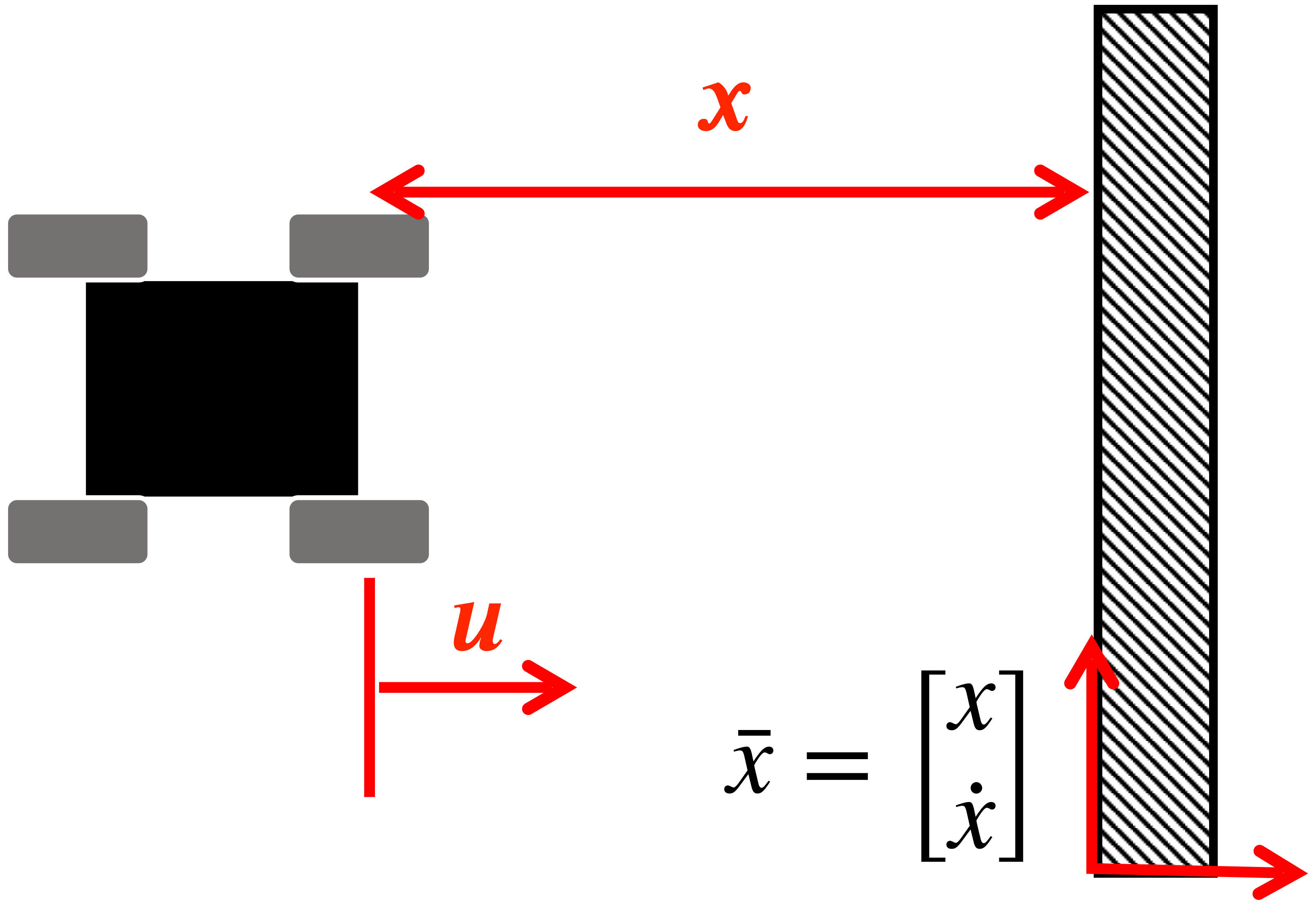
What are d and m?

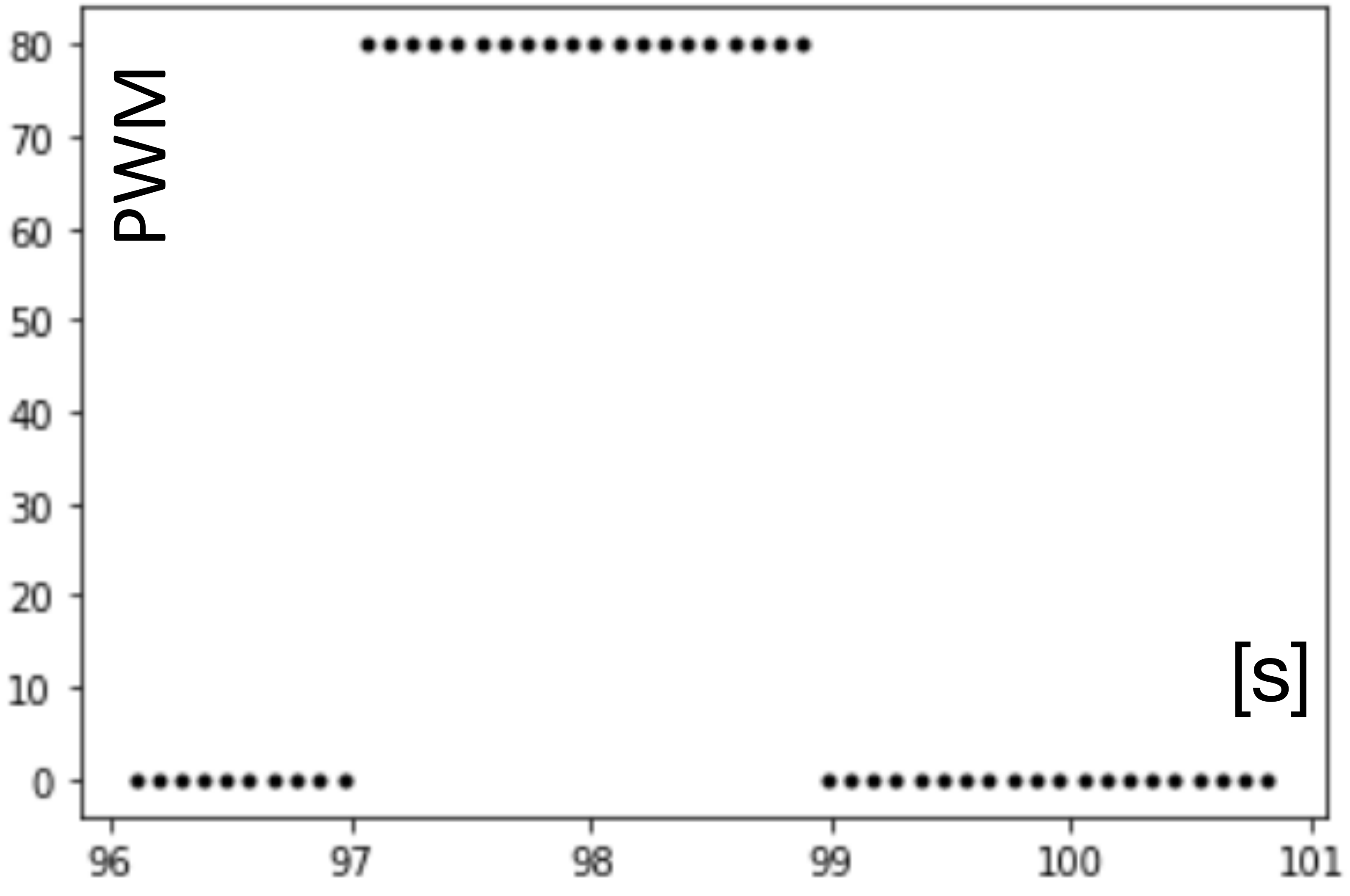
1st order system:

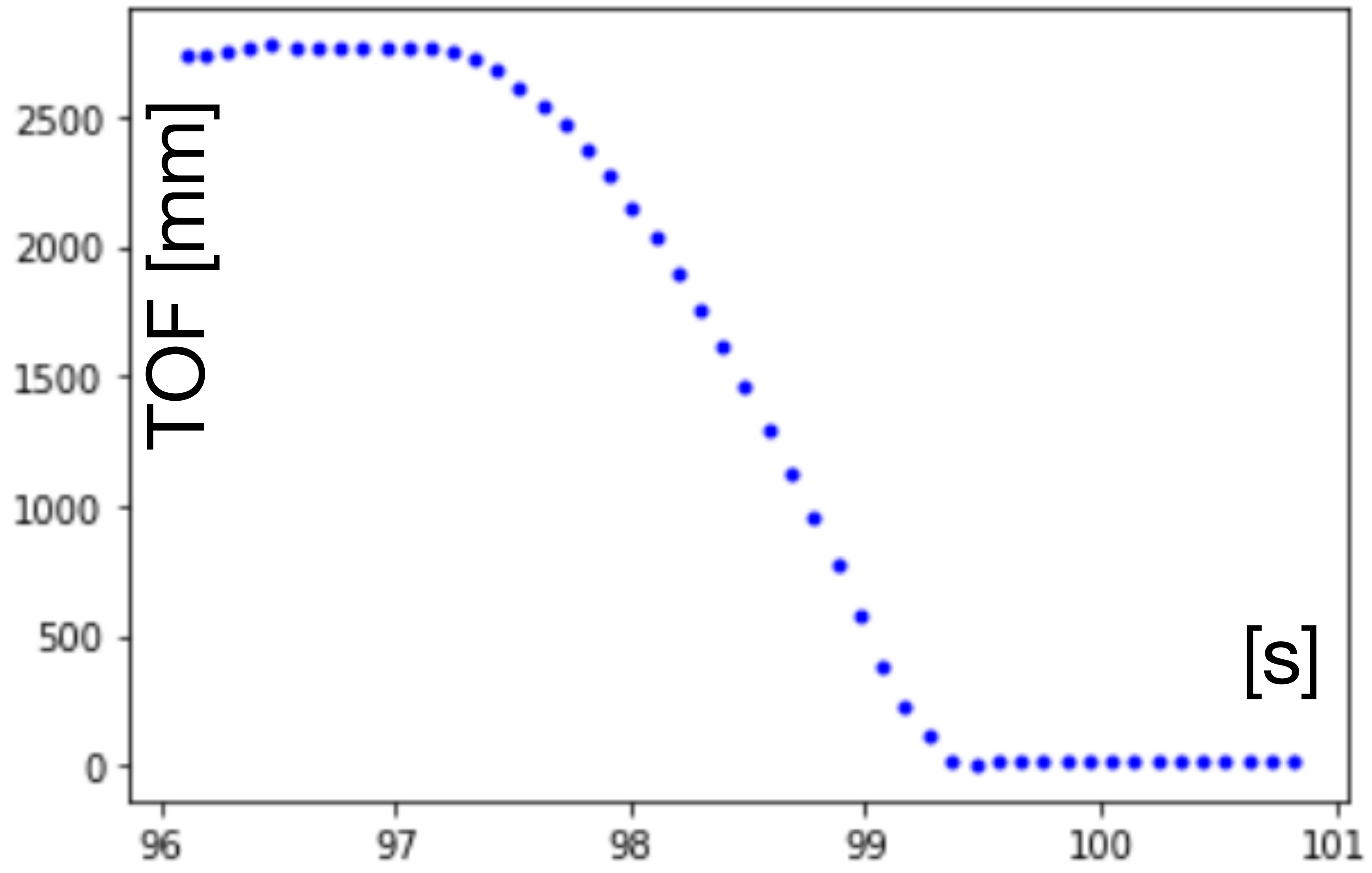
$$\frac{dy(t)}{dt} + \frac{1}{\tau}y(t) = x(t)$$

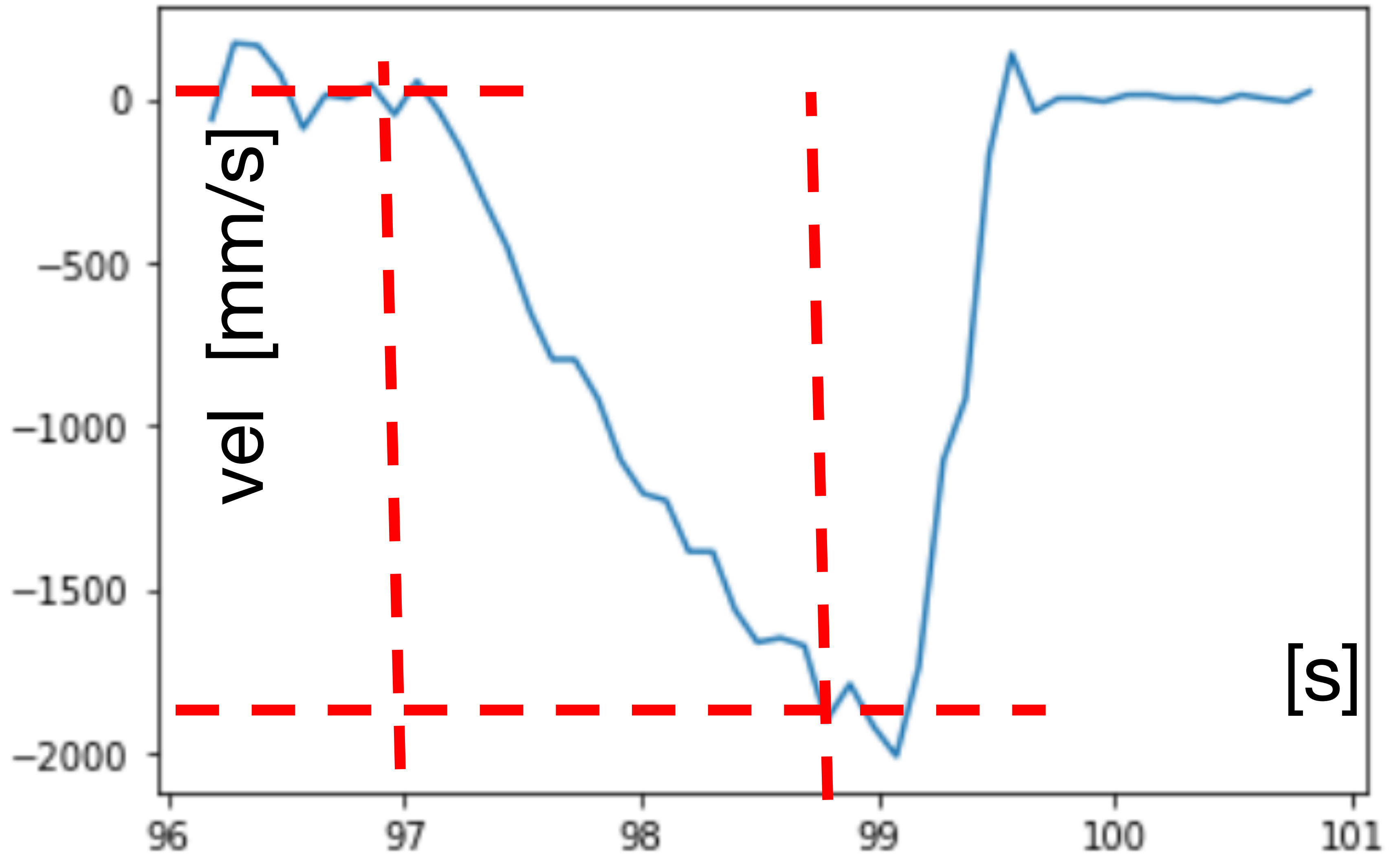
Unit step response solution:

$$y(t) = 1 - e^{-\frac{t}{\tau}}$$









Lab 7: Kalman Filter

$$F = ma = m\ddot{x}$$

$$F = u - d\dot{x}$$

$$m\ddot{x} = u - d\dot{x}$$

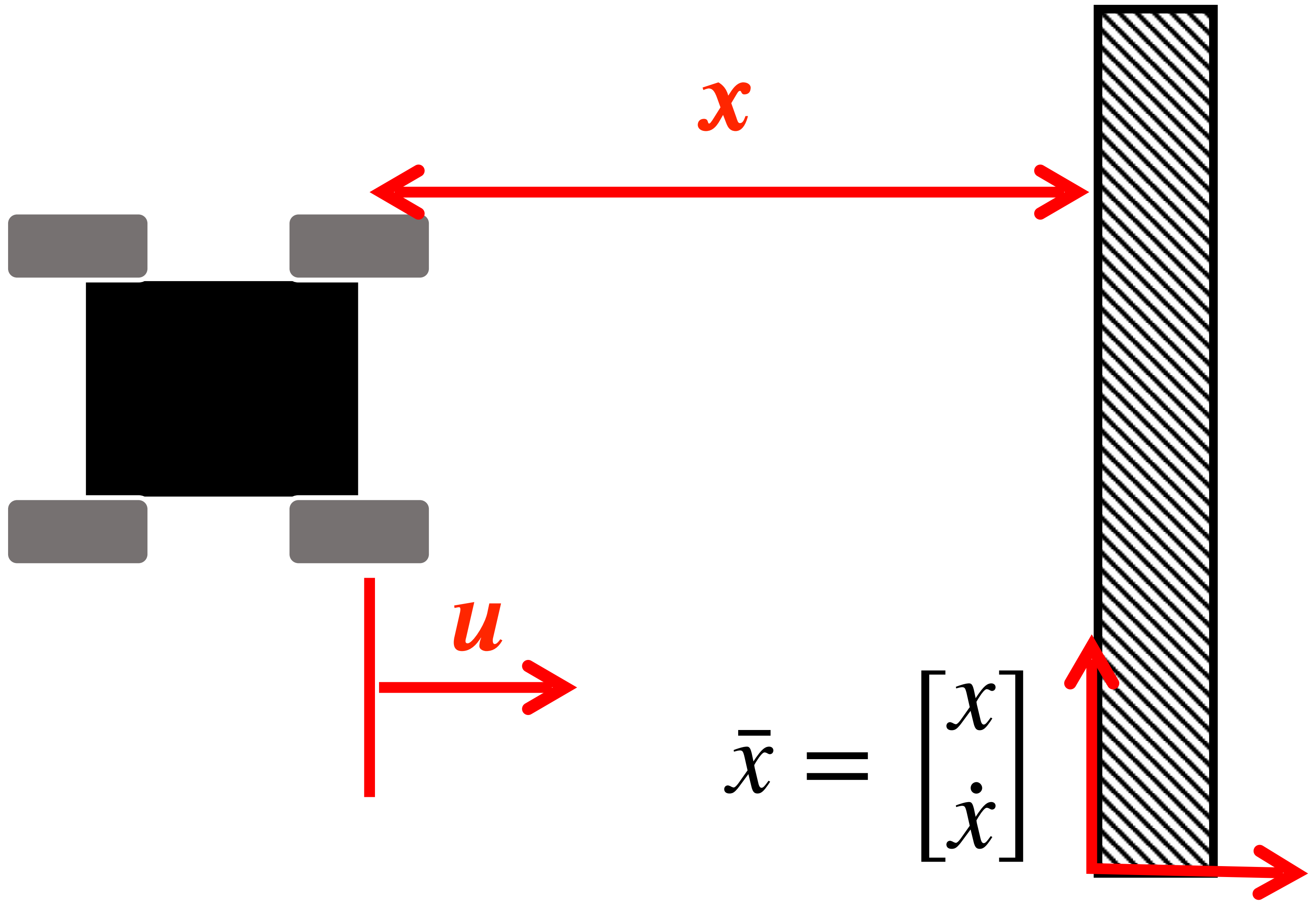
$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

1st order system:

$$\frac{dy(t)}{dt} + \frac{1}{\tau}y(t) = x(t)$$

Unit step response solution:

$$y(t) = 1 - e^{-\frac{t}{\tau}}$$



$$\bar{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

What are d and m?

Use the rise time to determine m

$$\dot{v} = \frac{u}{m} - \frac{d}{m}v$$

$$v = 1 - e^{-\frac{d}{m}t_{0.9}} \qquad \ln(1 - v) = -\frac{d}{m}t_{0.9}$$

$$m = \frac{-dt_{0.9}}{\ln(1 - 0.9)} = \frac{-0.0005 \cdot 1.9}{\ln(0.1)} = 4.1258 \cdot 10^{-4}$$

State space equations

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{d}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u$$

$$C = \begin{bmatrix} -1 & 0 \end{bmatrix}$$

Lab 7: Kalman Filter

$$F = ma = m\ddot{x}$$

$$F = u - d\dot{x}$$

$$m\ddot{x} = u - d\dot{x}$$

$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

What are d and m?

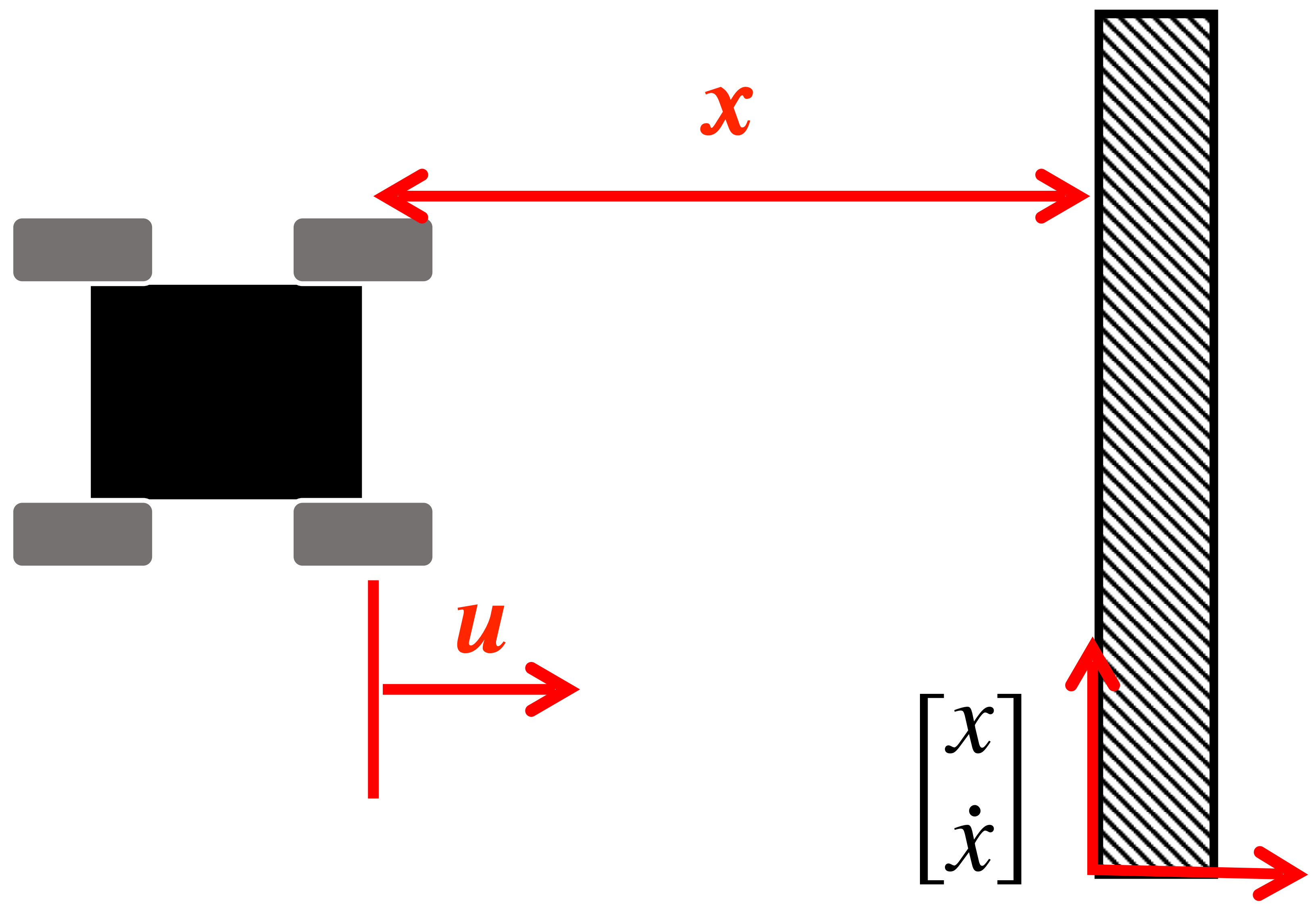
At steady state (constant speed) we can find d

$$d = \frac{u}{\dot{x}} \approx 0.0005$$

(assume u=1 for now)

We can use the rise time to find m

$$m = \frac{-dt_{0.9}}{\ln(1-0.9)} \approx 4.1258 \cdot 10^{-4}$$



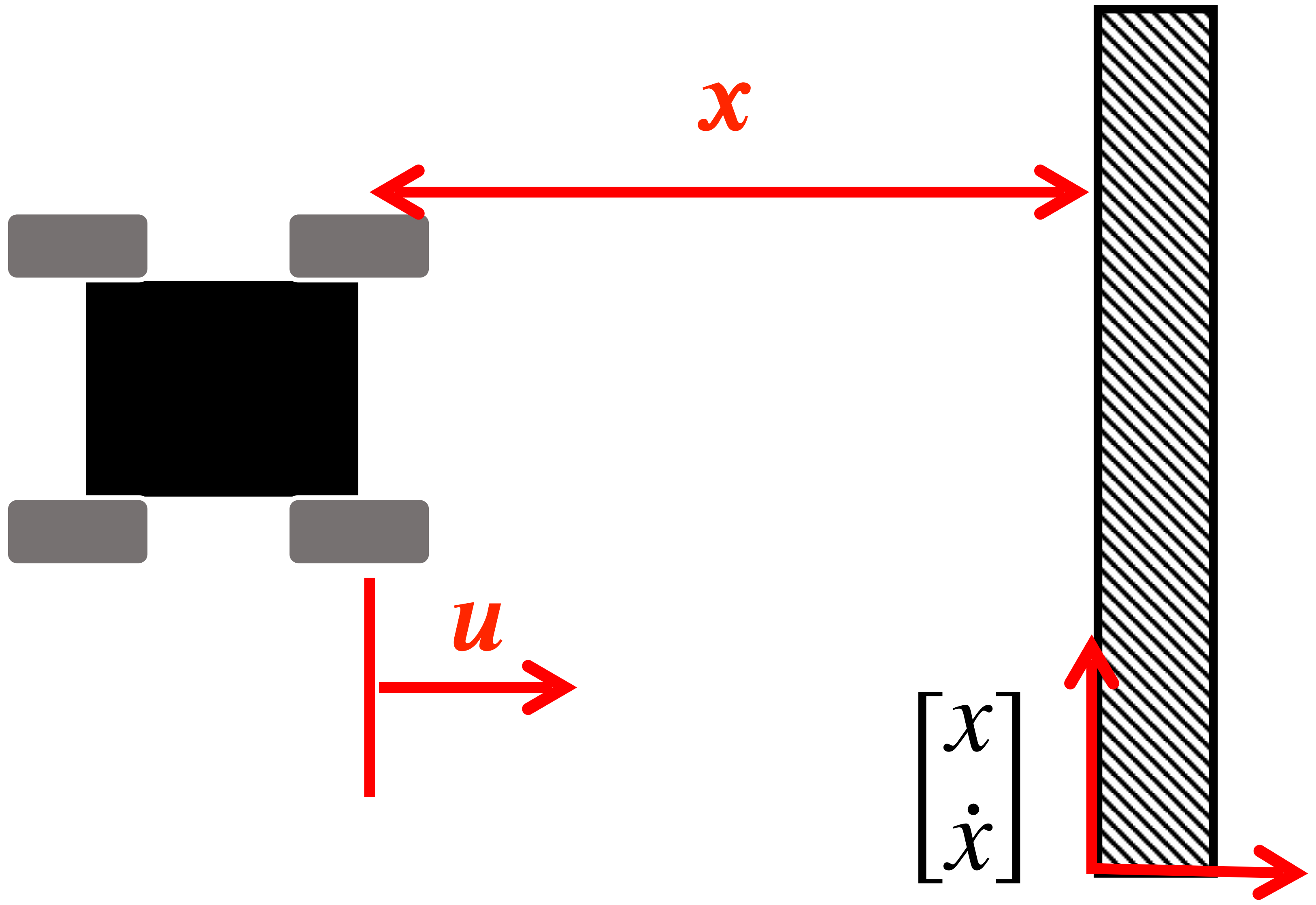
State space equations

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & -\frac{d}{m}\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{m}\end{bmatrix} u$$

$$C = [-1 \quad 0]$$

Lab 7: Kalman Filter

- We have $A,\ B,\ C$
- Discretize the A and B matrices
  - $x(n+1) = x(n) + dx$
  - $dx/dt = Ax + Bu \iff dx = dt(Ax + Bu)$
  - $x(n+1) = x(n) + dt(Ax(n) + Bu)$
  - $x(n+1) = \underbrace{(I + dt \cdot A)}_{A_d}x(n) + \underbrace{dt \cdot B}_{B_d}u$
  - dt is our sampling time (0.130s)
- Rescale from unity input to actual input



State space equations

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{d}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u$$

$$C = \begin{bmatrix} -1 & 0 \end{bmatrix}$$

Lab 7: Kalman Filter

Implement the Kalman Filter

Kalman Filter ($\mu(t-1)$, $\Sigma(t-1)$, $u(t)$, $z(t)$)

1. $\mu_p(t) = A\mu(t-1) + Bu(t)$
2. $\Sigma_p(t) = A\Sigma(t-1)A^T + \Sigma_u$
3. $K_{KF} = \Sigma_p(t)C^T(C\Sigma_p(t)C^T + \Sigma_z)^{-1}$
4. $\mu(t) = \mu_p(t) + K_{KF}(z(t) - C\mu_p(t))$
5. $\Sigma(t) = (I - K_{KF}C)\Sigma_p(t)$
6. Return $\mu(t)$ and $\Sigma(t)$

Next, determine measurement and process noise

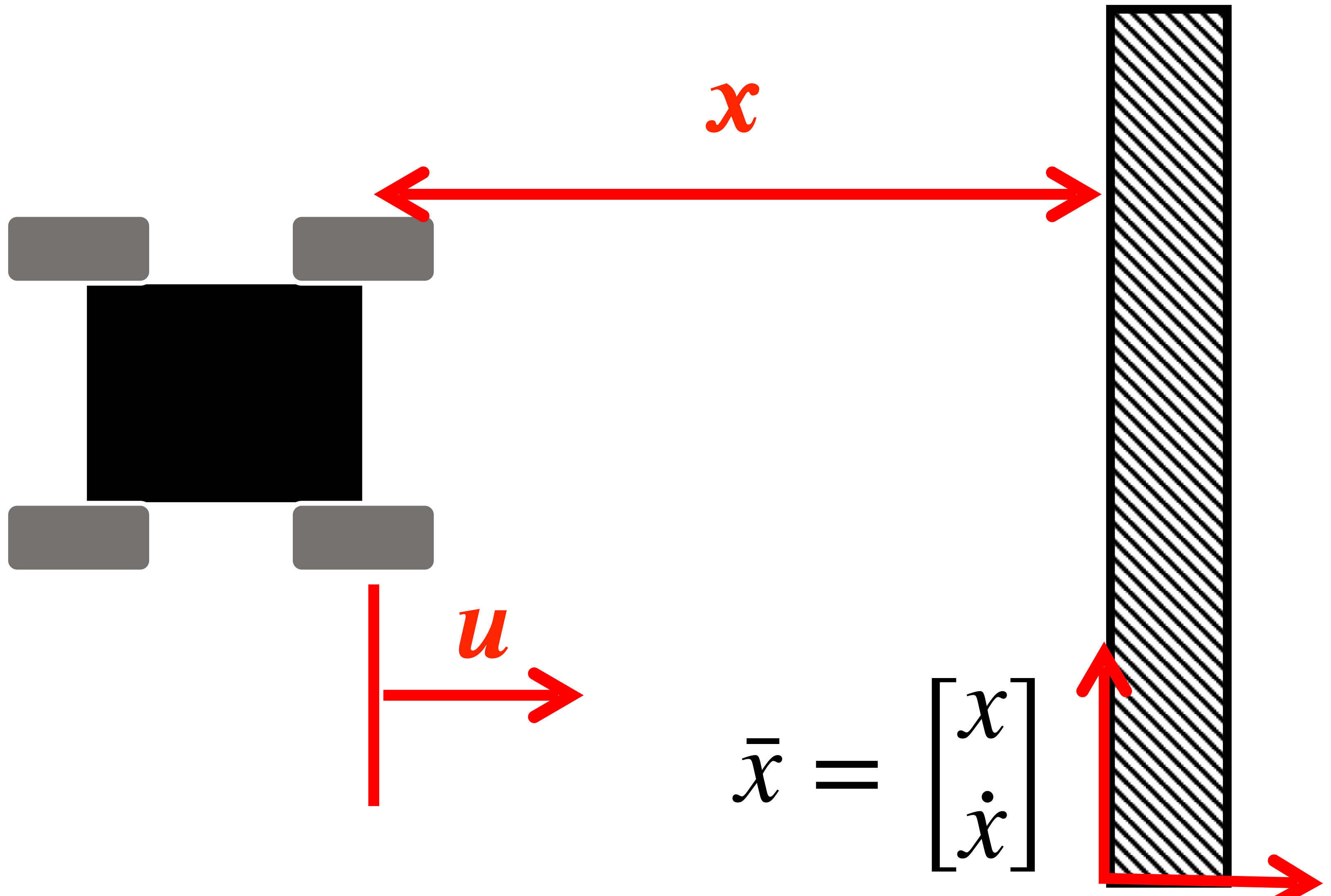
```
def kf(mu,sigma,u,y):

    mu_p = A.dot(mu) + B.dot(u)
    sigma_p = A.dot(sigma.dot(A.transpose())) + Sigma_u

    sigma_m = C.dot(sigma_p.dot(C.transpose())) + Sigma_z
    kkf_gain = sigma_p.dot(C.transpose().dot(np.linalg.inv(sigma_m)))

    y_m = y-C.dot(mu_p)
    mu = mu_p + kkf_gain.dot(y_m)
    sigma=(np.eye(2)-kkf_gain.dot(C)).dot(sigma_p)

    return mu,sigma
```

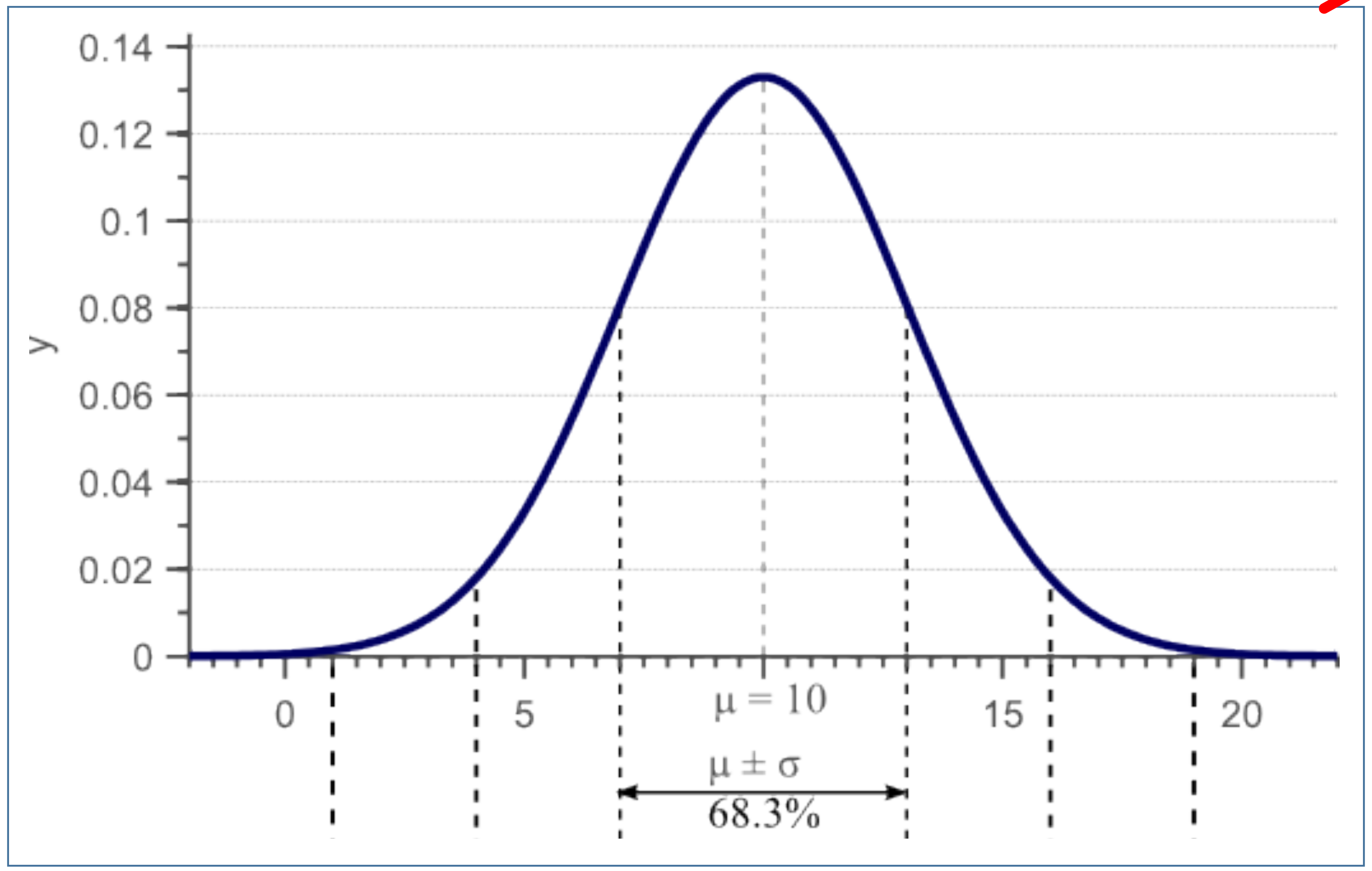
Lab 7: Kalman Filter

Implement the Kalman Filter

- Measurement noise
  - $\Sigma_z = [\sigma_3^2]$
  - $\sigma_3^2 = (20\text{mm})^2$
- Process noise (dependent on sampling rate)
  - $\Sigma_u = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$
  - Trust in modeled position:
    - Pos$_{\text{stddev}}$ after 1s: $\sqrt{10^2 \cdot \frac{1}{0.13}} = 27.7\text{mm}$
  - Trust in modeled speed:
    - Speed$_{\text{stddev}}$ after 1s: $\sqrt{10^2 \cdot \frac{1}{0.13}} = 27.7\text{mm/s}$

Sample time ~0.13s





Lab 7: Kalman Filter

Implement the Kalman Filter

Kalman Filter ($\mu(t-1)$, $\Sigma(t-1)$, $u(t)$, $z(t)$)

1. $\mu_p(t) = A\mu(t-1) + Bu(t)$
2. $\Sigma_p(t) = A\Sigma(t-1)A^T + \Sigma_u$
3. $K_{KF} = \Sigma_p(t)C^T(C\Sigma_p(t)C^T + \Sigma_z)^{-1}$
4. $\mu(t) = \mu_p(t) + K_{KF}(z(t) - C\mu_p(t))$
5. $\Sigma(t) = (I - K_{KF}C)\Sigma_p(t)$
6. Return $\mu(t)$ and $\Sigma(t)$

Finally, determine your initial state mean and covariance

$\mu(t-1)$

$\Sigma(t-1)$

```
def kf(mu,sigma,u,y):

    mu_p = A.dot(mu) + B.dot(u)
    sigma_p = A.dot(sigma.dot(A.transpose())) + Sigma_u

    sigma_m = C.dot(sigma_p.dot(C.transpose())) + Sigma_z
    kkf_gain = sigma_p.dot(C.transpose().dot(np.linalg.inv(sigma_m)))

    y_m = y-C.dot(mu_p)
    mu = mu_p + kkf_gain.dot(y_m)
    sigma=(np.eye(2)-kkf_gain.dot(C)).dot(sigma_p)

    return mu,sigma
```

Lab 7: Kalman Filter

Lab 7: Kalman Filter

- Define A, B, and C matrices
    - System ID on step response
- Sanity check
    - Run virtual kalman filter on data from Lab 5 PID
    - What is your initial state, and how confident are you in it?
    - How much trust do you put in your model versus your sensor values?
    - Experiment
        - Put less trust in the model
        - Put less trust in the sensor
        - Start with a bad initial estimate
            - Our dynamic model is a bad estimate for the static robot

Linear System Review

- Linear system: $\dot{x} = Ax$
- Solution: $x(t) = e^{At}x(0)$
- Eigenvectors: $T = \begin{bmatrix} \xi_1 & \xi_2 & \dots & \xi_n \end{bmatrix}$
- Eigenvalues: $D = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}$

```
>>[T,D] = eig(A)
```

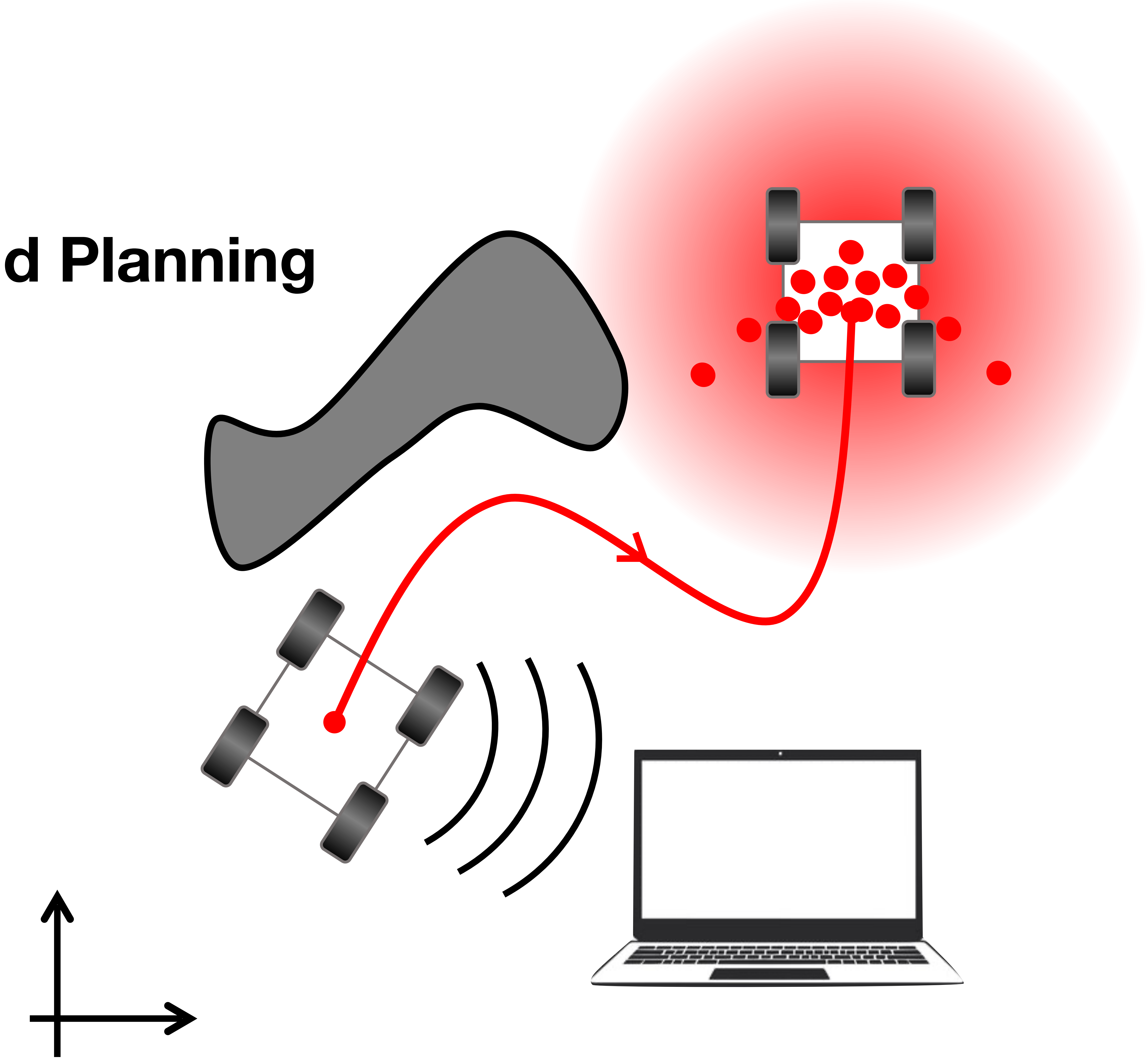
- Linear Transform: $AT = TD$
- Solution: $e^{At} = e^{TDT^{-1}t}$
- Mapping from x to z to x: $x(t) = Te^{Dt}T^{-1}x(0)$
- Stability in continuous time: $\lambda = a + ib$, stable iff $a < 0$
- Discrete time: $x(k+1) = \tilde{A}x(k)$, where $\tilde{A} = e^{A\Delta t}$
- Stability in discrete time: $\tilde{\lambda}^n = R^n e^{in\theta}$, stable iff $R < 1$
- Nonlinear systems: $\dot{x} = f(x)$
- Linearization: $\left.\frac{Df}{Dx}\right|_{\bar{x}}$
- Controllability: $\dot{x} = (A - BK)x$ `>>rank(ctrb(A,B))`
- Reachability
- Controllability Gramian
- Pole Placement `>>place(A,B,poles)`
- Optimal Control (LQR) `>>LQR(A,B,Q,R)`
- Optimal Observer (KF): sensor/model noise

What we've covered so far…

- Configuration space and transformations
- Data types
- Sensors
- Actuators/Motors
- Wiring/EMI
- Control
  - State space models
  - PID/LQR control
  - Observers
- Deterministic vs. Probabilistic Robots
  - Bayes Theorem

Next up….

Navigation and Planning



Navigation and Planning

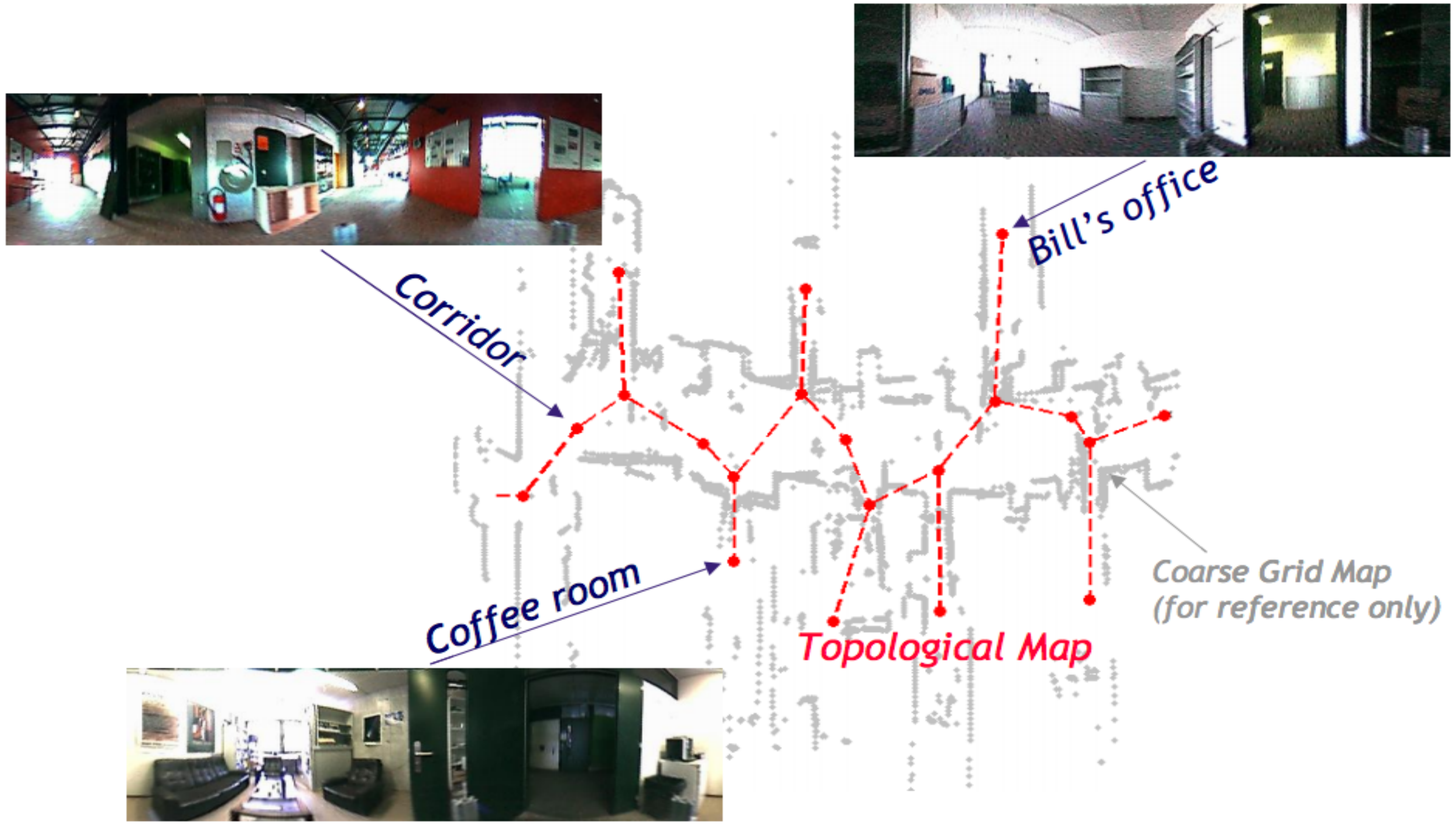
Slides adapted from Vivek Thangavelu

Navigation

- **Problem**: Find the path in the workspace from an initial location to a goal location, while avoiding collisions
- How do you get to your goal?
  - Can you see your goal?
  - Do you have a map?
  - Are obstacles unknown or dynamic?
  - Does it matter how fast you get there?
  - Does it matter how smooth the path is?
  - How much compute power do you have?
  - How precise and accurate is your motion control?
  - What sensors do you have available?
  - etc.

Navigation

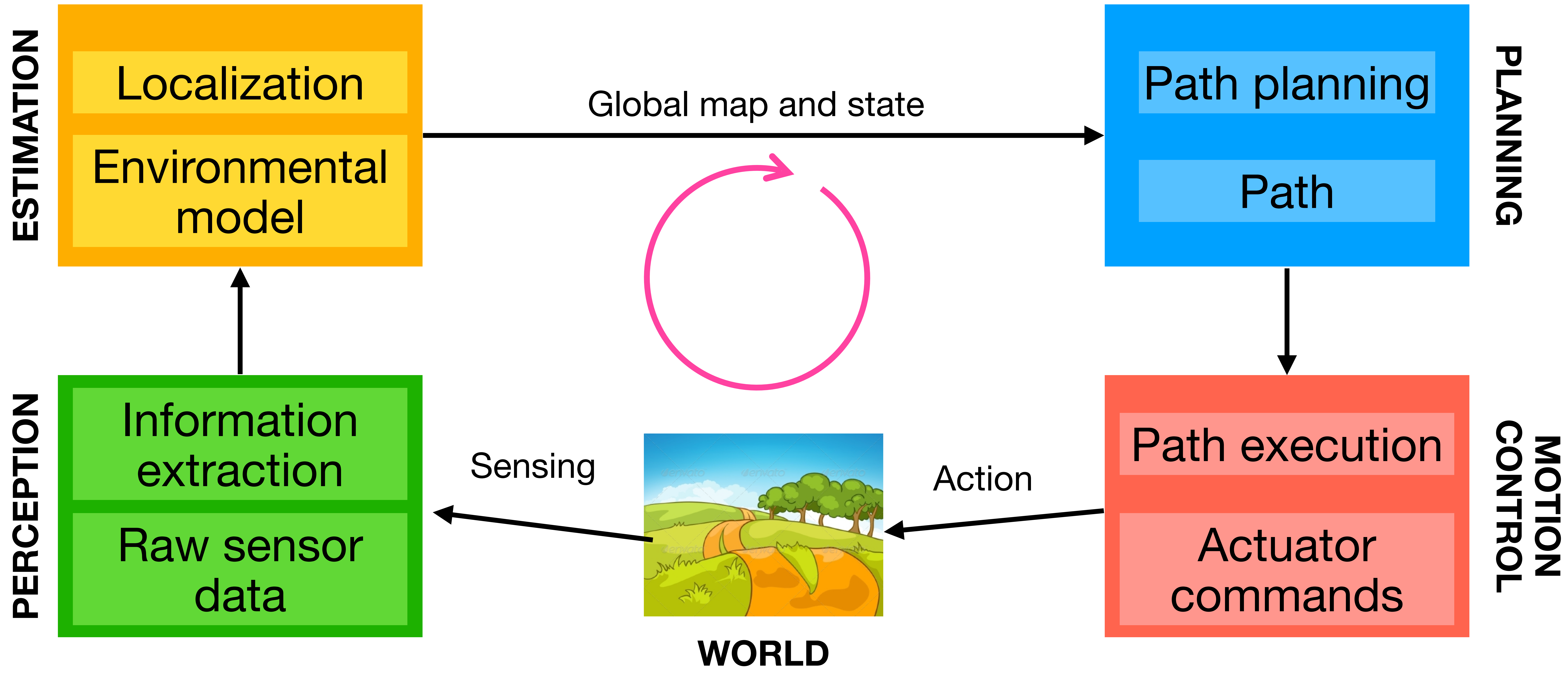
- **Problem**: Find the path in the workspace from an initial location to a goal location, while avoiding collisions
- **Assumption:** A good map for navigation exists
- **Global navigation**



- **Local navigation**

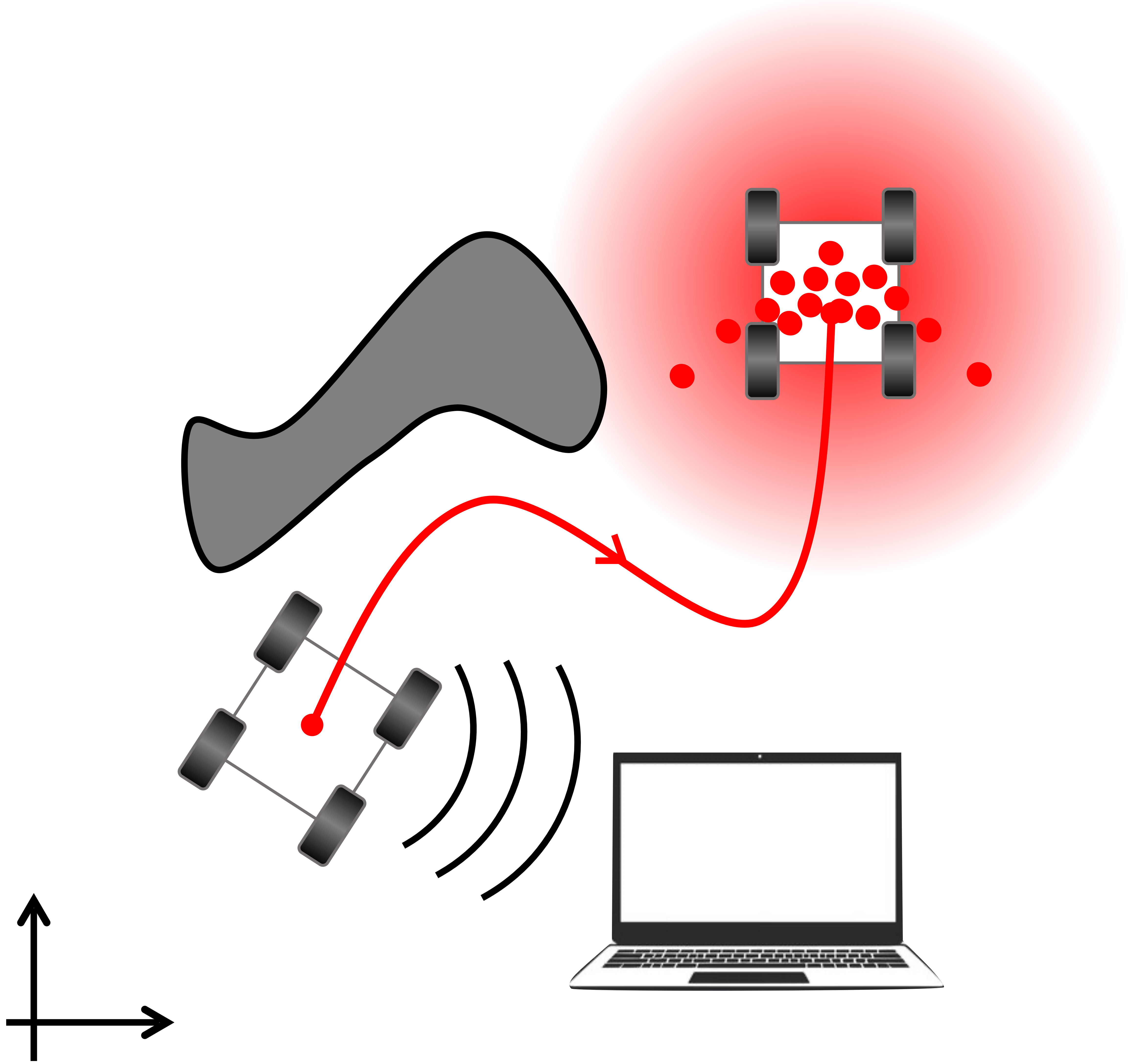
Navigation

- **Break the problem down:** localization, map building, path planning



Next module on navigation

- Local planners
- Global localization and planning
  - Map representations
    - Continuous
    - Discrete
    - Topological
  - Maps as graphs
    - Graph search algorithms
      - Breadth first search
      - Depth first search
      - Dijkstras
      - A*



Local Planners

Local path planning/ obstacle avoidance

- Use goal position, recent sensor readings, and relative position of robot to goal
  - Can be based on a local map
  - Often implemented as a separate task
  - Runs at a much faster rate than the global planner
- 3 examples:
  - BUG algorithms
  - Vector Field Histogram (VFH)
  - Dynamic Window Approach (DWA)

Wagner, ITS 2015

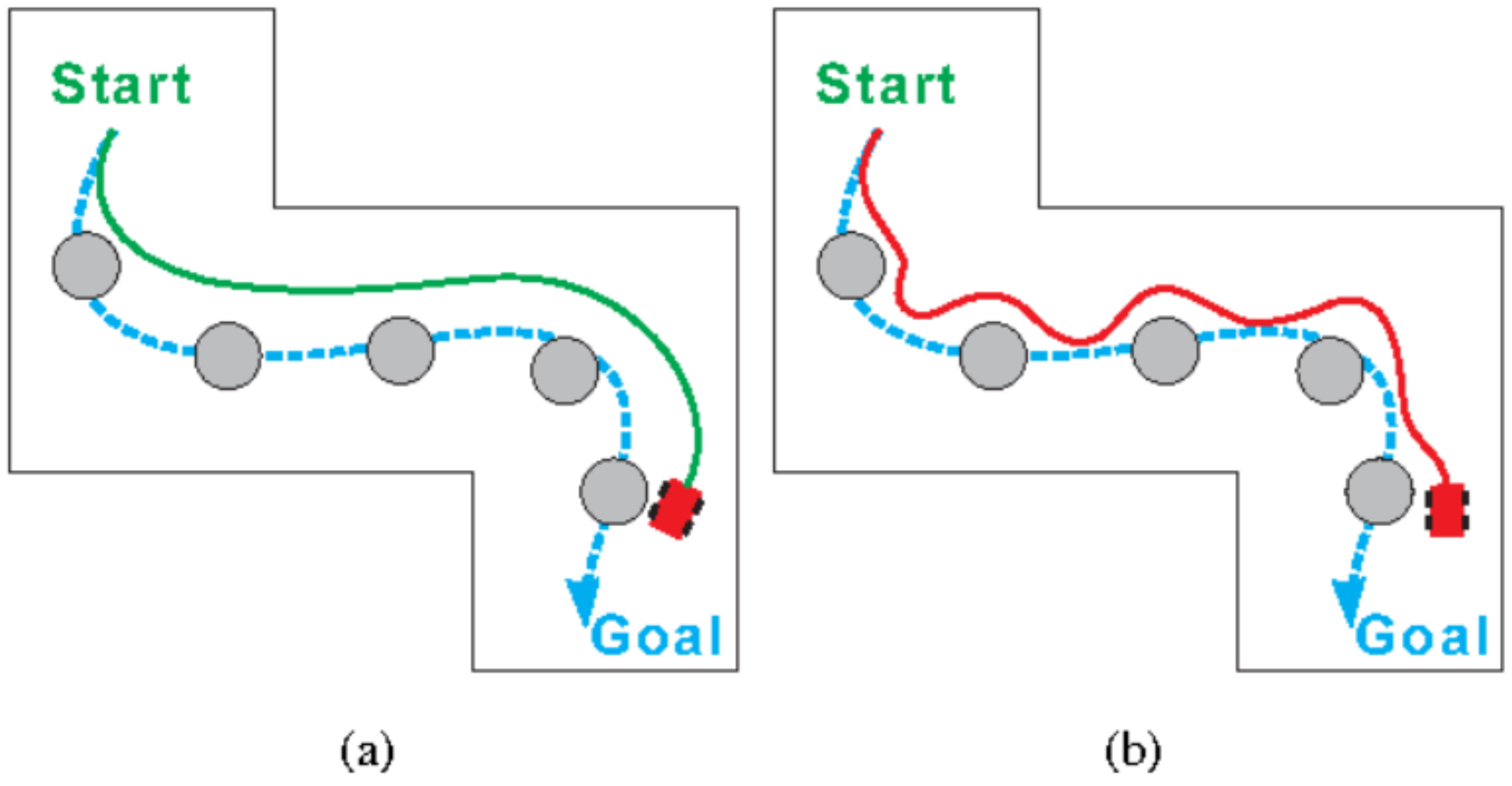
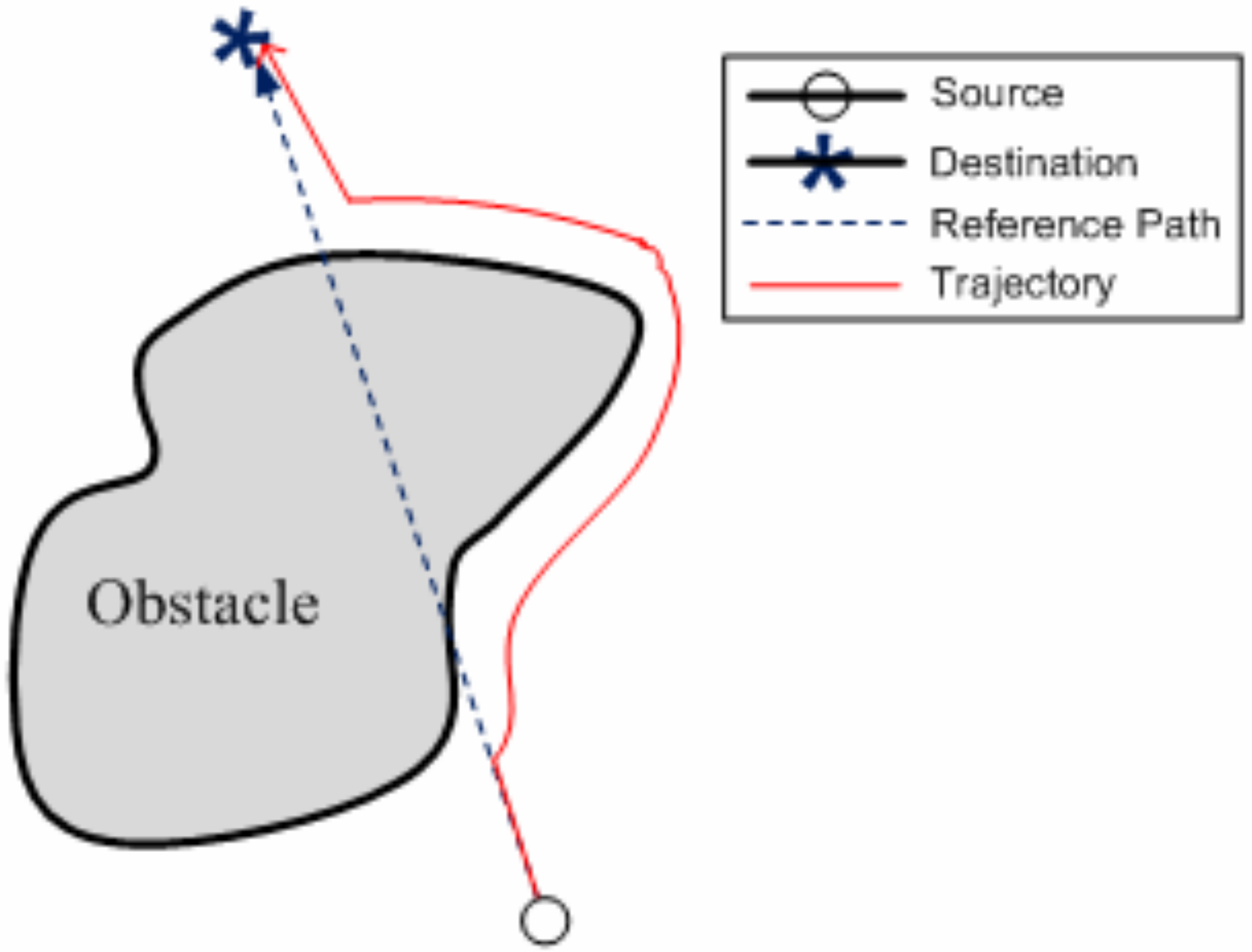


Fig. 1. Dashed blue spline is global path: a) Green spline is ideal local path; b) Red spline is actual local path

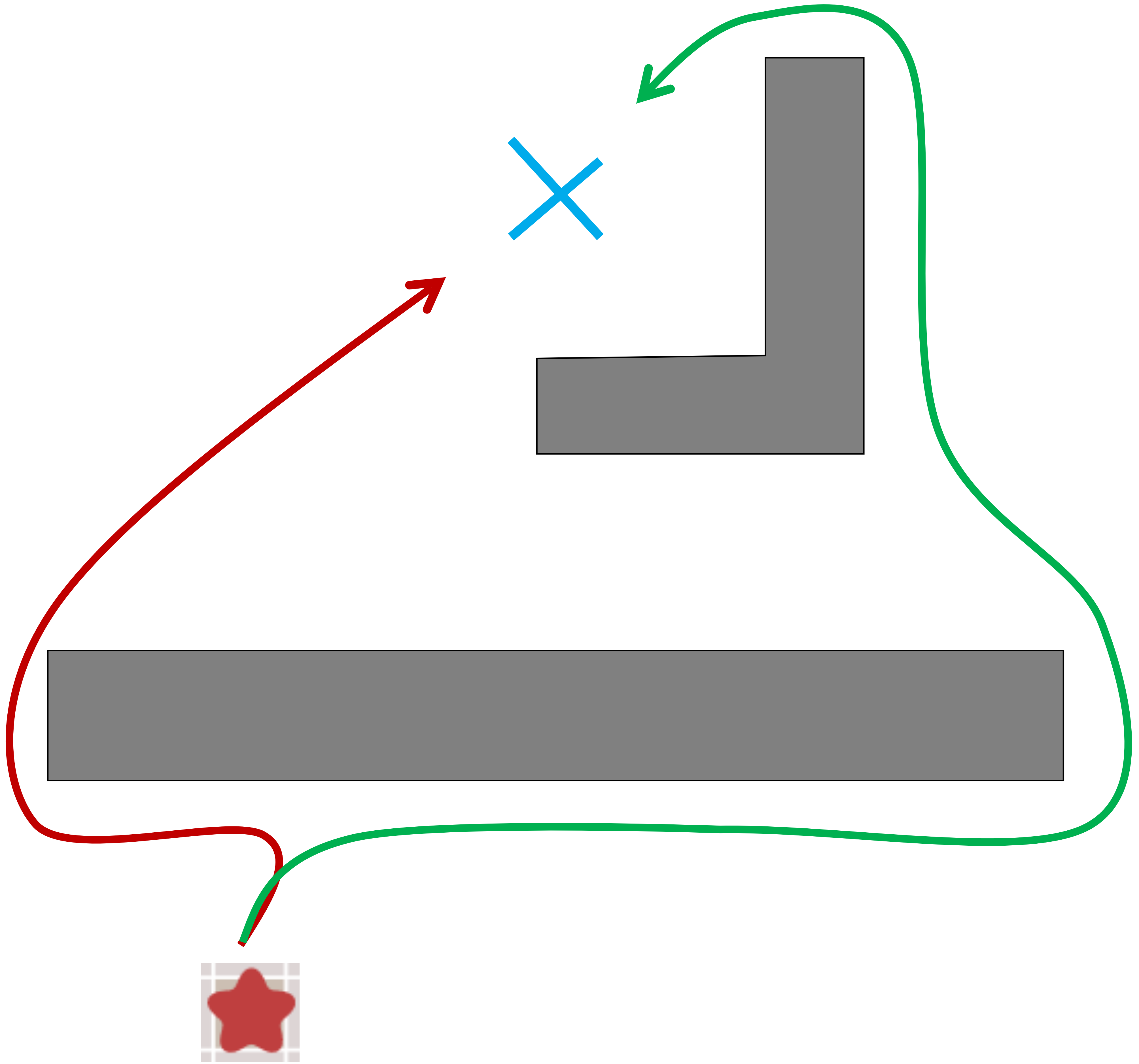
Bug algorithms

- Uses local knowledge and the direction and distance to the goal
- Basic idea
  - Follow the contour of obstacles until you see the goal
  - State 1: seek goal
  - State 2: follow wall
- Different Variants: Bug0, Bug1, Bug2
- Advantages
  - Super simple
  - No global map
  - Completeness
- Disadvantages
  - Suboptimal



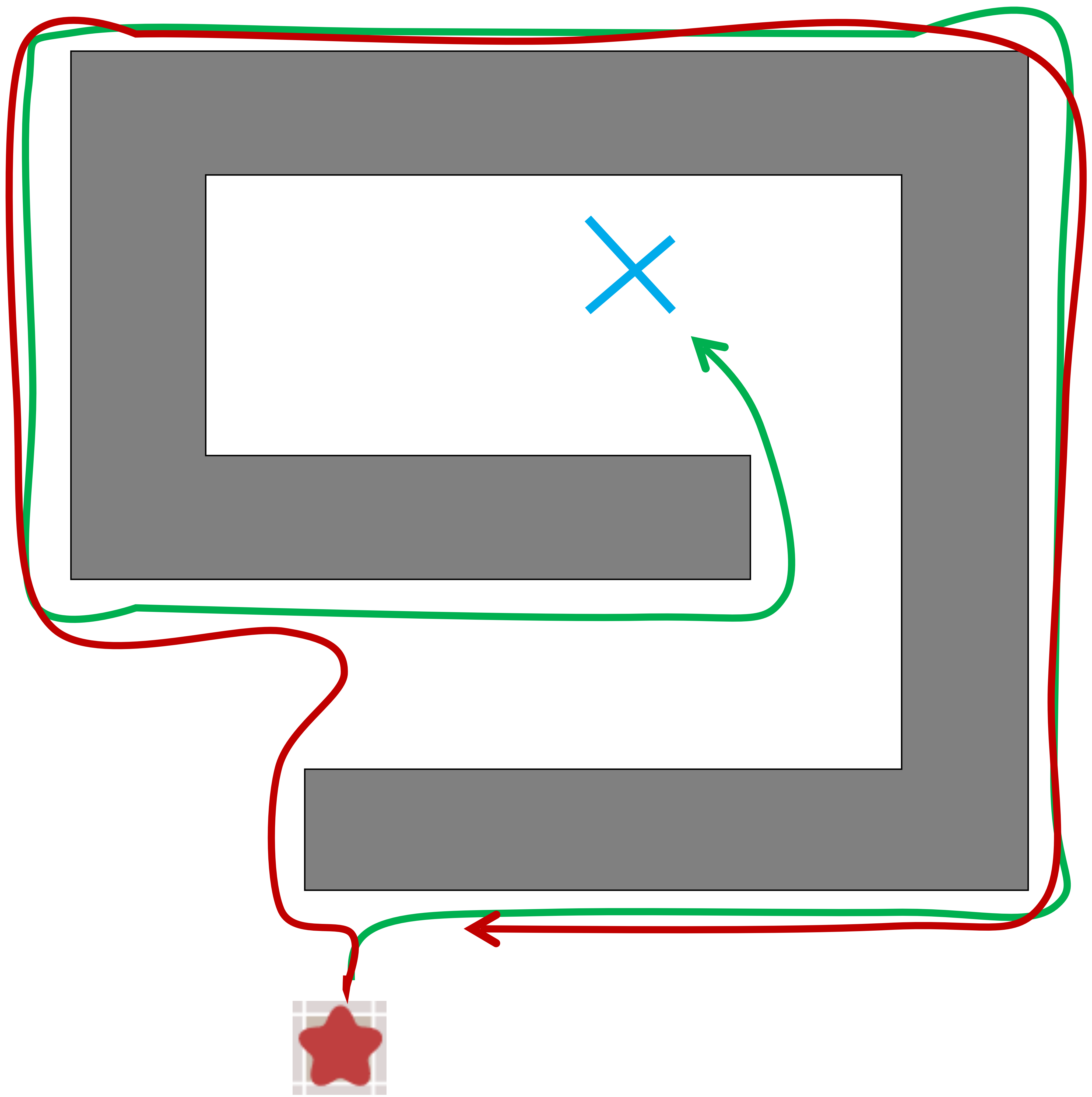
Bug 0

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
- Algorithm
  - Go towards goal
  - Follow obstacles until you can go towards goal again
  - Loop



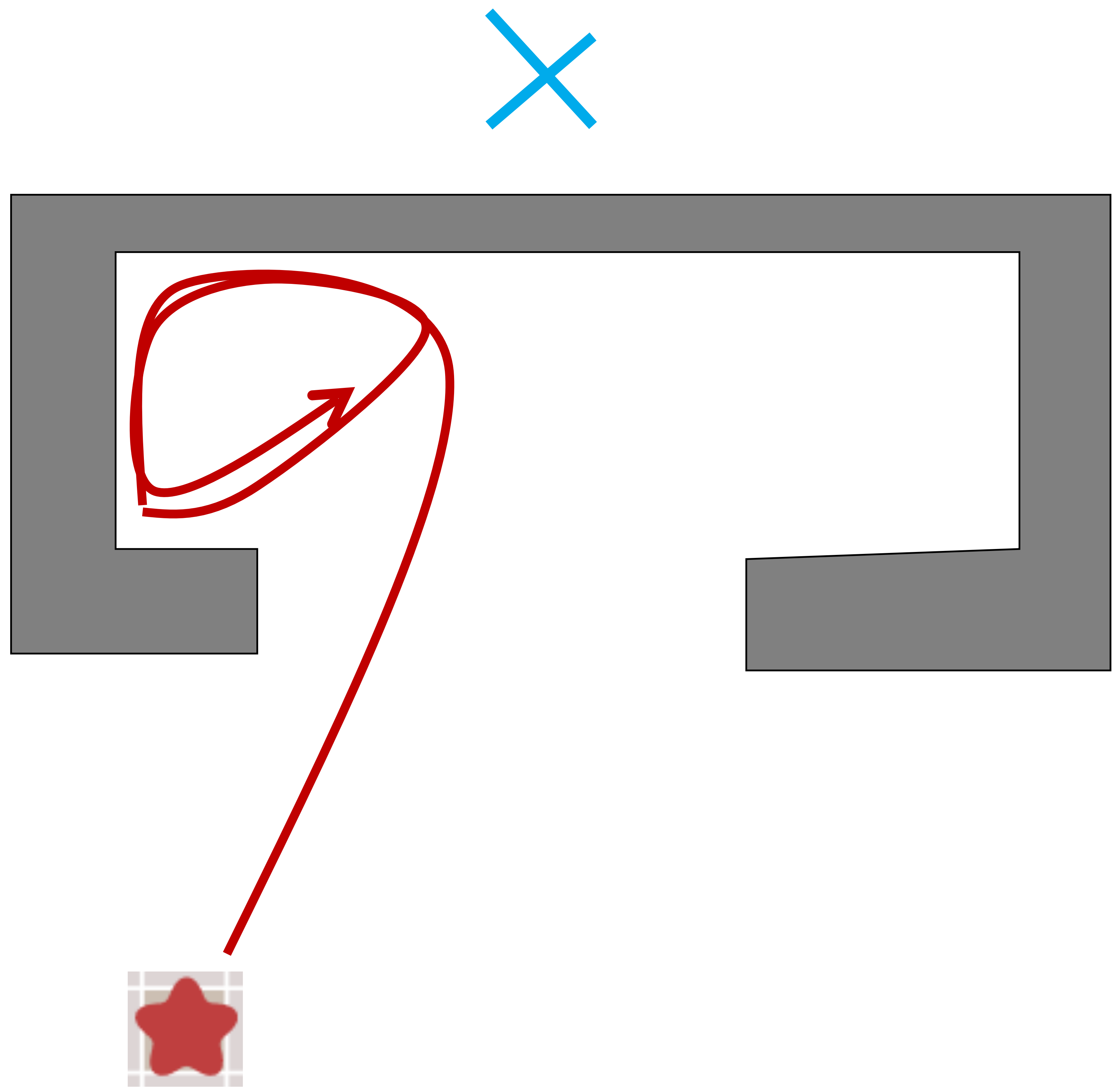
Bug 0

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
- Algorithm
  - Go towards goal
  - Follow obstacles until you can go towards goal again
  - Loop



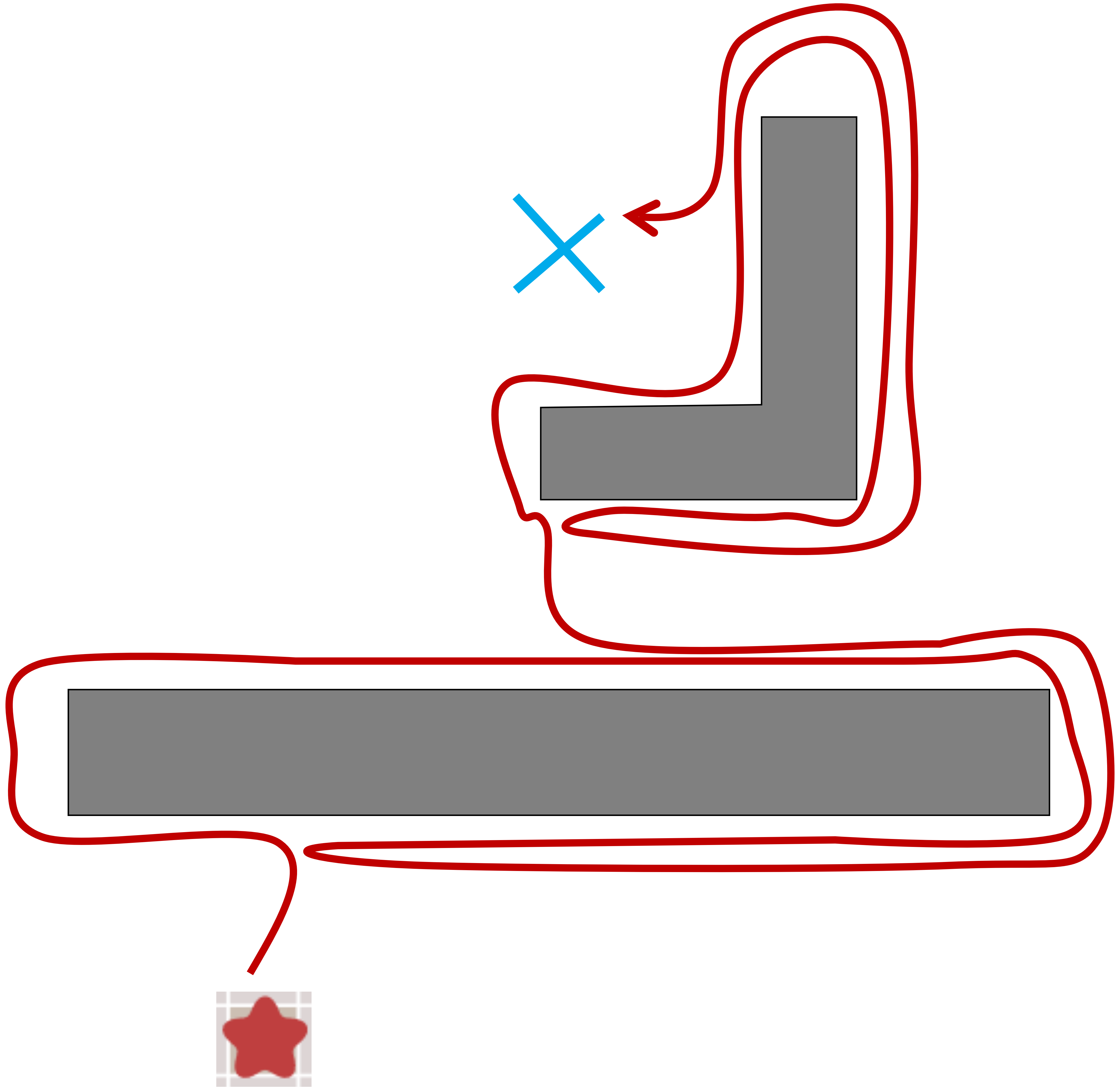
Bug 0

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
- Algorithm
  - Go towards goal
  - Follow obstacles until you can go towards goal again
  - Loop



Bug 1

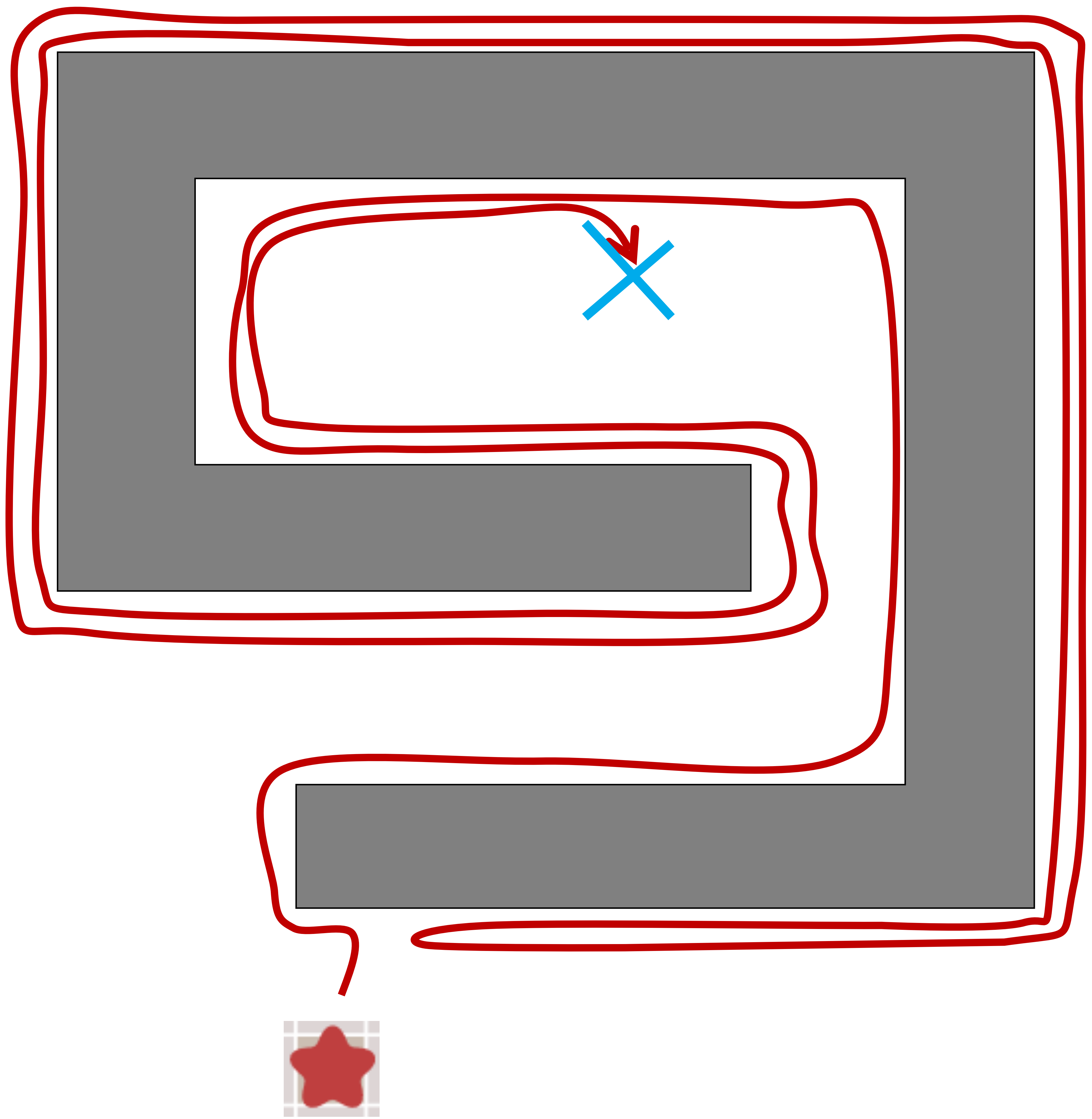
- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
- Algorithm
  - Go towards goal
  - Follow obstacles *and remember how close you got to the goal*
  - Return to the closest point, loop



Howie Choset 16-735

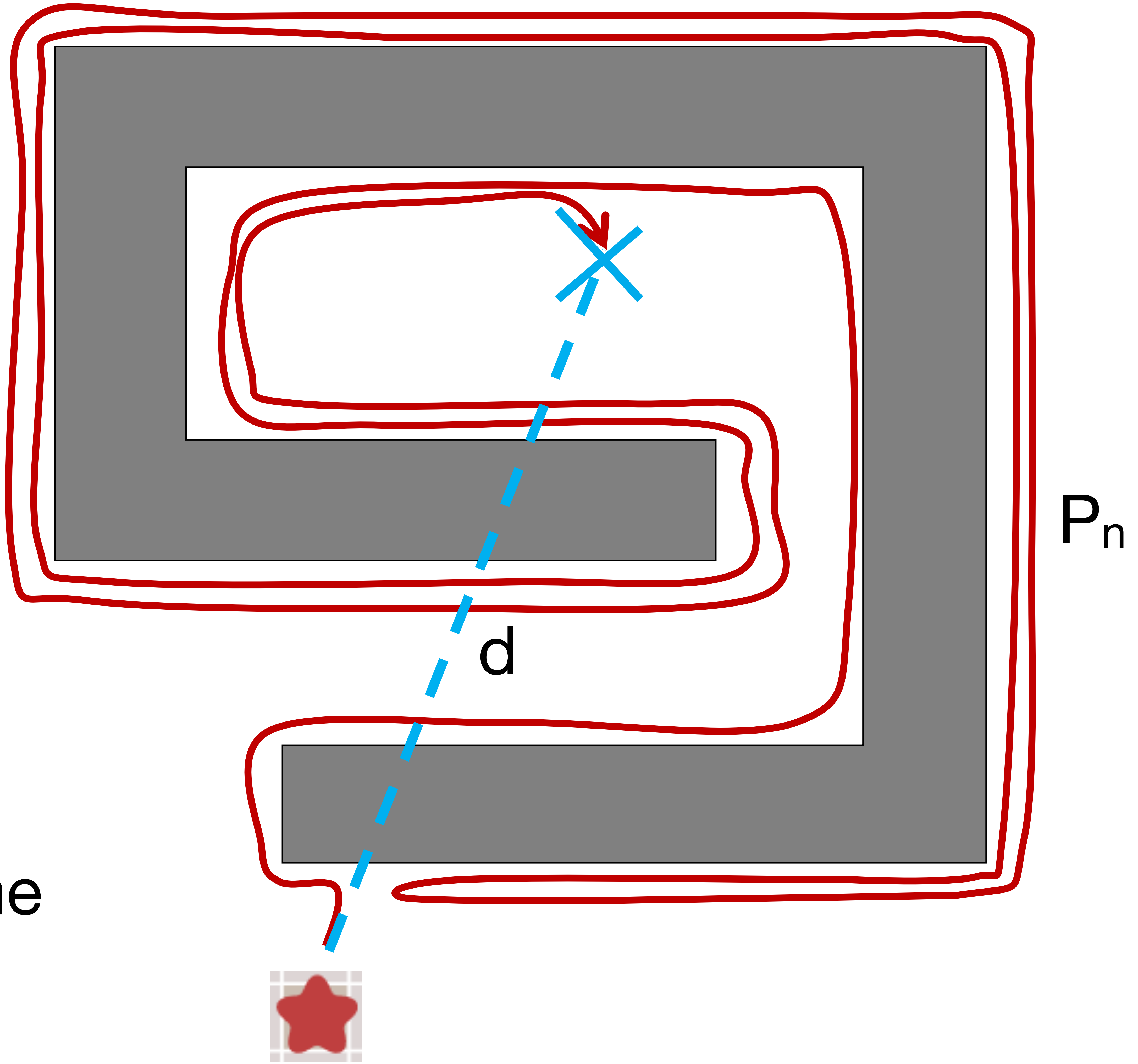
Bug 1

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
- Algorithm
  - Go towards goal
  - Follow obstacles *and remember how close you got to the goal*
  - Return to the closest point, loop



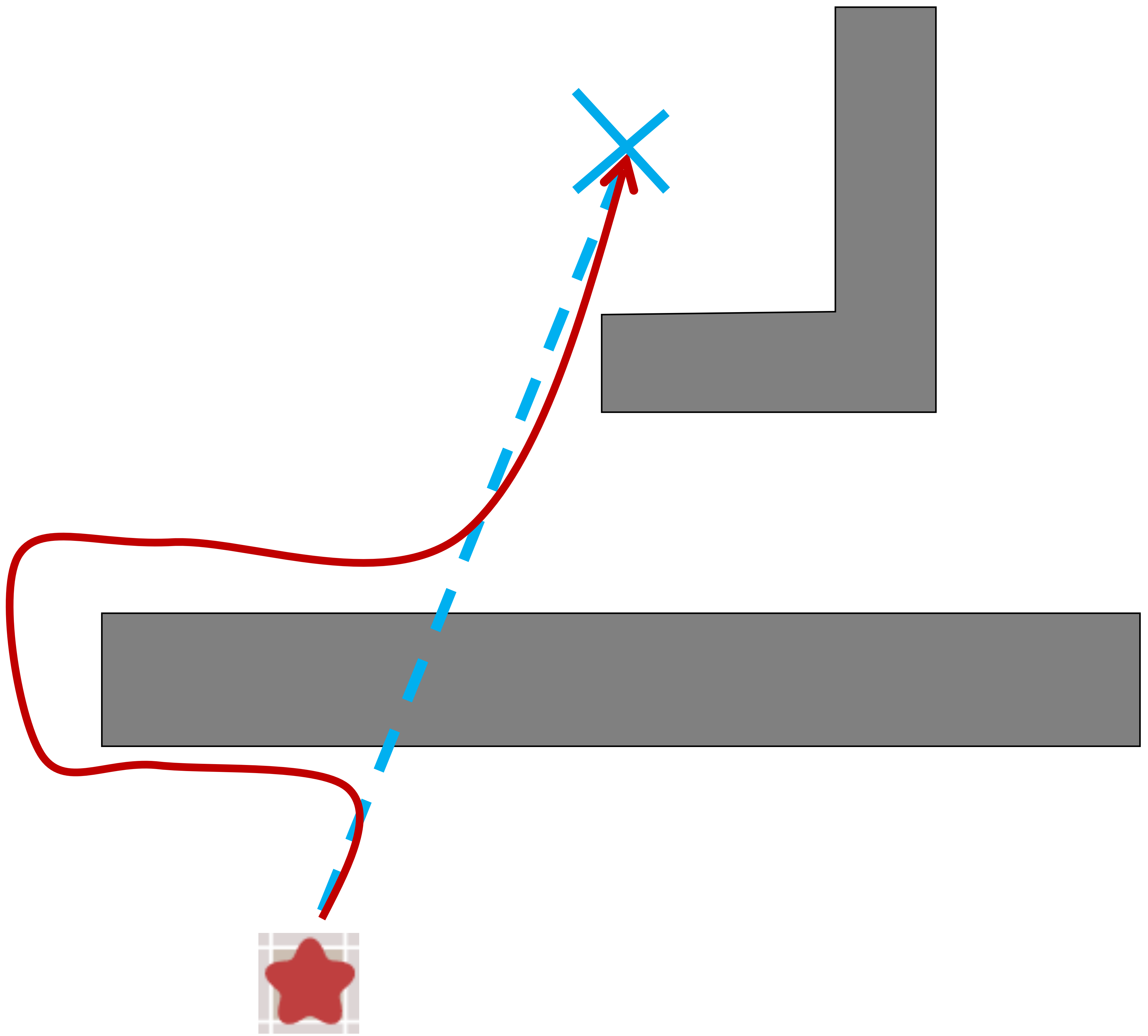
Bug 1 - formally

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
- Lower bound traversal? **d**
- Upper bound traversal? **d + 1.5∑(P_n)**
- Pros?
  - If a path exists, it returns in finite time
  - It knows if none exist!



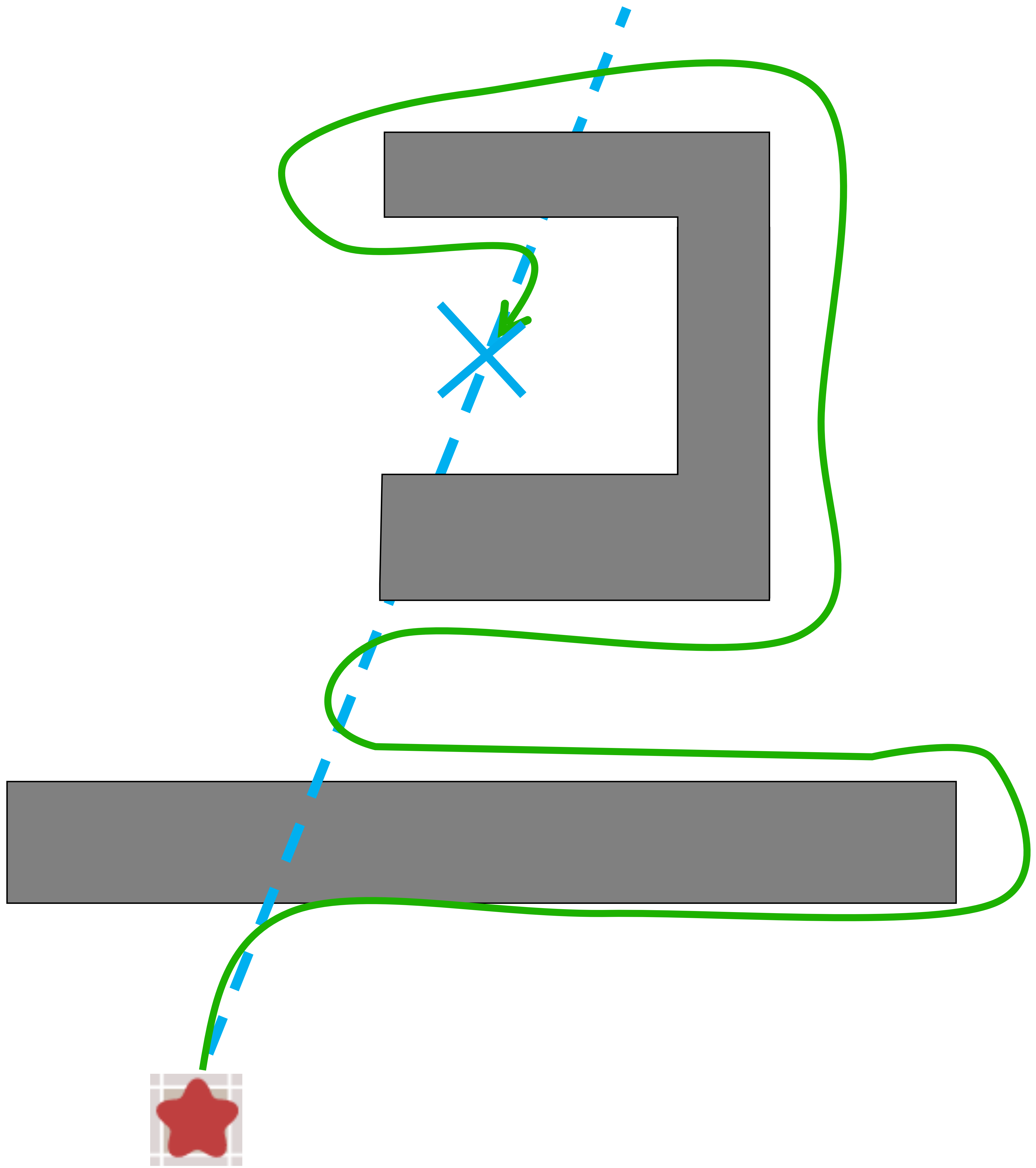
Bug 2

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
  - Original vector to the goal
- Algorithm
  - Go towards goal on the vector
  - Follow obstacles *until you are back on the vector (and closer to the obstacle)*
  - Loop



Bug 2

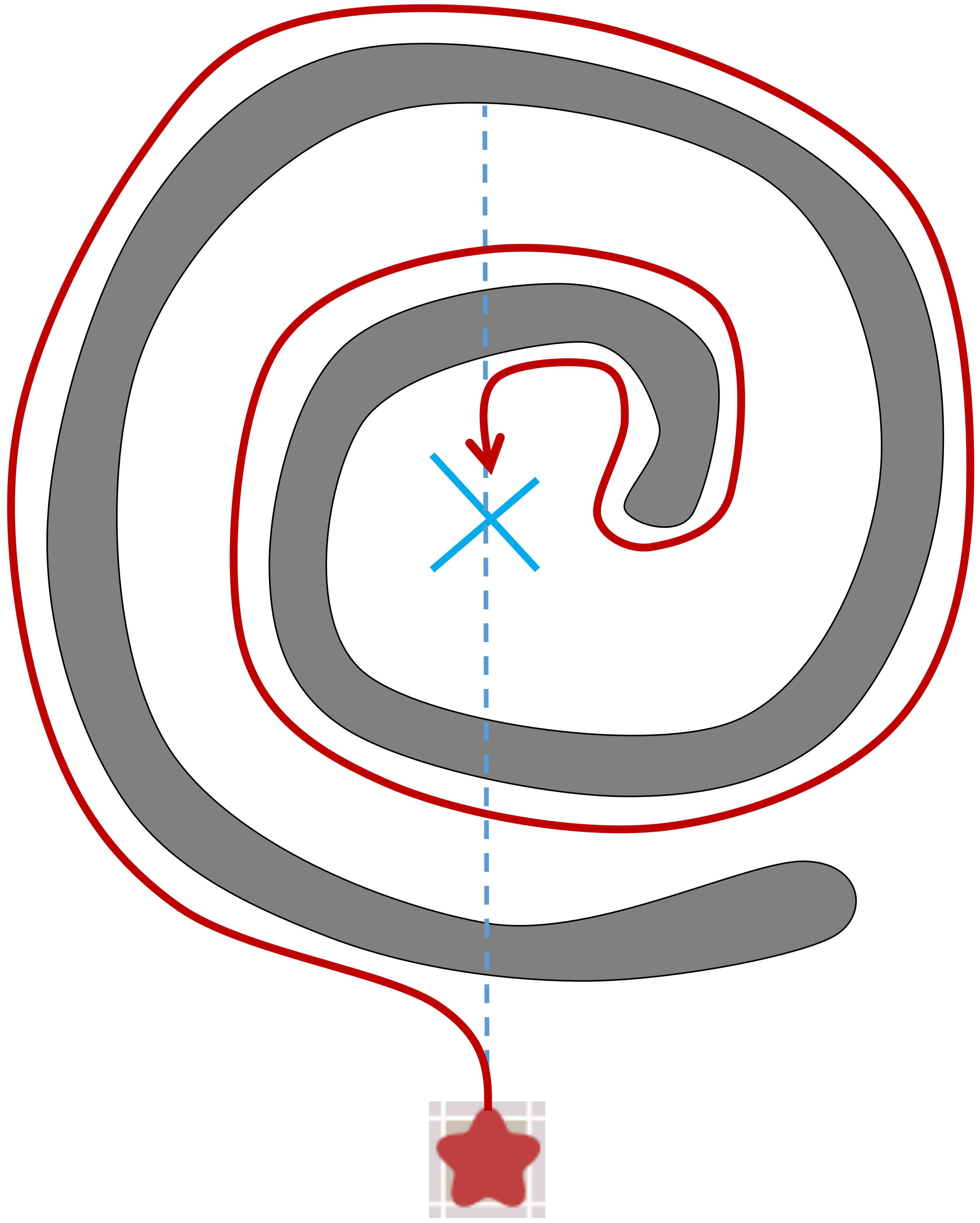
- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
  - Original vector to the goal
- Algorithm
  - Go towards goal on the vector
  - Follow obstacles *until you are back on the vector (and closer to the obstacle)*
  - Loop



Bug 2

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
  - Original vector to the goal
- Algorithm
  - Go towards goal on the vector
  - Follow obstacles *until you are back on the vector (and closer to the obstacle)*
  - Loop

What is faster, right- or left- wall following?



Battle of the bugs (1 vs 2)

Bug 1
Layout 1

Bug 1
Layout 1

https://www.youtube.com/watch?v=T2PVaKyxMmY

Battle of the bugs (1 vs 2)

Exhaustive search

Bug 1
Layout 2

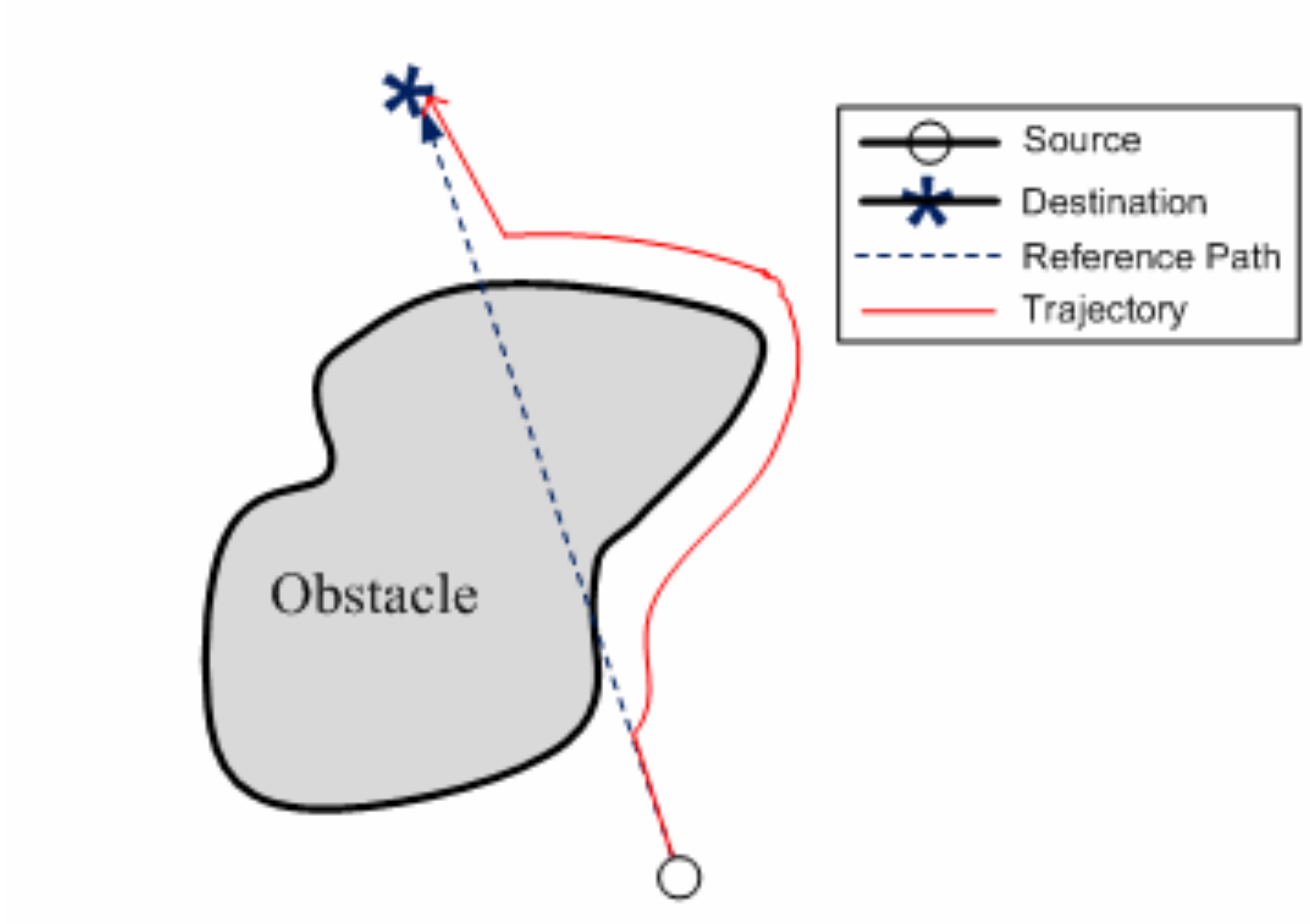
Greedy search

Bug 2
Layout 2

https://www.youtube.com/watch?v=T2PVaKyxMmY

Bug algorithms

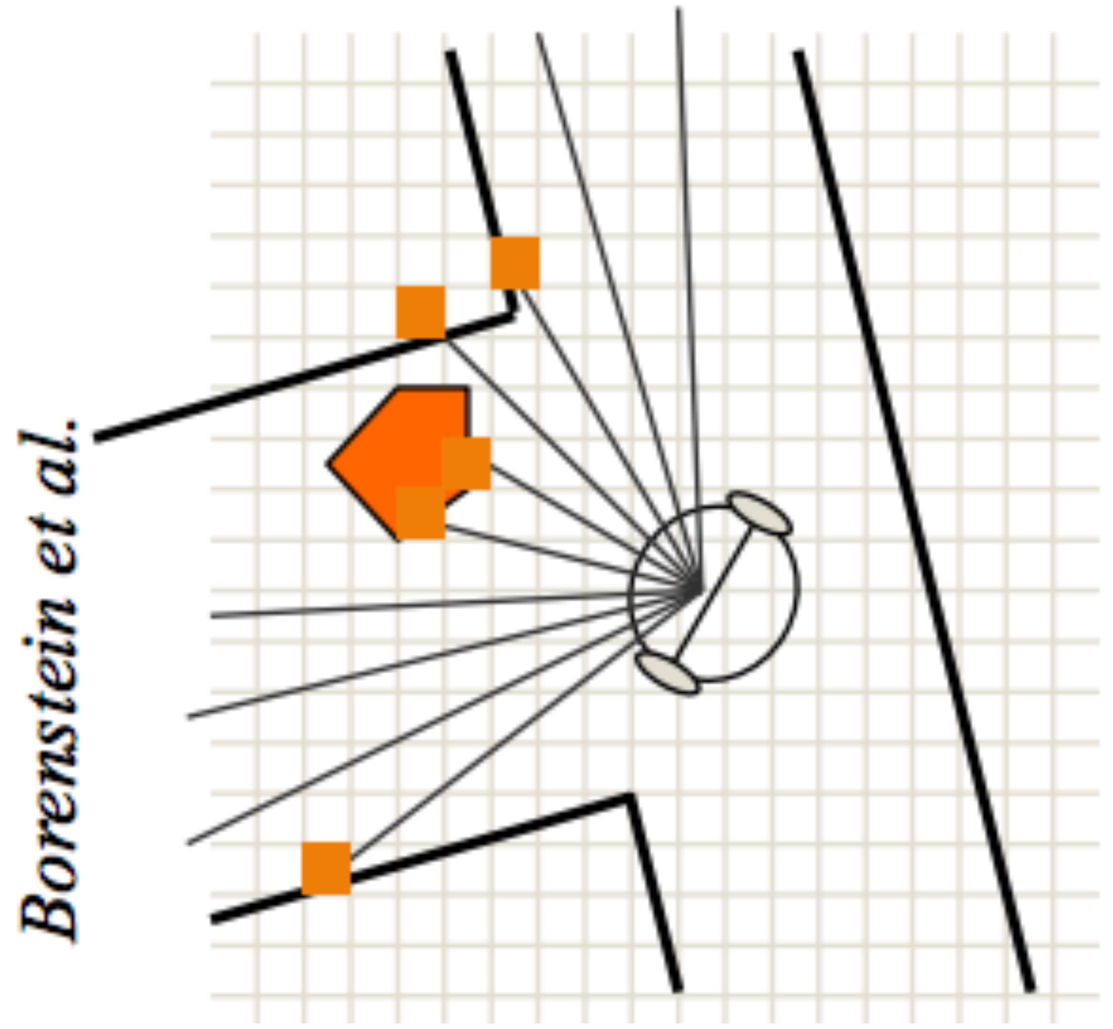
- Uses local knowledge and the direction and distance to the goal
- Basic idea
  - Follow the contour of obstacles until you see the goal
  - State 1: seek goal
  - State 2: follow wall
- Different Variants: Bug0, Bug1, Bug2

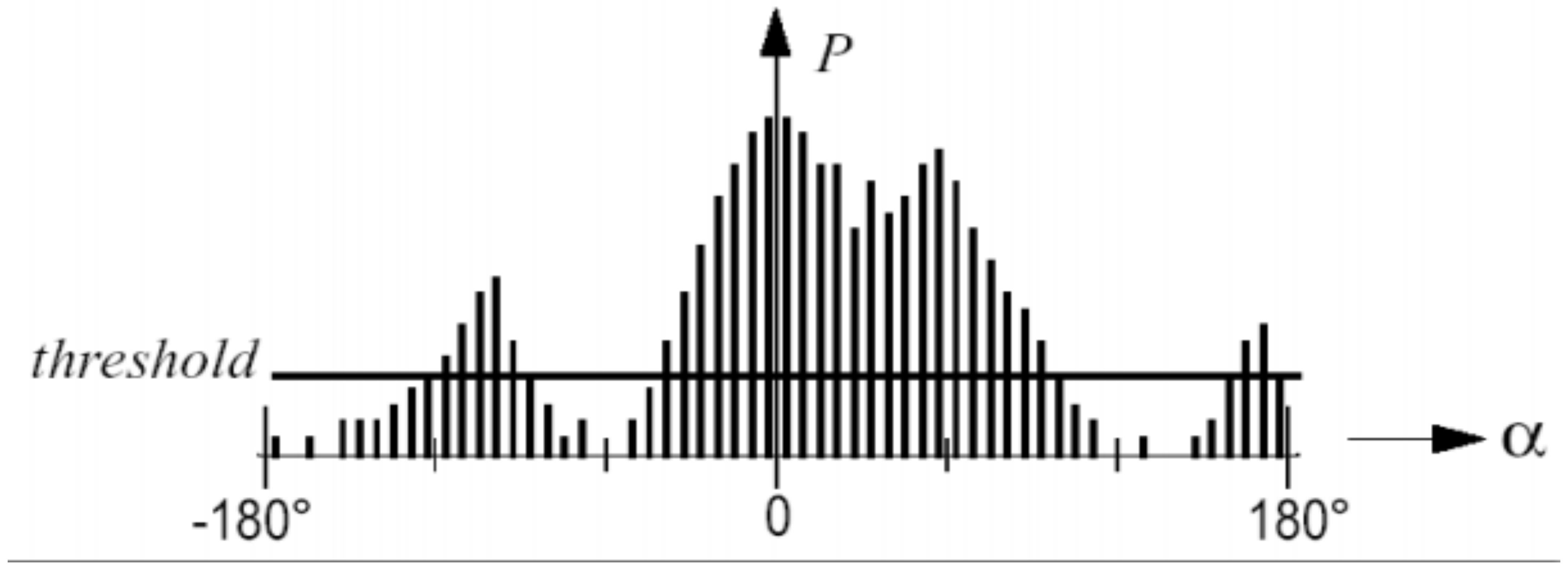


- The robot motion behavior is reactive
- Issues if the instantaneous sensor readings do not provide enough information or are noisy

Vector Field Histograms

- VFH creates a local map of the environment around the robot populated by “relatively” recent sensor readings
- Build a local 3D grid map reduce to a 1-DOF histogram
- Planning
  - Find all openings large enough for robot to pass
  - Choose the one with the lowest cost, G
  - G = a*goal_direction + b*orientation +c*prev_direction

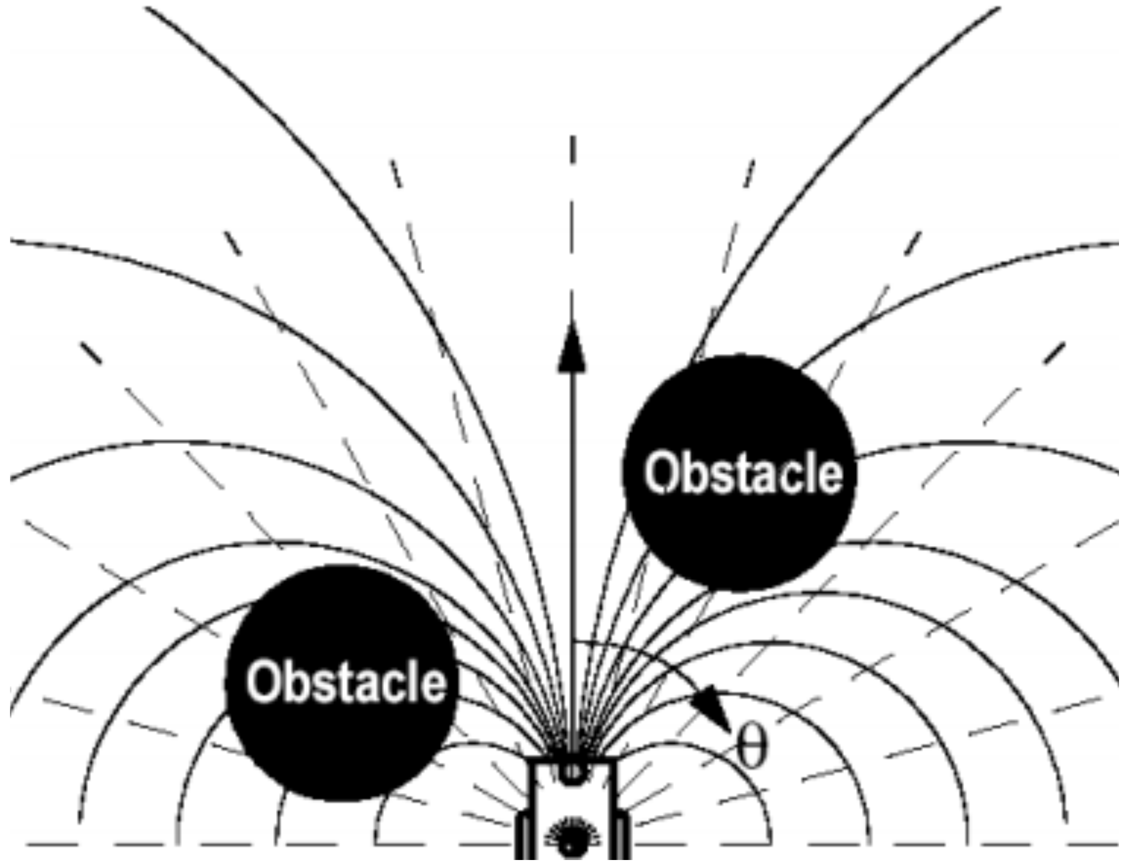


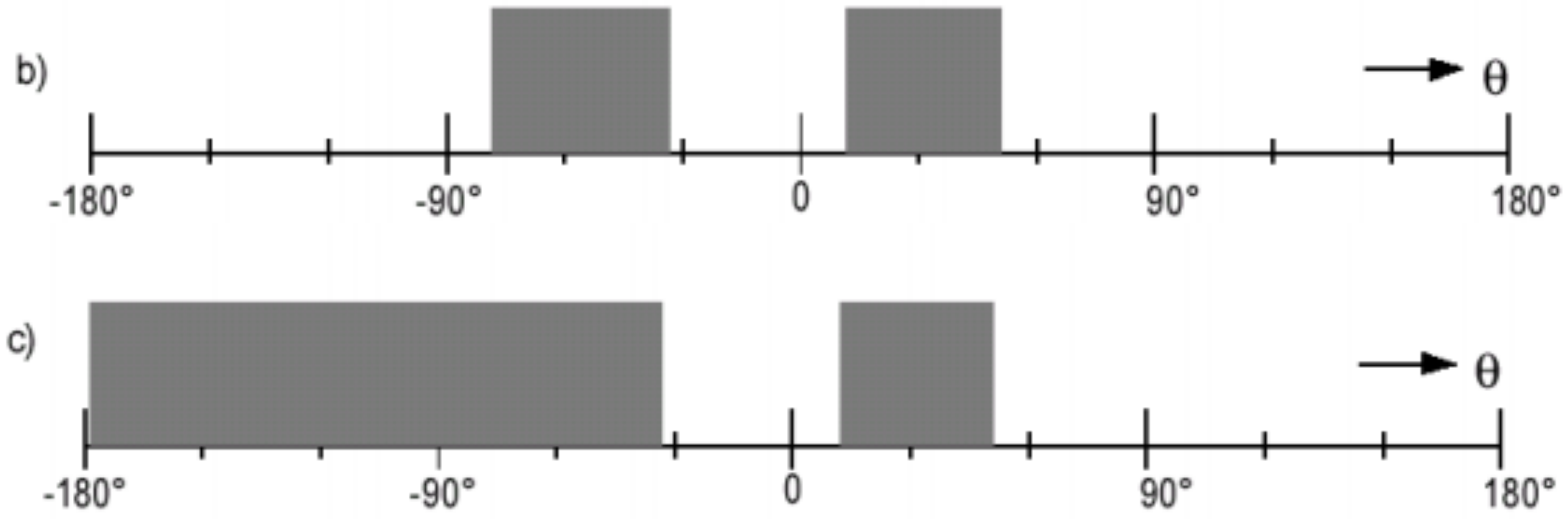


http://www.personal.umich.edu/~johannb/Papers/paper16.pdf

Vector Field Histograms

- VFH creates a local map of the environment around the robot populated by “relatively” recent sensor readings
- Build a local 3D grid map reduce to a 1-DOF histogram
- Planning
  - Find all openings large enough for robot to pass
  - Choose the one with the lowest cost, G
  - G = a*goal_direction + b*orientation +c*prev_direction
  - VFH+: incorporate kinematics
- Limitations
  - Does not avoid local minima
  - Not guaranteed to reach goal





http://www.personal.umich.edu/~johannb/Papers/paper16.pdf

Dynamic Window Approach

- Search in the velocity space (robot moves in circular arcs)
  - Takes into account robot acceleration and update rates
- A dynamic window, V_d, is the set of all tuples (v_d, ω_d) that can be reached
- Admissable velocities, V_a, include those where the robot can stop before collision
- The search space is then $V_r = V_s \cap V_a \cap V_d$
- Cost function: $G(v,\omega) = \sigma(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot velocity(v,\omega))$

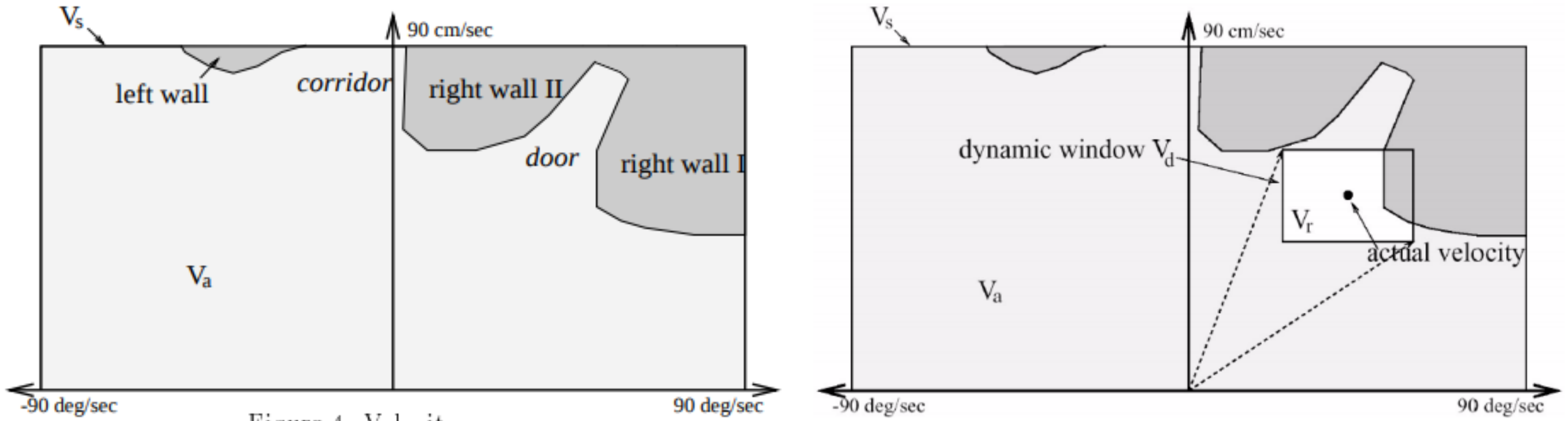


Figure 4. Velocity space

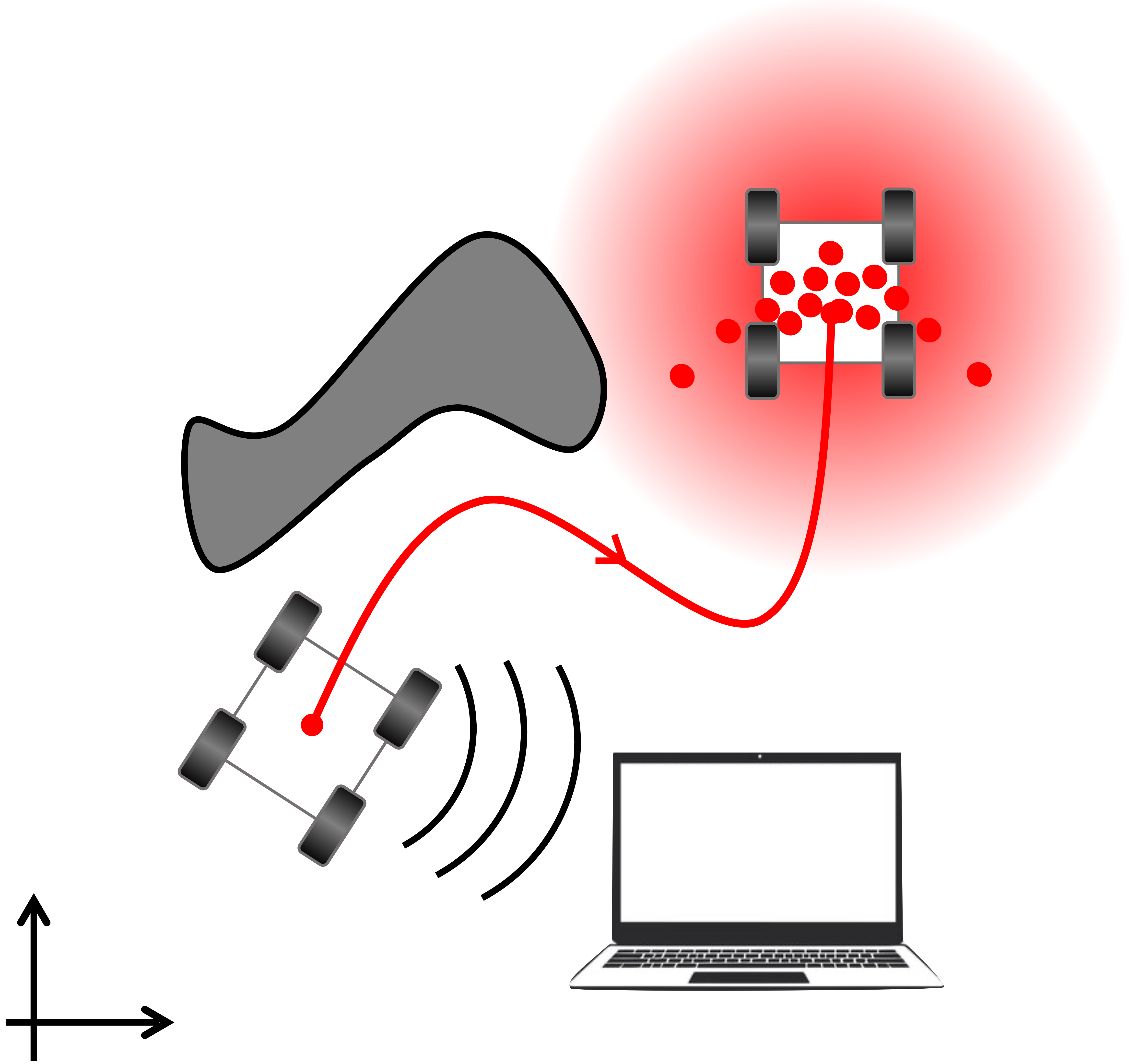
http://www4.cs.umanitoba.ca/~jacky/Teaching/Courses/74.795-LocalVision/ReadingList/fox97dynamic.pdf

Local Planners

- Bug algorithms
  - Inefficient but can be exhaustive
- Vector Field Histograms
  - Takes into account probabilistic sensor measurements
- Vector Field Histograms+
  - Takes into account probabilistic sensor measurements and robot kinematics
- Dynamic window approach
  - Takes into account robot dynamics

Next module on navigation

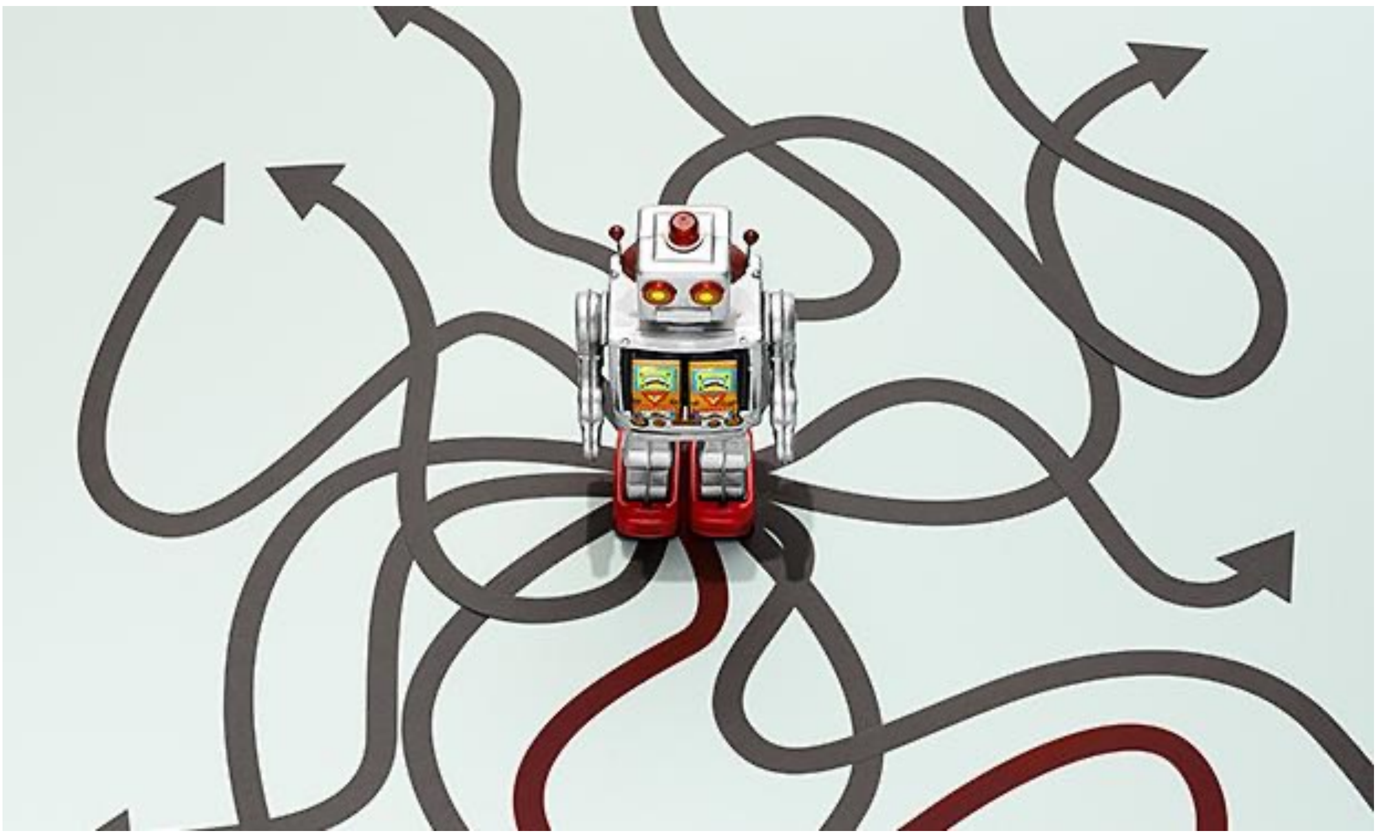
- Local planners
- Global localization and planning
  - Map representations
    - Continuous
    - Discrete
    - Topological
  - Maps as graphs
    - Graph search algorithms
      - Breadth first search
      - Depth first search
      - Dijkstras
      - A*



Localization Problem

Position Tracking	Global Localization
• Initial **robot pose is known**	• Initial **robot pose is unknown**
• Either deterministically (odometry) or through Bayesian statistic (motion and sensor models)	• Need to estimate position from scratch
• It is a "**local**" problem, as the uncertainty is local (often small) and confined to a region near the robot's true pose	• A more difficult "**global**" problem, where you cannot assume boundedness in pose error

kidnapped robot problem



Next class…

- Local planners
- Global localization and planning
  - Map representations
    - Continuous
    - Discrete
    - Topological
  - Maps as graphs
    - Graph search algorithms
      - Breadth first search
      - Depth first search
      - Dijkstras
      - A*