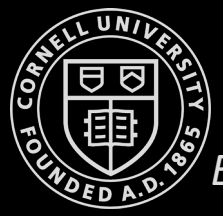
ECE 4960

Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

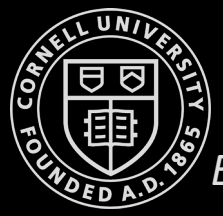
Fast Robots



Linear Systems Control

Based on “Control Bootcamp”, Steve Brunton, UW
https://www.youtube.com/watch?v=Pi7l8mMjYVE

- Linear systems review
- Eigenvectors and eigenvalues
- Stability
- Discrete time systems
- Linearizing non-linear systems
- Controllability, LQR
- Observability
    - Kalman Filters



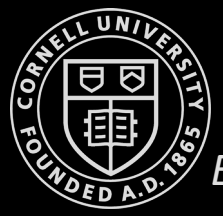
Linear Systems Control – "review of review"

- Linear system: $\dot{x} = Ax$
- Solution: $x(t) = e^{At}x(0)$
- Eigenvectors: $T = [\xi_1 \quad \xi_2 \quad \dots \quad \xi_n]$
- Eigenvalues:

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

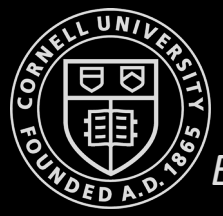
$$\mathrm{D} = \begin{bmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \dots & \\ 0 & & & \lambda_n \end{bmatrix}$$

- Linear transform: $AT = TD$
- Solution: $e^{At} = Te^{Dt}T^{-1}$
- Mapping from z to x: $x(t) = Te^{Dt}T^{-1}x(0)$
- Stability in continuous time: $\lambda = a + ib$, stable iff a<0
  - Discrete time: $x(k+1) = \tilde{A}x(k), \tilde{A} = e^{A\Delta t}$
    - Stability in discrete time: $\tilde{\lambda}^n = R^n e^{in\theta}$, stable iff R<1



Linear Systems Control – “review of review”

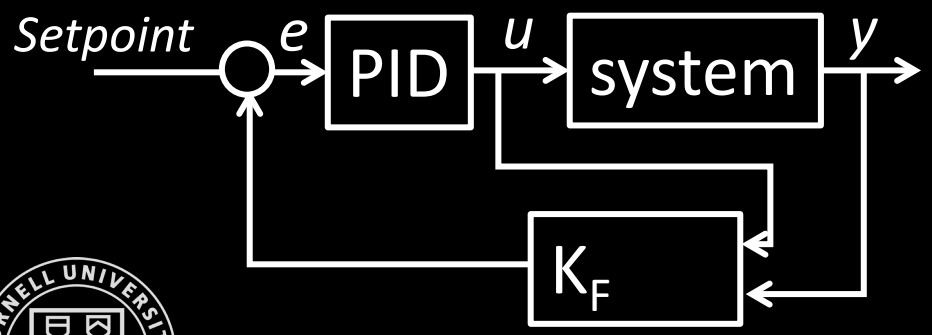
- Linearizing non-linear systems
  - Fixed points
  - Jacobian
- Controllability
  - `>>rank(ctrb(A,B))`
  - Reachability
  - Controllability Gramian
  - Pole placement
    - $\dot{x} = (A - BK)x$
  - LQR
- Observability
  - `>>rank(obsv(A,C))`
  - Observability Gramian
  - Kalman Filter



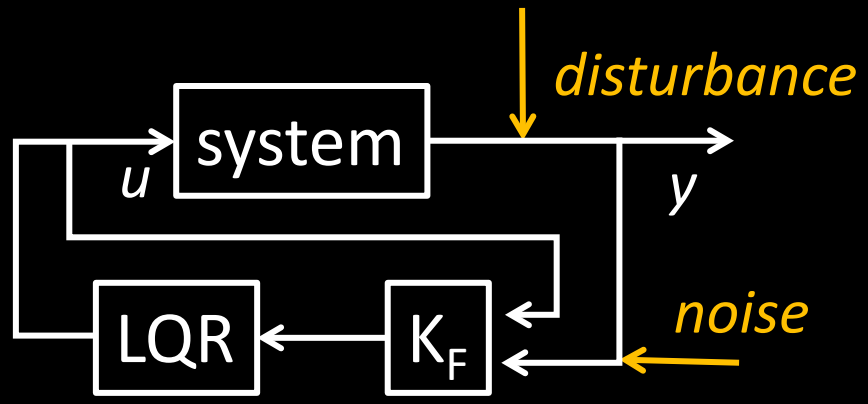
Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

Kalman Filter (continued)

KF with PID:



What you typically apply KF on:



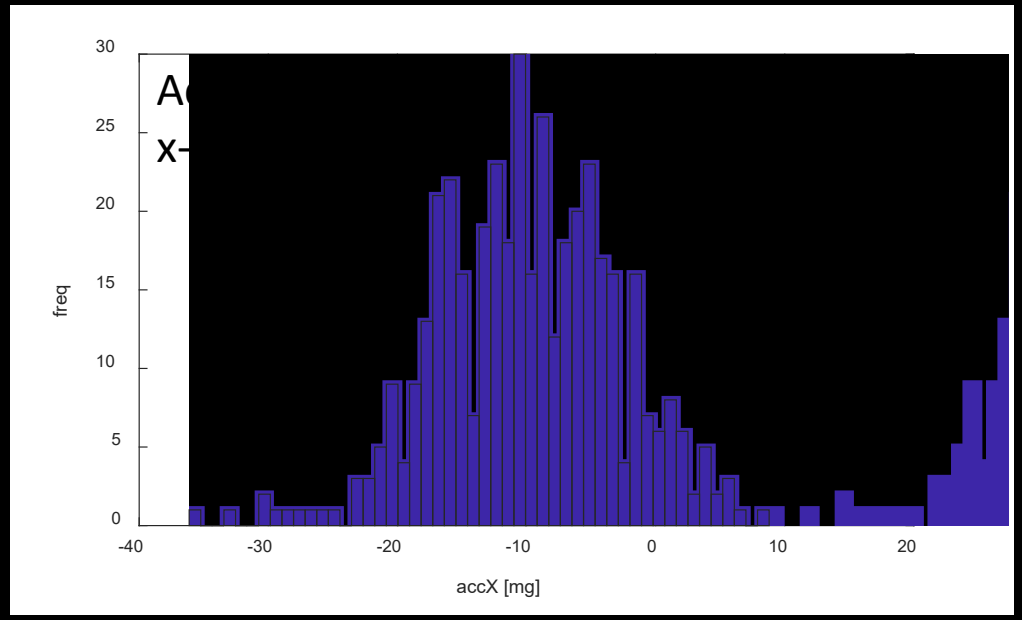
Why KF?

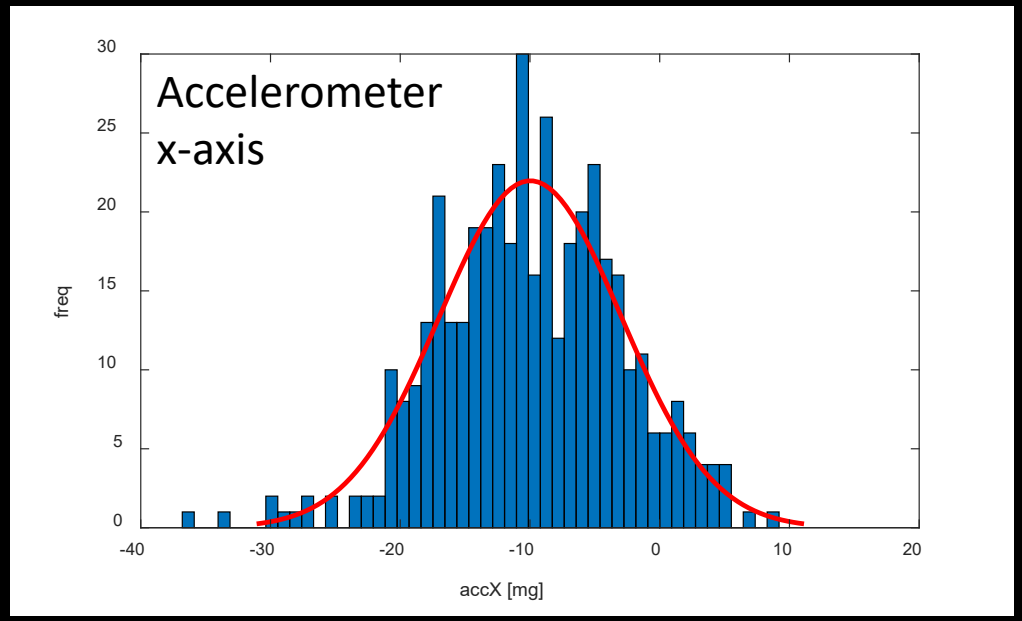
- Not full state feedback
- Bad sensors
- Slow feedback

Probabilistic Robotics

- Measurements are uncertain
- Actions are uncertain
- Models are uncertain
- States are uncertain

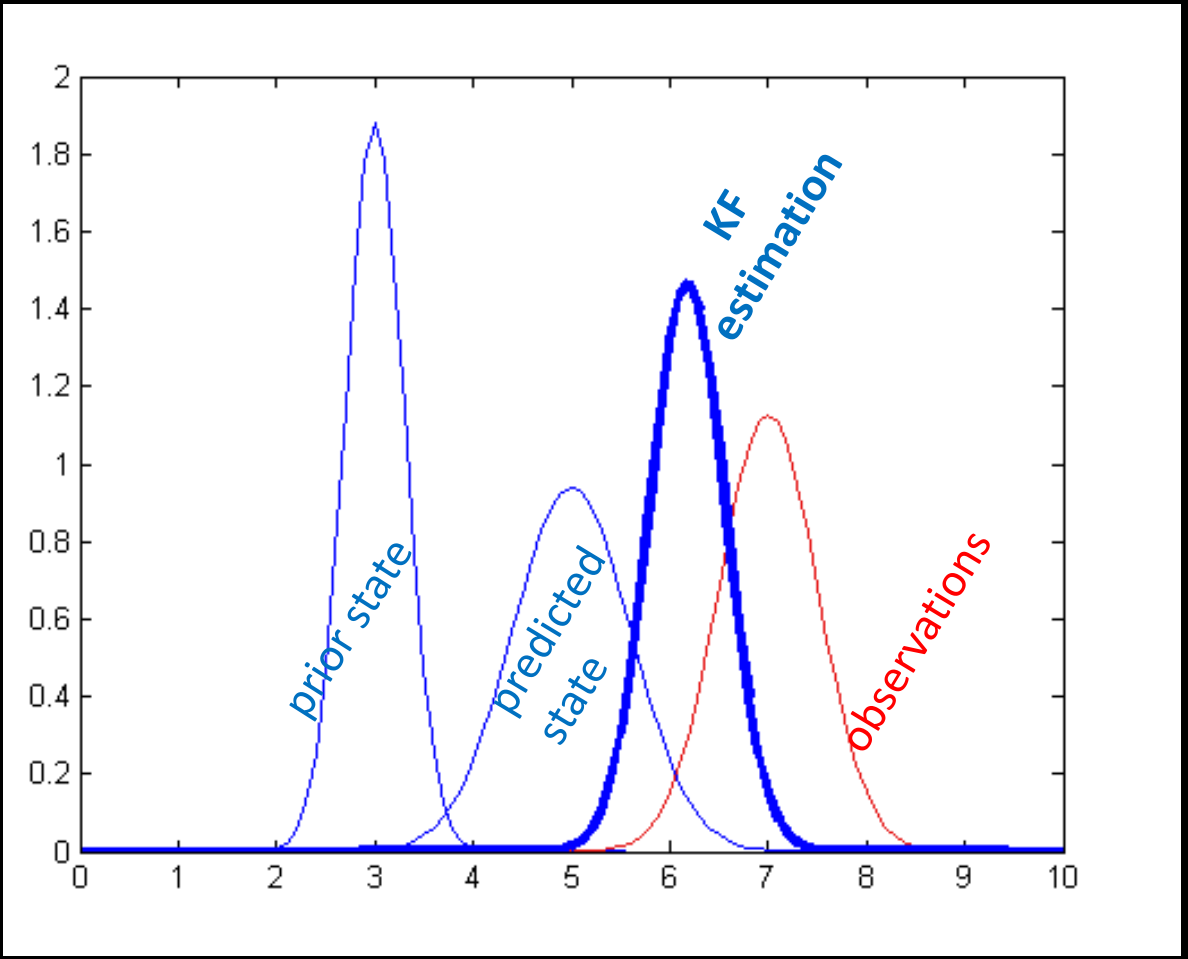
- Noise example: Accelerometer
- Solution?
  - Take more samples
  - Mean: μ = -9.97306mg
  - std dev: σ =7.0318mg
  - Variance: σ^2
- Gaussian distributions
  - $[\mu \mp \sigma]$
  - Symmetric
  - Unimodal
  - Sum to “unity”





Kalman Filter

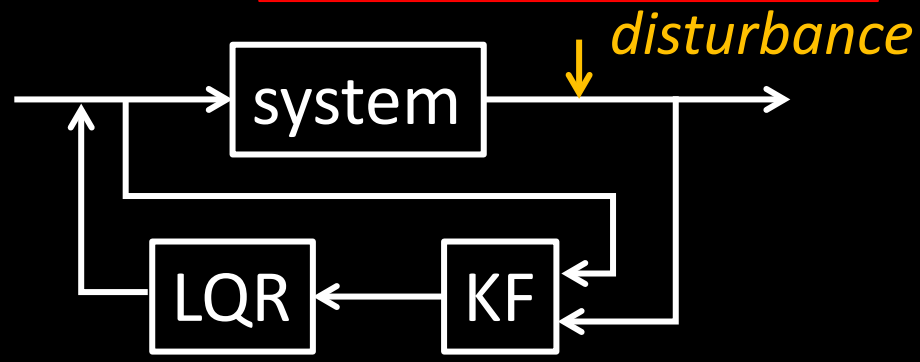
- Incorporate uncertainty to get better estimates based on inputs and observations

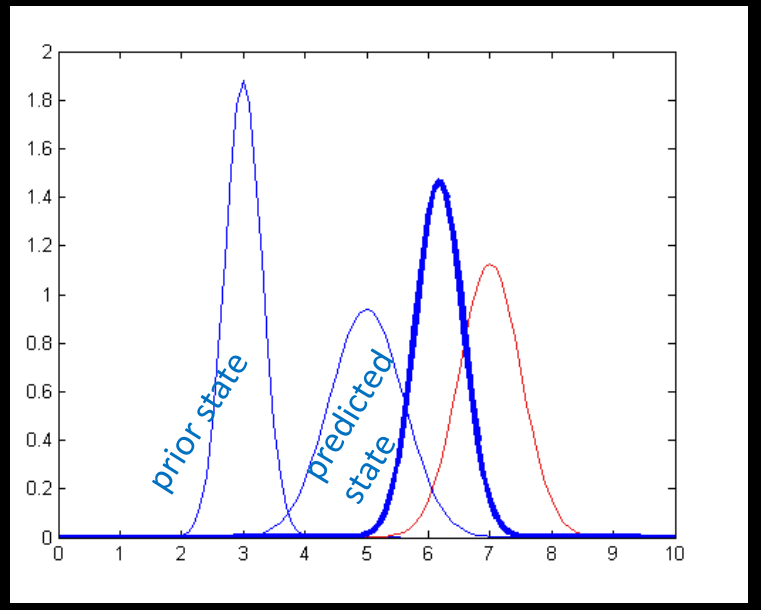


Kalman Filter

State estimate: $\mu(t)$
State uncertainty: $\Sigma(t)$
Process noise: Σ_u

- Assume that posterior and prior belief are Gaussian variables
  - Prediction step
    - $x(t) = A\,x(t-1) + Bu(t) + n$, where…
      - $\mu_p(t) = A\,\mu(t-1) + B\,u(t)$
      - $\Sigma_p(t) = A\,\Sigma(t-1)\,A^T + \Sigma_u$

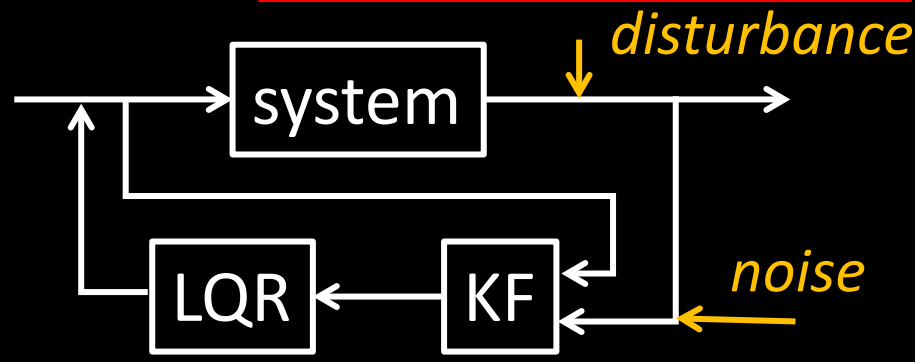


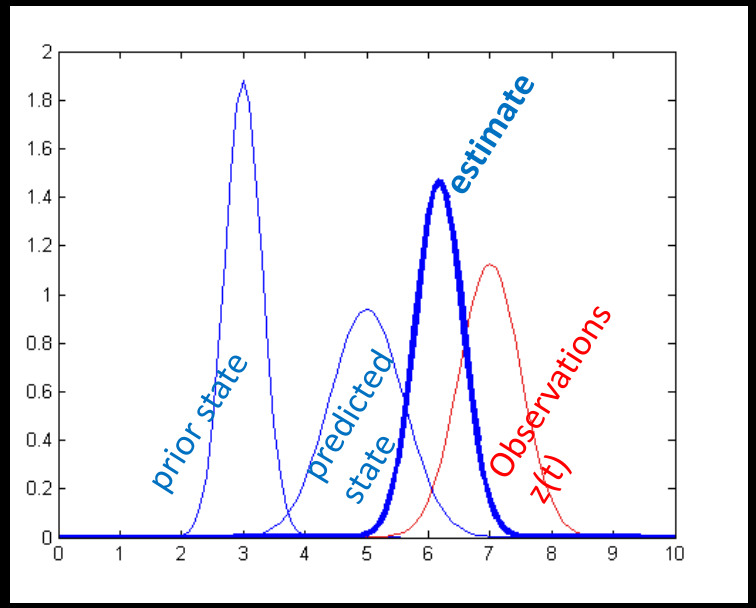


Kalman Filter

- Assume that posterior and prior belief are Gaussian variables
  - Prediction step
    - $x(t) = A\,x(t-1) + Bu(t) + n$, where…
      - $\mu_p(t) = A\,\mu(t-1) + B\,u(t)$
      - $\Sigma_p(t) = A\,\Sigma(t-1)\,A^T + \Sigma_u$
  - Update step
    - $K_{KF} = \Sigma_p(t)\,C^T\,(\,C\,\Sigma_p(t)\,C^T + \Sigma_z)^{-1}$
    - $\mu(t) = \mu_p(t) + K_{KF}\,(\,z(t) - C\,\mu_p(t)\,)$
    - $\Sigma(t) = (\,\mathbf{I} - K_{KF}\,C)\,\Sigma_p(t)$

State estimate: $\mu(t)$
State uncertainty: $\Sigma(t)$
Process noise: Σ_u
Kalman filter gain: K_{KF}
Measurement noise: Σ_z





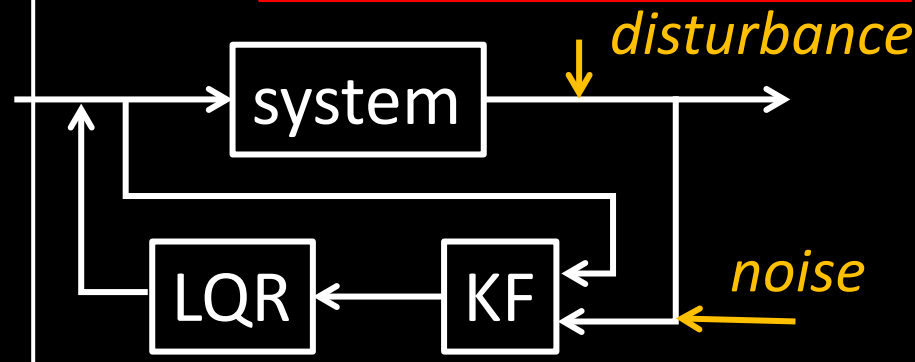
Kalman Filter Implementation

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

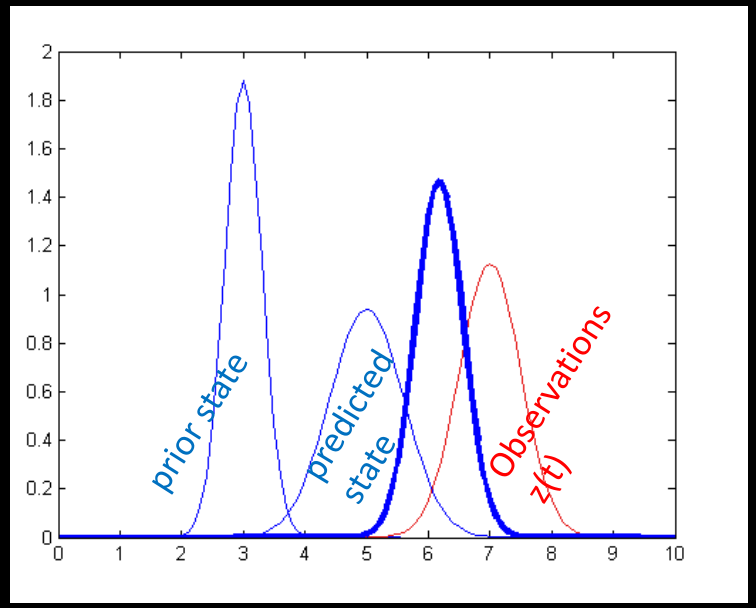
1. $\mu_p(t) = A\,\mu(t-1) + B\,u(t)$
2. $\Sigma_p(t) = A\,\Sigma(t-1)\,A^T + \Sigma_u$
3. $K_{KF} = \Sigma_p(t)\,C^T\,(C\,\Sigma_p(t)\,C^T + \Sigma_z)^{-1}$
4. $\mu(t) = \mu_p(t) + K_{KF}\,(z(t) - C\,\mu_p(t))$
5. $\Sigma(t) = (I - K_{KF}\,C)\,\Sigma_p(t)$
6. Return $\mu(t)$ and $\Sigma(t)$

prediction: steps 1–2; update: steps 3–5

State estimate: $\mu(t)$
State uncertainty: $\Sigma(t)$
Process noise: Σ_u
Kalman filter gain: K_{KF}
Measurement noise: Σ_z



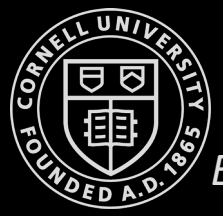
$$\Sigma_u = \begin{bmatrix} {\sigma_1}^2 & 0 & 0 \\ 0 & {\sigma_2}^2 & 0 \\ 0 & 0 & {\sigma_3}^2 \end{bmatrix}, \Sigma_z = \begin{bmatrix} {\sigma_4}^2 & 0 \\ 0 & {\sigma_5}^2 \end{bmatrix}$$



Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

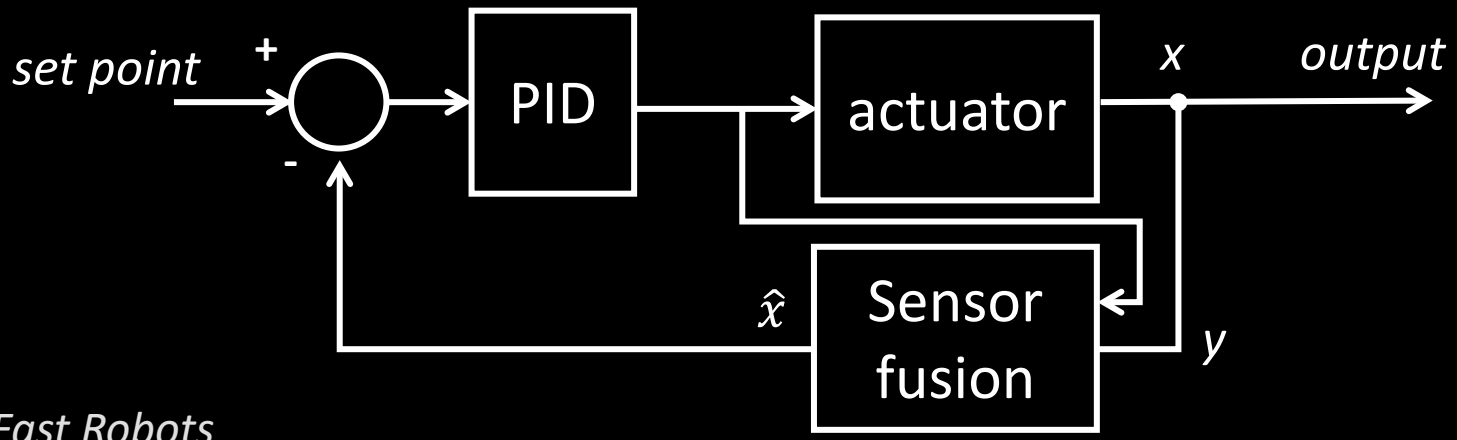
Lab Prep

- Lab 6: PID control
- Lab 7: Sensor Fusion
- Lab 8: Stunt



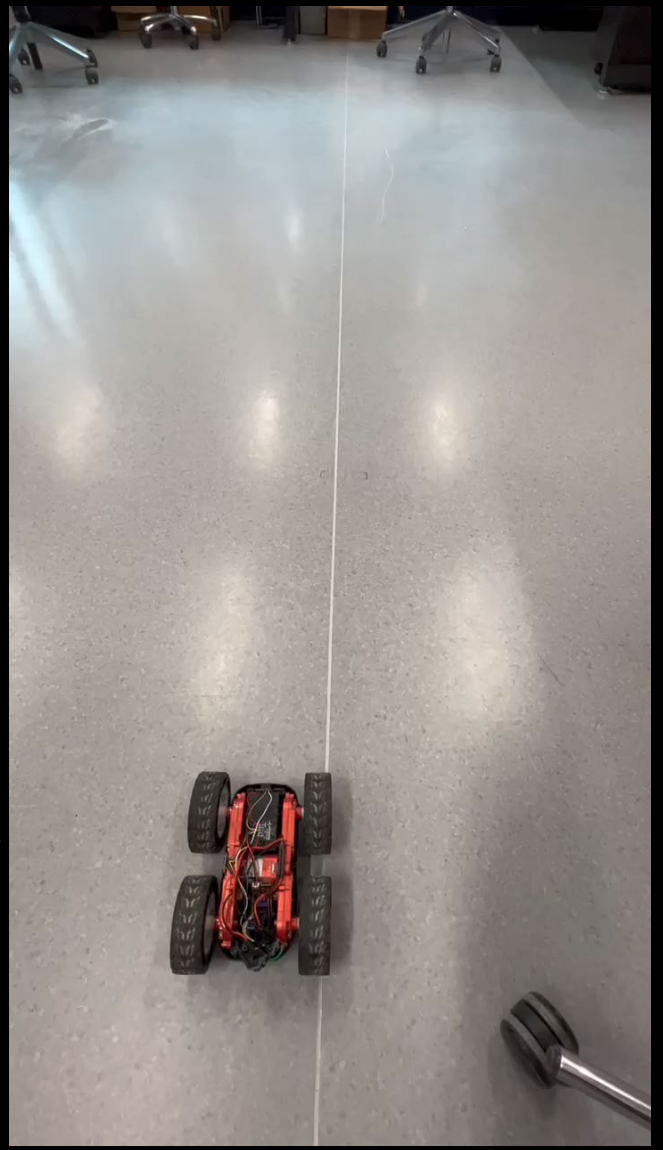
Lab 6-8: PID control – Sensor Fusion - Stunt

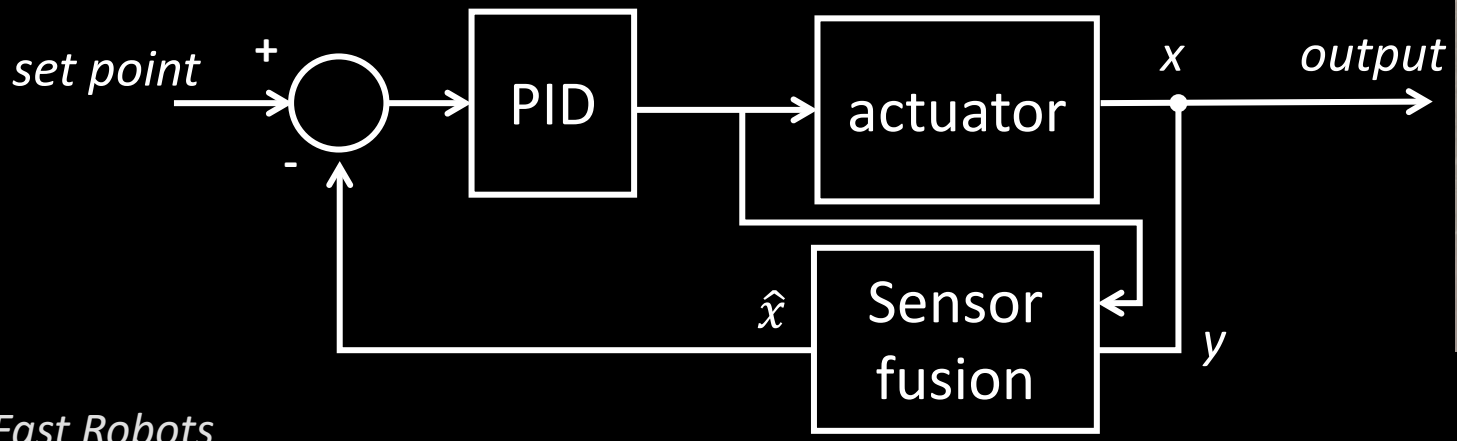
- Task A: Don’t Hit the Wall!
  - Benefit: Easiest



Lab 6-8: PID control – Sensor Fusion - Stunt

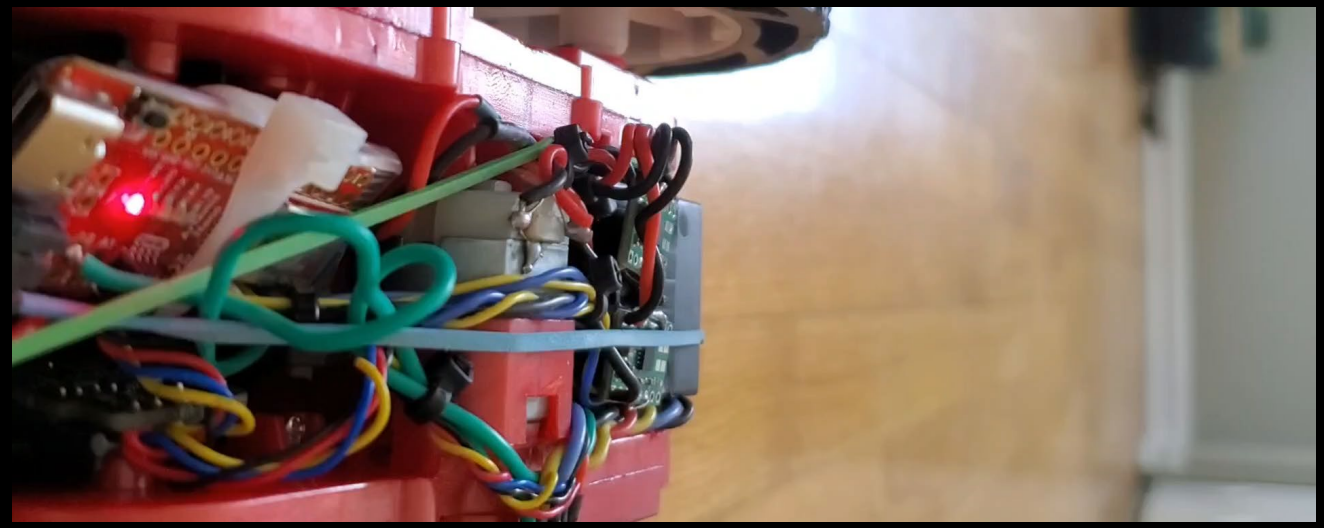
- Task A: Don’t Hit the Wall!
- Task B: Drift much?
  - Benefit: Good start to lab 9

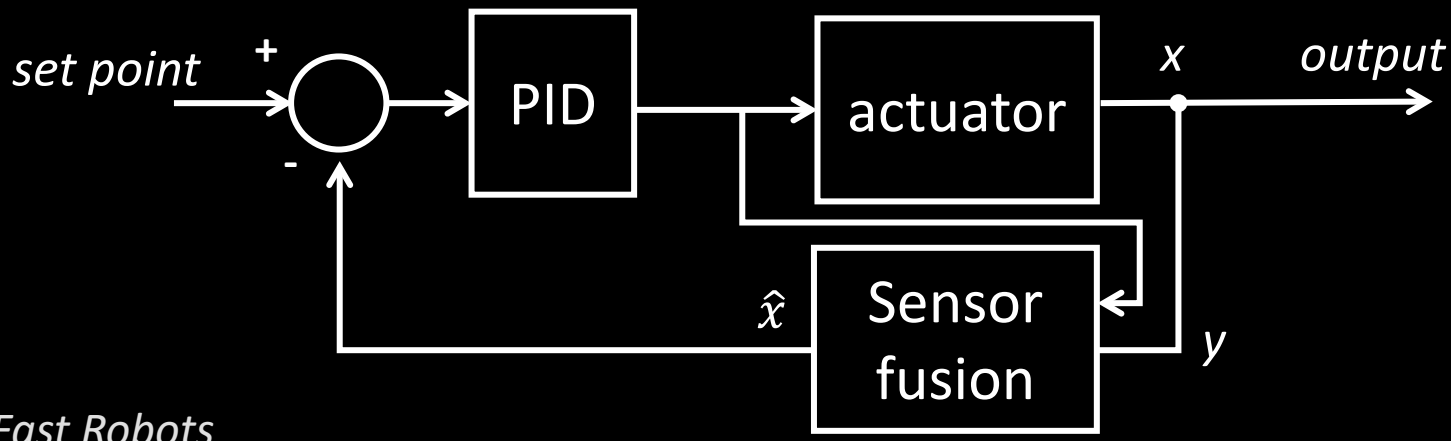




Lab 6-8: PID control – Sensor Fusion - Stunt

- Task A: Don’t Hit the Wall!
- Task B: Drift much?
- Task C: Thread the Needle!
  - Benefit: Best use of a Kalman Filter and LQG
- *Team up and work together!*



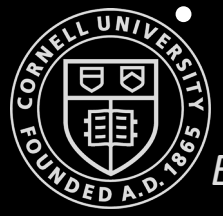


Lab 6-8: PID control – Sensor Fusion - Stunt

- Task A: Don’t Hit the Wall!
- Task B: Drift much?
- Task C: Thread the Needle!

Procedure

- Lab 6: Get basic PID to work
- Do the pre-lab: you need good debugging scripts
- Start simple and work your way up, then hack away…
  - Start slow (sampling rates, control frequency)
  - Avoid blocking statements
- Wind-up, derivative LPF, derivative kick
- Motor scaling function
  - Range of analogWrite: [0;255]
  - Directionality
  - Deadband



Cornell University: ECE 4960

Return to main page

Lab 6: Closed-loop control (PID)

Objective

The purpose of this lab is to get experience with PID control. The lab is fairly open ended, you can choose to do closed loop control on position or orientation, and you may do so with whatever works best for your system (P, PI, PID, PD). Your hand-in will be judged upon your demonstrated understanding of PID control and practical implementation constraints, and the quality of your solution.

This lab is part of a series of labs (6-8) on PID control, sensor fusion, and stunts, and you will continue to improve on your system throughout these weeks. This week, we will simply aim to get the basic behavior working. While we give you tips for improving the behavior now, if you run out of time, you can optimize more over the coming weeks.

Parts Required

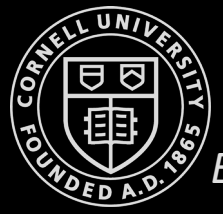
- 1 x R/C stunt car
- 1 x SparkFun RedBoard Artemis Nano
- 1 x USB cable
- 2 x Li-Po 3.7V 650mAh (or more) battery
- 2 x Dual motor driver
- 2 x 4m ToF sensor
- 1 x 9DOF IMU sensor
- 1 x Qwiic connector

Lab 6-8: PID control – Sensor Fusion - Stunt

- Task A: Don’t Hit the Wall!
- Task B: Drift much?
- Task C: Thread the Needle!

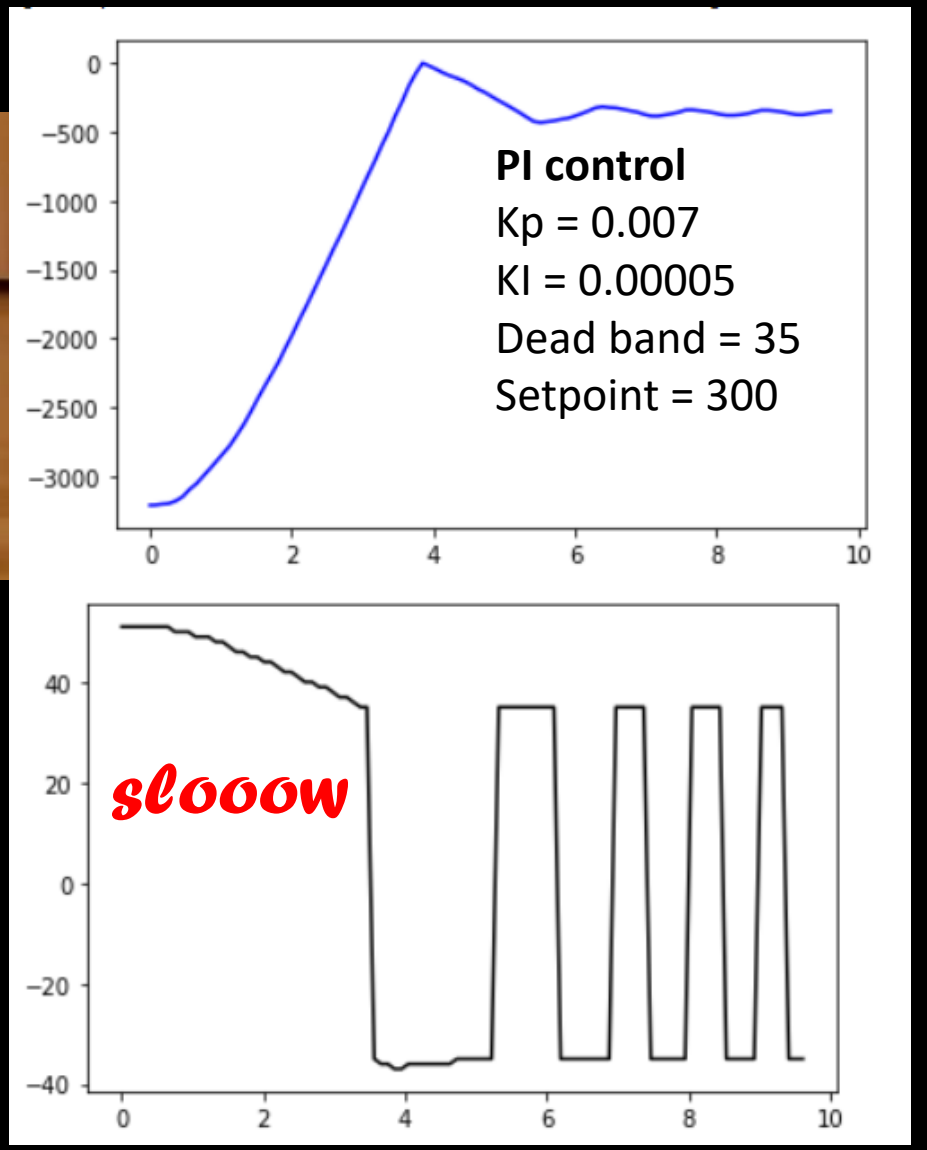
Procedure

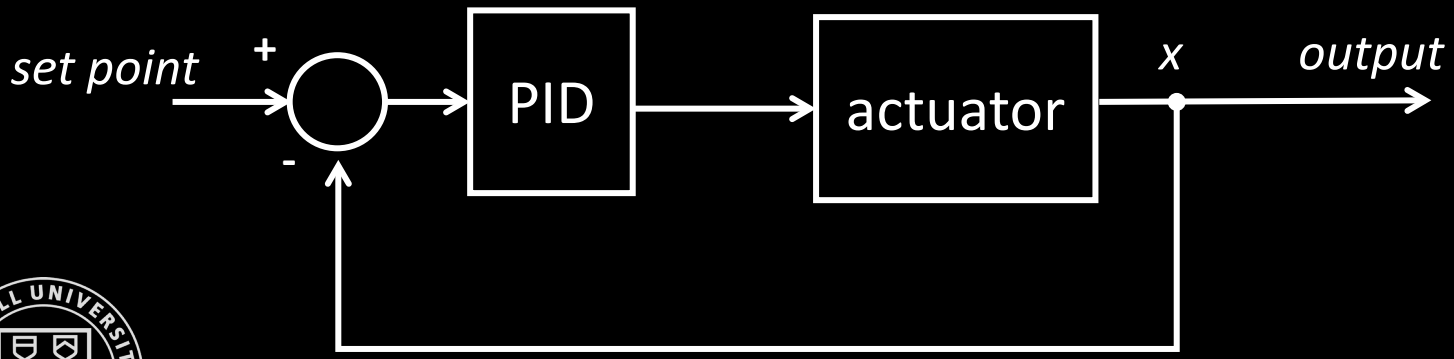
- Lab 6: Get basic PID to work
- Lab 7: Sensor Fusion
  - Approximate the state space equations
    - Step response
  - Implement Kalman Filter
    - Determine process and measurement noise
    - Try it offline on solution from lab 6
    - Try it online on your robot
  - Lab 8: Use KF and PID control to execute fast stunts



Task A: Don’t Hit the Wall

- Lab 6, Task A, example solution

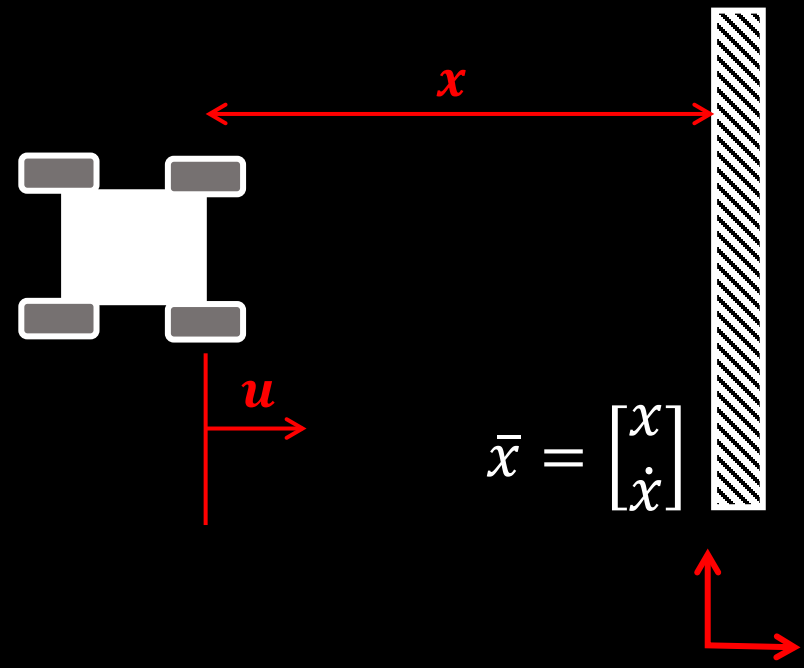




Lab 7, Task A: State Space Equations

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

What is d and m?



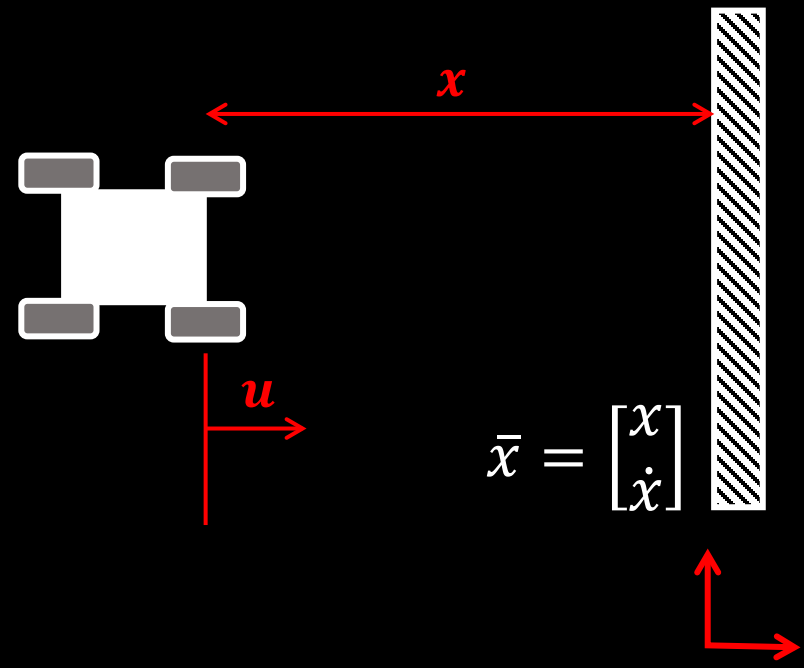
State space equation

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

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

Lab 7, Task A: State Space Equations

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



What is d and m?

- At steady state (cst speed), we can find *d*
  - $0 = \frac{u}{m} - \frac{d}{m}\dot{x}$
  - $0 = \frac{u}{m} - \frac{d}{m}\dot{x} \quad \leftrightarrow \quad d = \frac{u}{\dot{x}}$

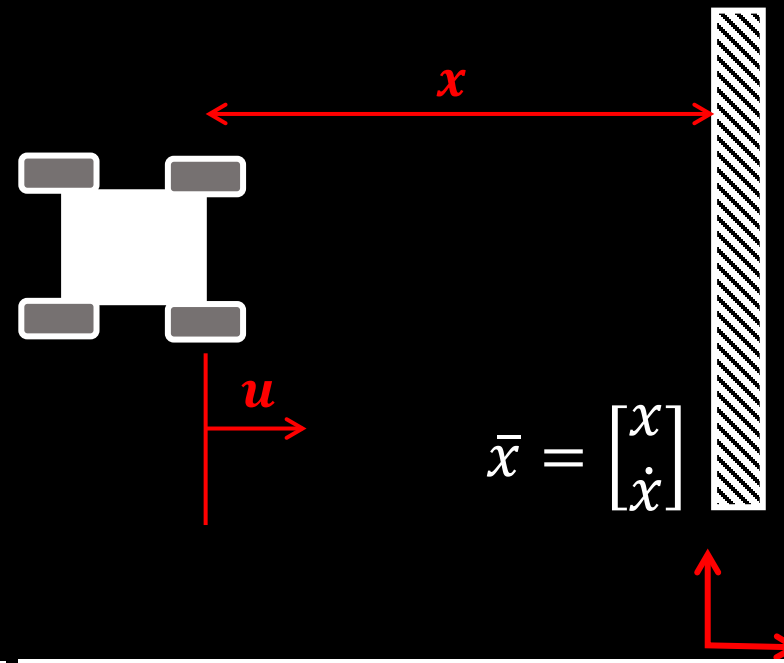
State space equation

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

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

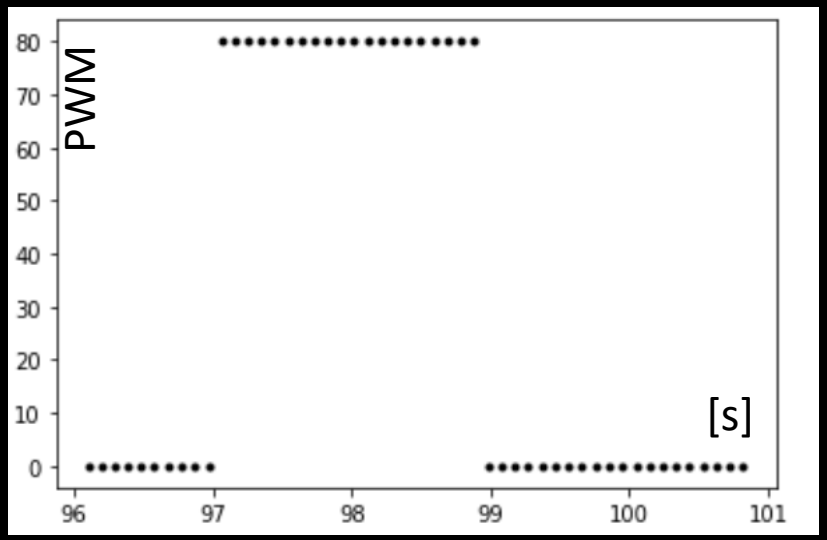
Lab 7, Task A: State Space Equations

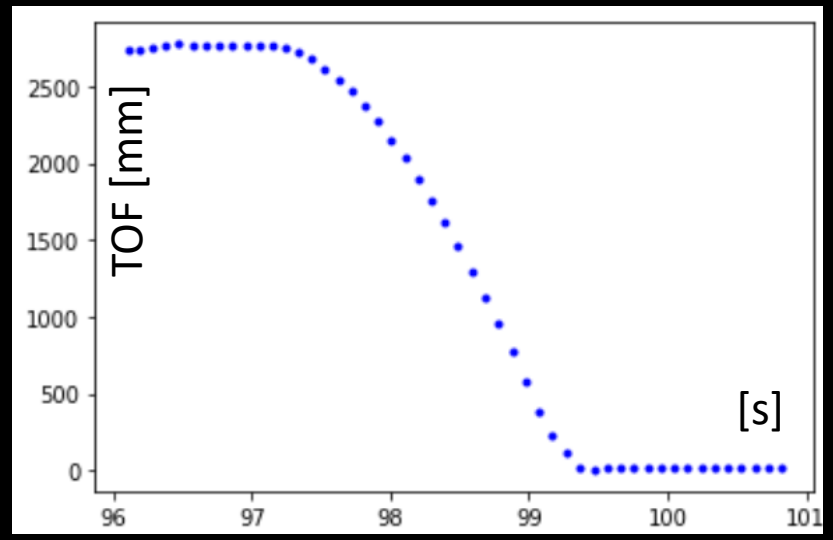
$$F = ma = m\ddot{x}$$
$$F = u - d\dot{x}$$
$$u - d\dot{x} = m\ddot{x}$$
$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

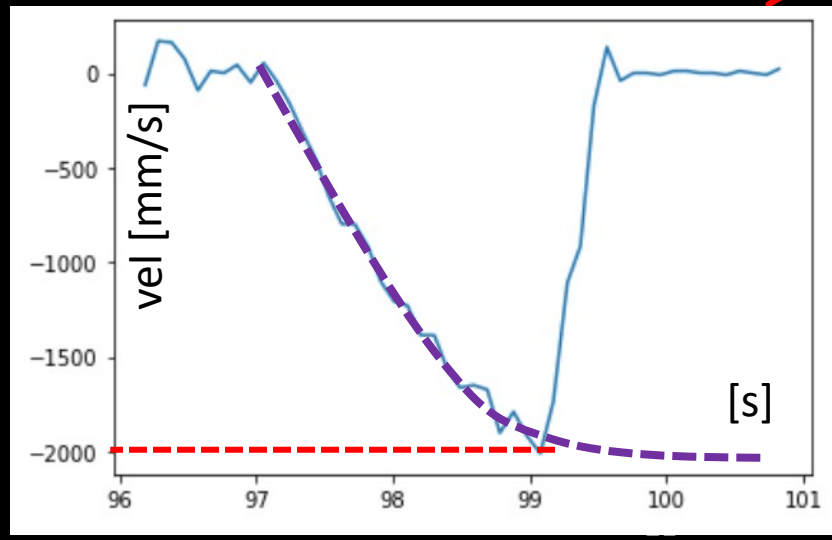


What is d and m?

- At steady state (cst speed), we can find *d*

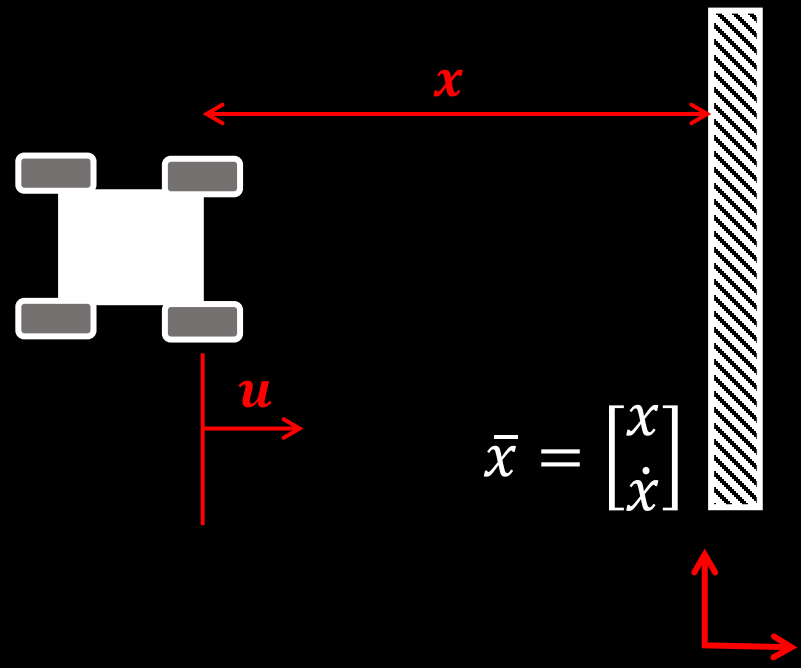






Lab 7, Task A: State Space Equations

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



What is d and m?

- At steady state (cst speed), we can find *d*
  - $0 = \frac{u}{m} - \frac{d}{m}\dot{x}$
  - $0 = \frac{u}{m} - \frac{d}{m}\dot{x} \quad \leftrightarrow \quad d = \frac{u}{\dot{x}}$
  - $d \approx \frac{1}{2000mm/s}$ *(Assume u=1 for now)*

State space equation

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

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

Lab 7, Task A: State Space Equations

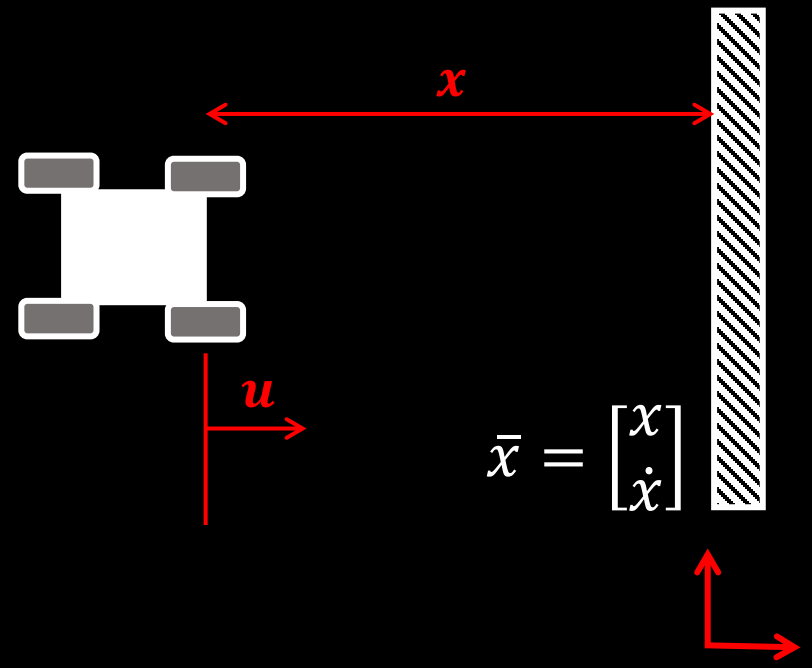
$$F = ma = m\ddot{x}$$
$$F = u - d\dot{x}$$
$$u - d\dot{x} = m\ddot{x}$$
$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

1st order system:

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

Unit step response solution:

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



What is d and m?

- Find m
- Use the 90% rise time to determine m
  - $\dot{v} = \frac{u}{m} - \frac{d}{m}v$
  - $v = 1 - e^{-\frac{d}{m}t_{0.9}} \leftrightarrow 1 - v = e^{-\frac{d}{m}t_{0.9}}$
  - $\ln(1 - v) = -\frac{d}{m}t_{0.9}$
  - $m = \frac{-dt_{0.9}}{\ln(1-0.9)}$

State space equation

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

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

Lab 7, Task A: State Space Equations

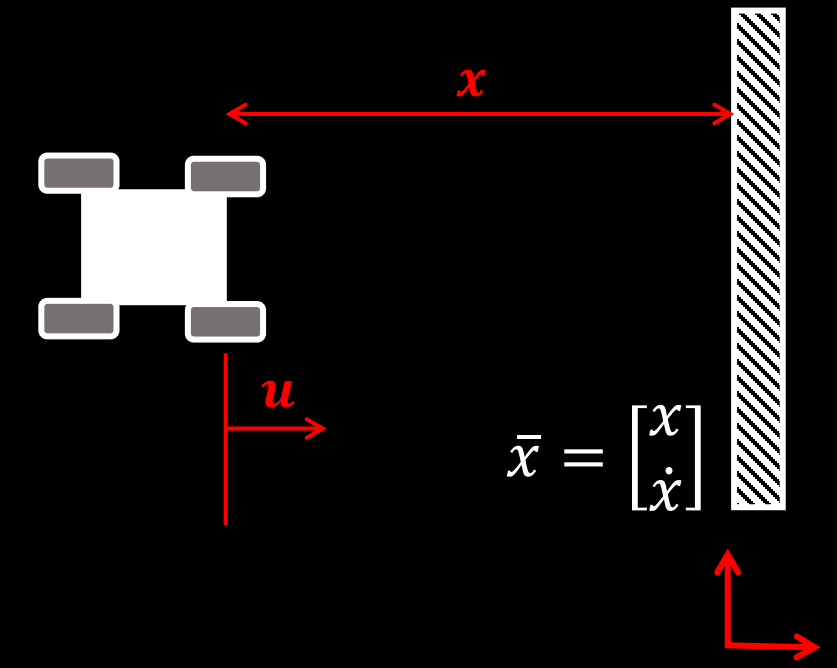
$$F = ma = m\ddot{x}$$
$$F = u - d\dot{x}$$
$$u - d\dot{x} = m\ddot{x}$$
$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

1st order system:

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

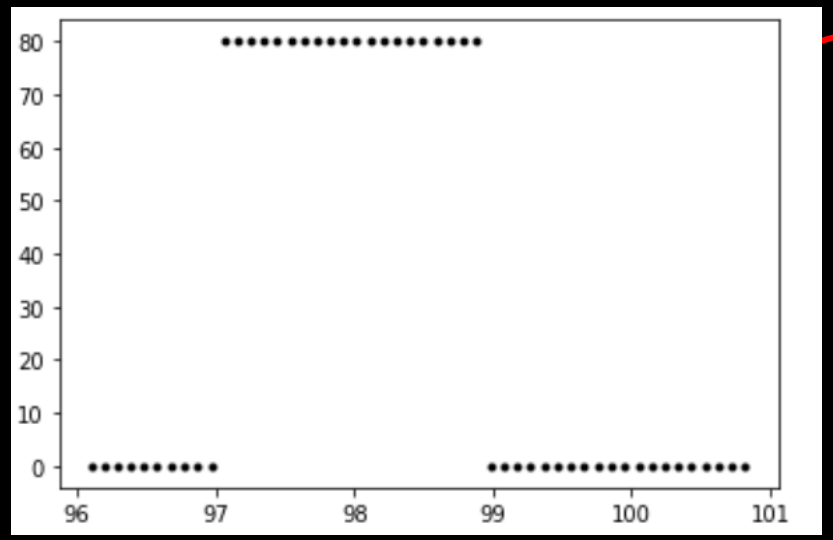
Unit step response solution:

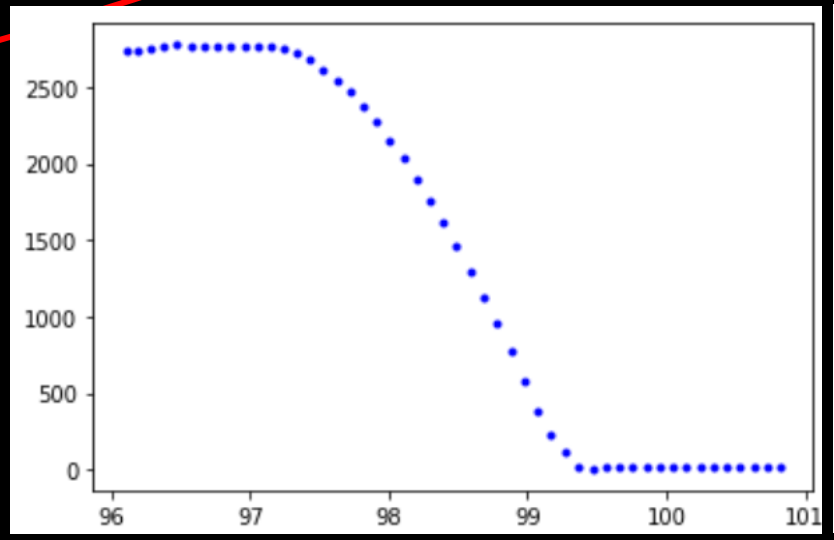
$$y(t) = 1 - e^{-\frac{t}{\tau}}$$

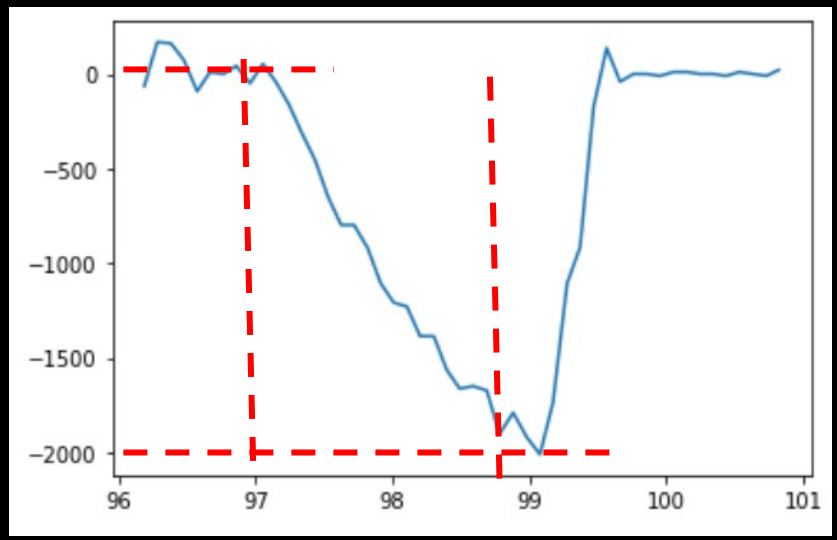


What is d and m?

- Find m
- Use the 90% rise time to determine m







Lab 7, Task A: State Space Equations

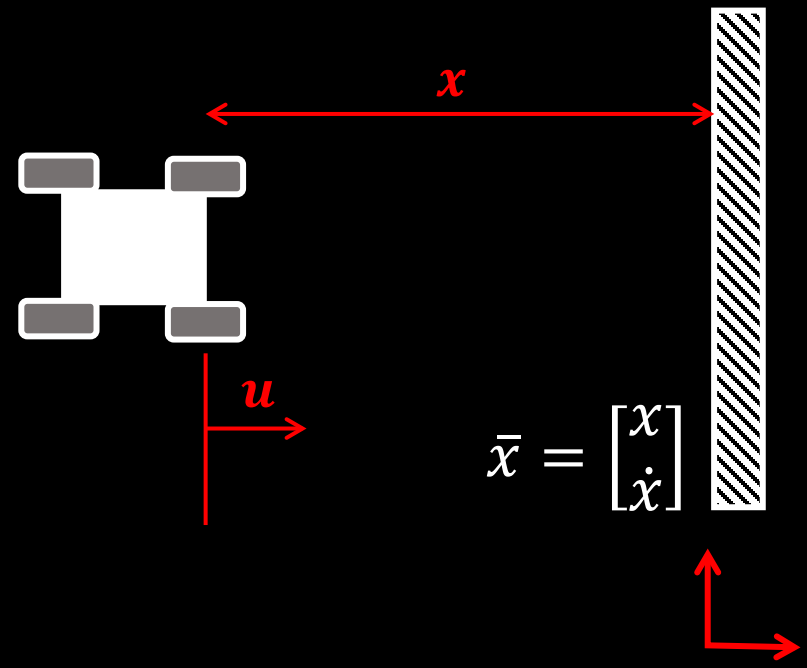
$$F = ma = m\ddot{x}$$
$$F = u - d\dot{x}$$
$$u - d\dot{x} = m\ddot{x}$$
$$\ddot{x} = \frac{u}{m} - \frac{d}{m}\dot{x}$$

1st order system:

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

Unit step response solution:

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



What is d and m?

- Use the 90% rise time to find m
  - $\dot{v} = \frac{u}{m} - \frac{d}{m}v$
  - $v = 1 - e^{-\frac{d}{m}t_{0.9}} \leftrightarrow 1 - v = e^{-\frac{d}{m}t_{0.9}}$
  - $\ln(1 - v) = -\frac{d}{m}t_{0.9}$
  - $m = \frac{-dt_{0.9}}{\ln(1-0.9)} = \frac{-0.0005 \cdot 1.9}{\ln(0.1)} = 4.1258 \cdot 10^{-4}$

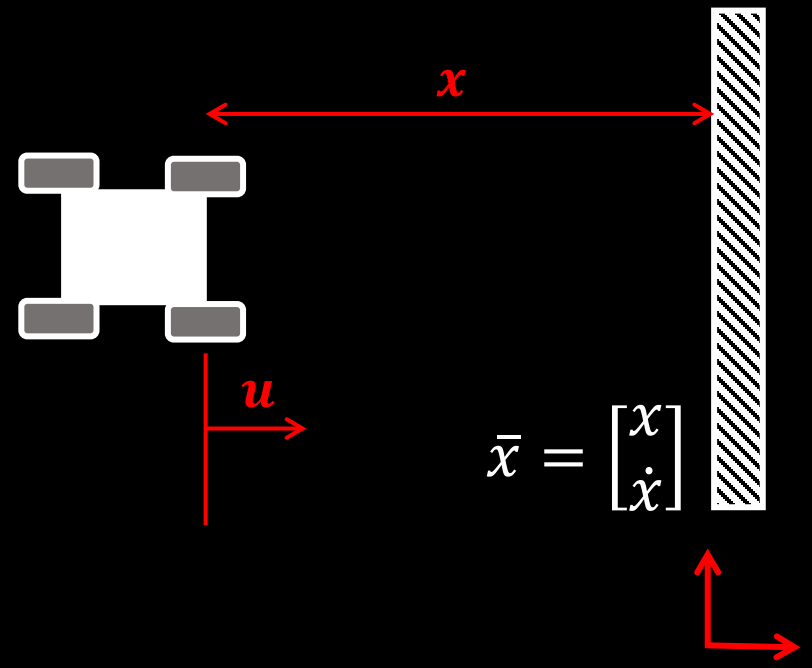
State space equation

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

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

Lab 7, Task A: State Space Equations

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



What is d and m?

- At steady state (cst speed), we can find *d*
  - $d = \frac{u}{\dot{x}} \approx 0.0005$ *(Assume u=1 for now)*
- We can use the 90% rise time to find *m*
  - $m = \frac{-dt_{0.9}}{\ln(0.1)} \approx 4.1258 \cdot 10^{-4}$

State space equation

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

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

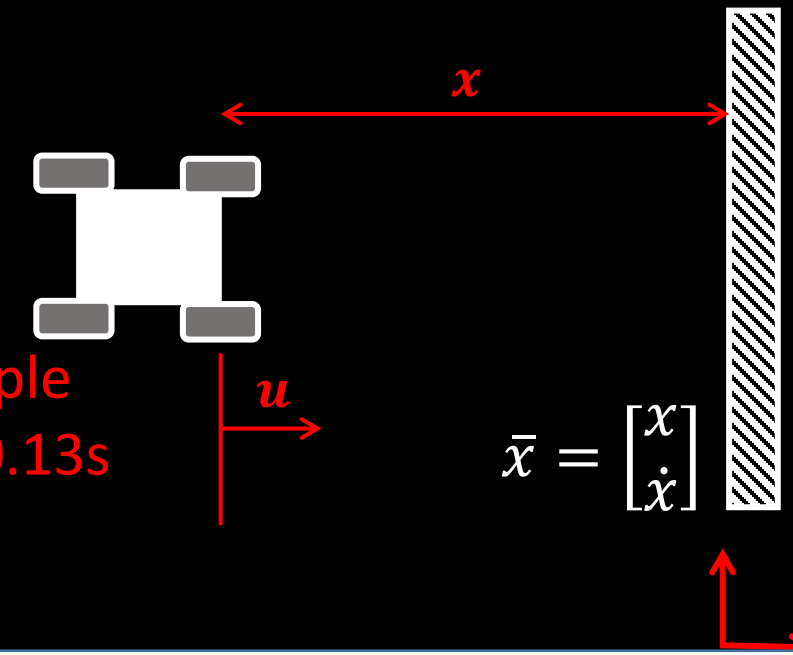
Lab 7, Task A: State Space Equations

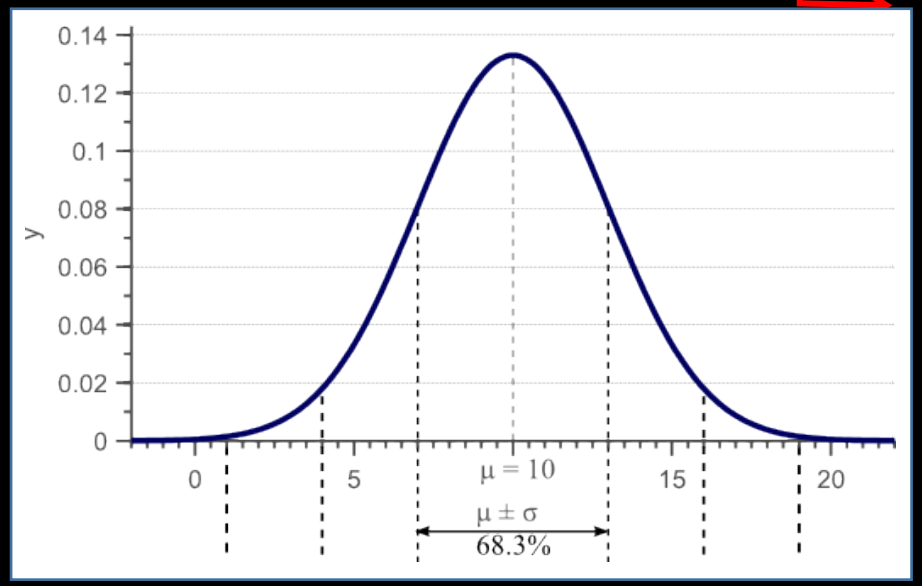
Implement the Kalman Filter

- Process noise (dependent on sampling rate)

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

  - Trust in modeled position:
    - Position std after 1s: $\sqrt{10^2 \cdot \frac{1}{0.13}} = 27.7mm$
  - Trust in modeled speed:
    - Speed std after 1s: $\sqrt{10^2 \cdot \frac{1}{0.13}} = 27.7mm/s$
- Measurement noise
  - $\Sigma_z = [{\sigma_3}^2]$
  - $\sigma_3^2 = (20mm)^2$





Lab 7, Task A: State Space Equations

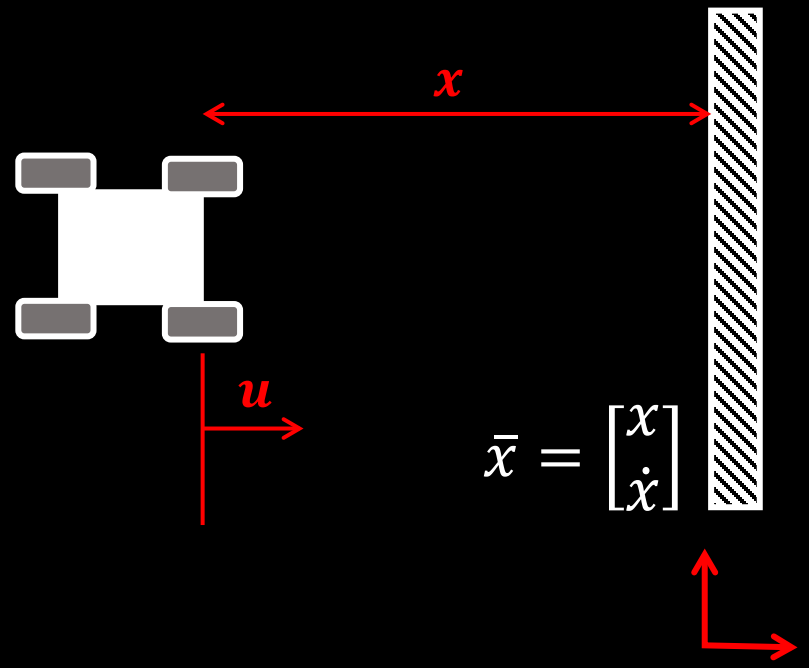
- We have A, B, C, Σ_u, Σ_z

- Discretize the A and B matrices
  - x(n+1) = x(n) + dx
  - dx = dt (Ax + Bu)
  - x(n+1) = x(n) + dt (Ax(n) + Bu)
  - x(n+1) = (I + dt*A) x(n) + dt*B u

A_d B_d

  - dt is our sampling time (0.130s)

- Rescale from unity input to actual input

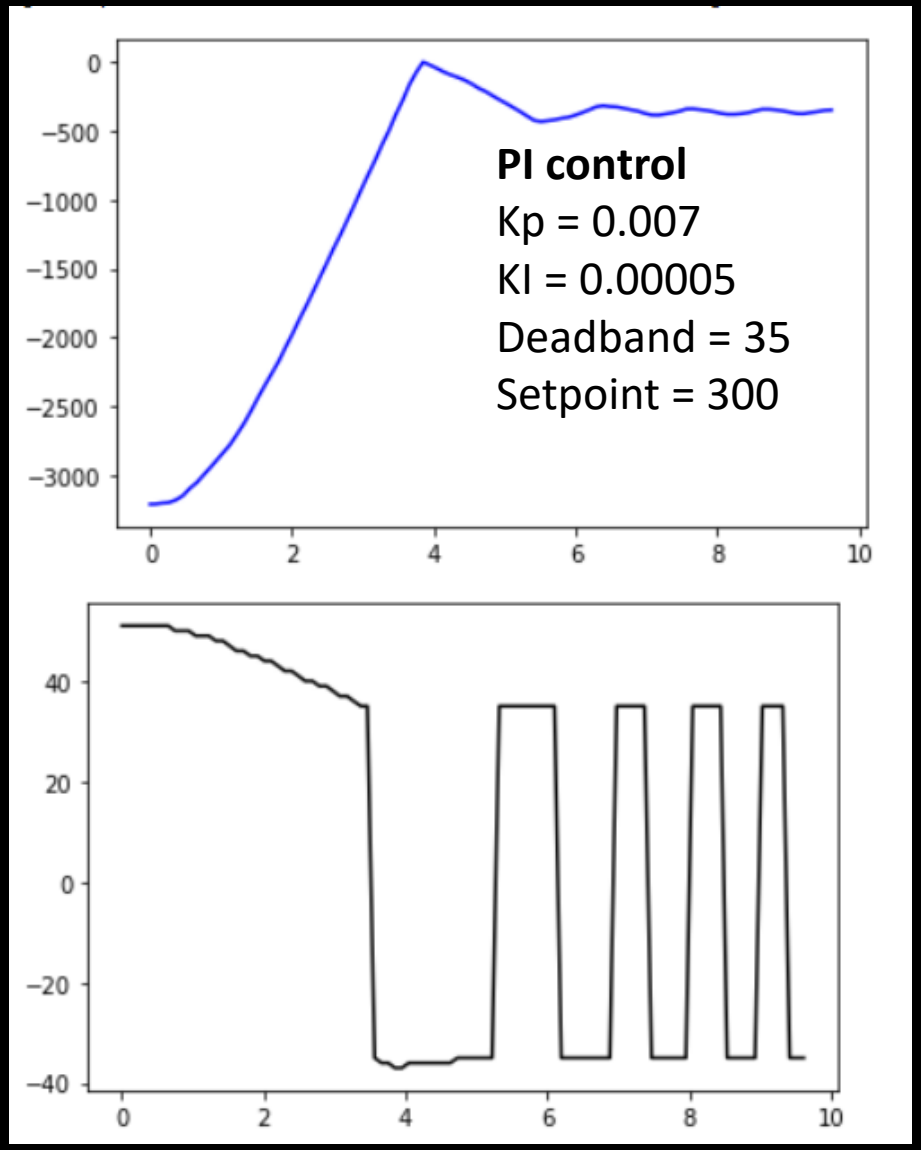


State space equation

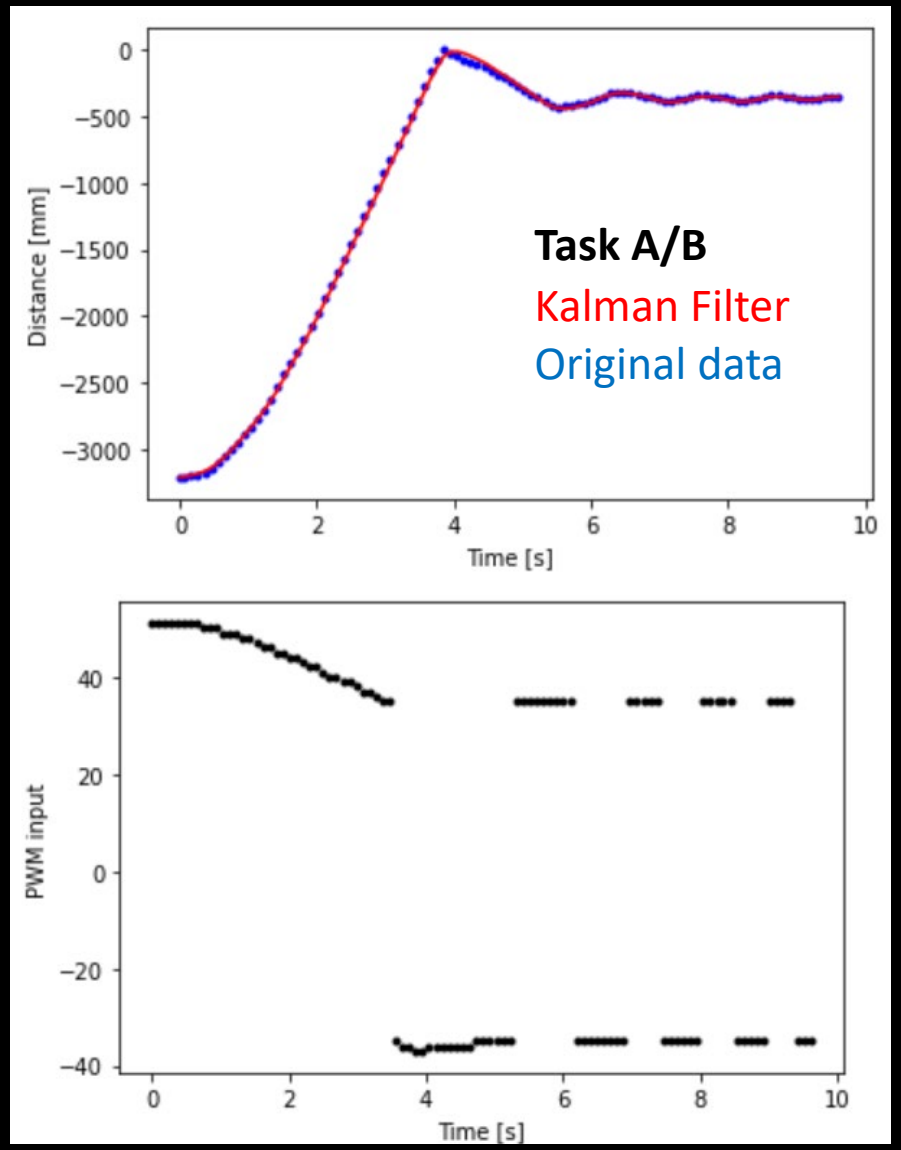
$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{d}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u$$

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

Lab 7, Task A: Kalman Filter



With Kalman filter



ECE4960-2022

Course on "Fast Robots", offered Spring 2022 in the ECE dept at Cornell University

View On GitHub

This project is maintained by CEI-lab

Cornell University: ECE 4960 / 5960

Return to main page

Lab 7: Kalman Filter

Objective

The objective of Lab 7 is to implement a Kalman Filter, which will help you execute the behavior you did in Lab 6 fast, such that you can complete stunts in Lab 8. Remember to stick to the Task (A/B/C) you completed in Lab 6.

Parts Required

- 1 x Fully assembled robot, with Artemis, batteries, TOF sensor, and IMU.
- If you intend to do most of your testing at home, you'll need to find an obstacle free path (at least 2m), and a bright pillow or similar to mitigate a potential crash. Remember, you will have to eventually re-calibrate everything to the lab floor where we will perform the stunts.

Lab Procedure

1. Step Response

Recall Lectures 11 and 12. To implement a Kalman Filter, you will need to estimate the terms in your A and B matrices using a step response.

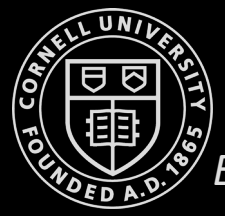
- If you chose Task A or B, you will now use sensor fusion to help you estimate the distance to the wall frequently, inspite of the slow sampling time of the ToF sensor.

Lab 6-8: PID control – Sensor Fusion - Stunt

- Task A: Don’t Hit the Wall!
- Task B: Drift much?
- Task C: Thread the Needle!

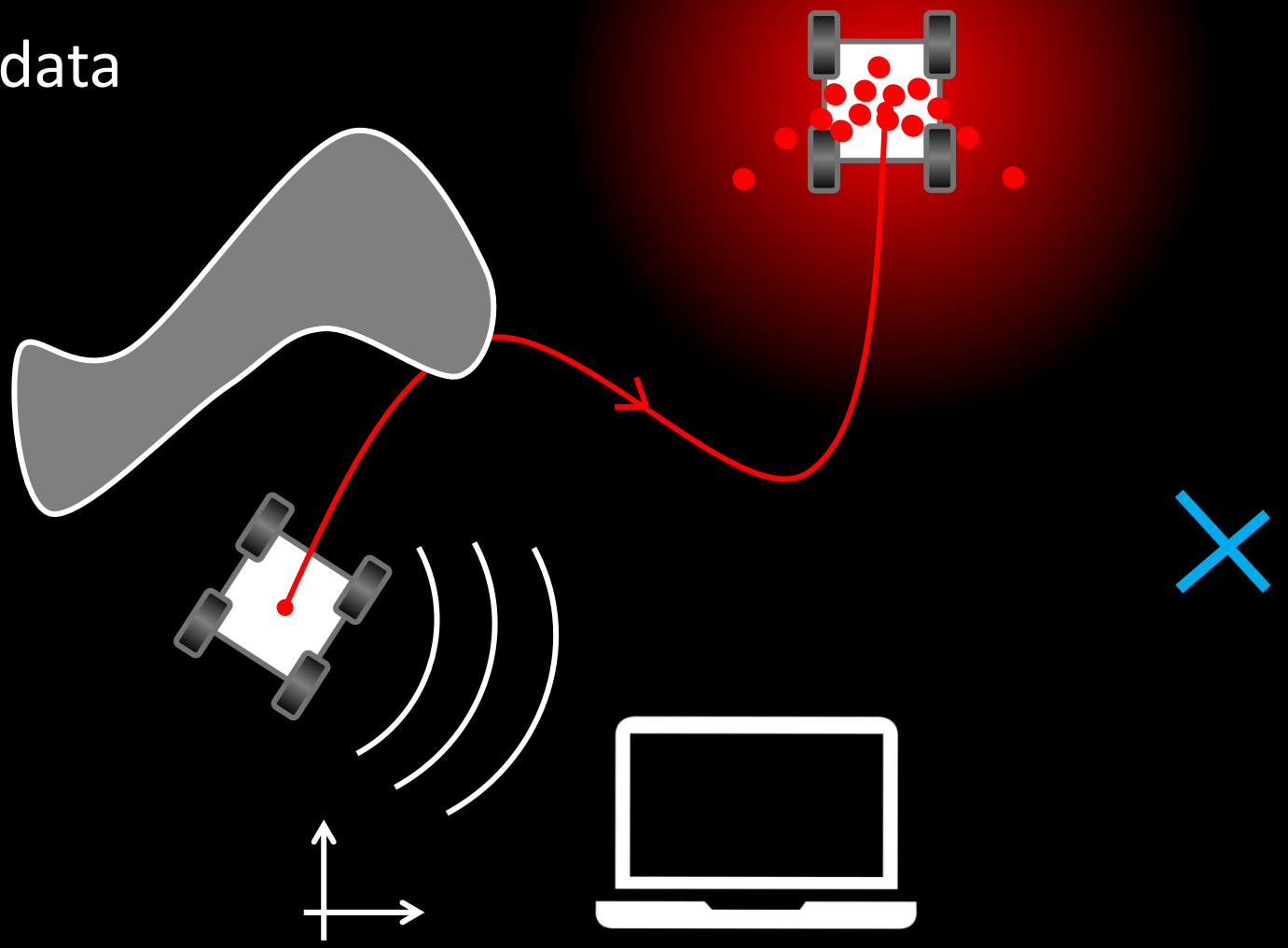
Procedure

- Lab 6: Get basic PID to work, consider sampling time, start slow
- Lab 7: Sensor Fusion
  - Approximate the state space equations
    - Step response
  - Implement Kalman Filter
    - Determine process and measurement noise
    - Try it offline on solution from lab 6
    - Try it online on your robot
- Lab 8: Use KF and PID control to execute stunt
    - **You’re welcome to try an LQG (LQR and KF) controller!*



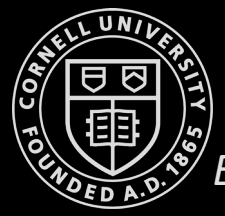
What we covered in class so far…

- Transformation matrices
- Bluetooth communication and data types
- Distance Sensors
- Odometry and IMU
- Actuators
- Controllers
  - PID control
  - LQR
- Observers
- Navigation
  - Deterministic → Probabilistic robots



Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

Navigation

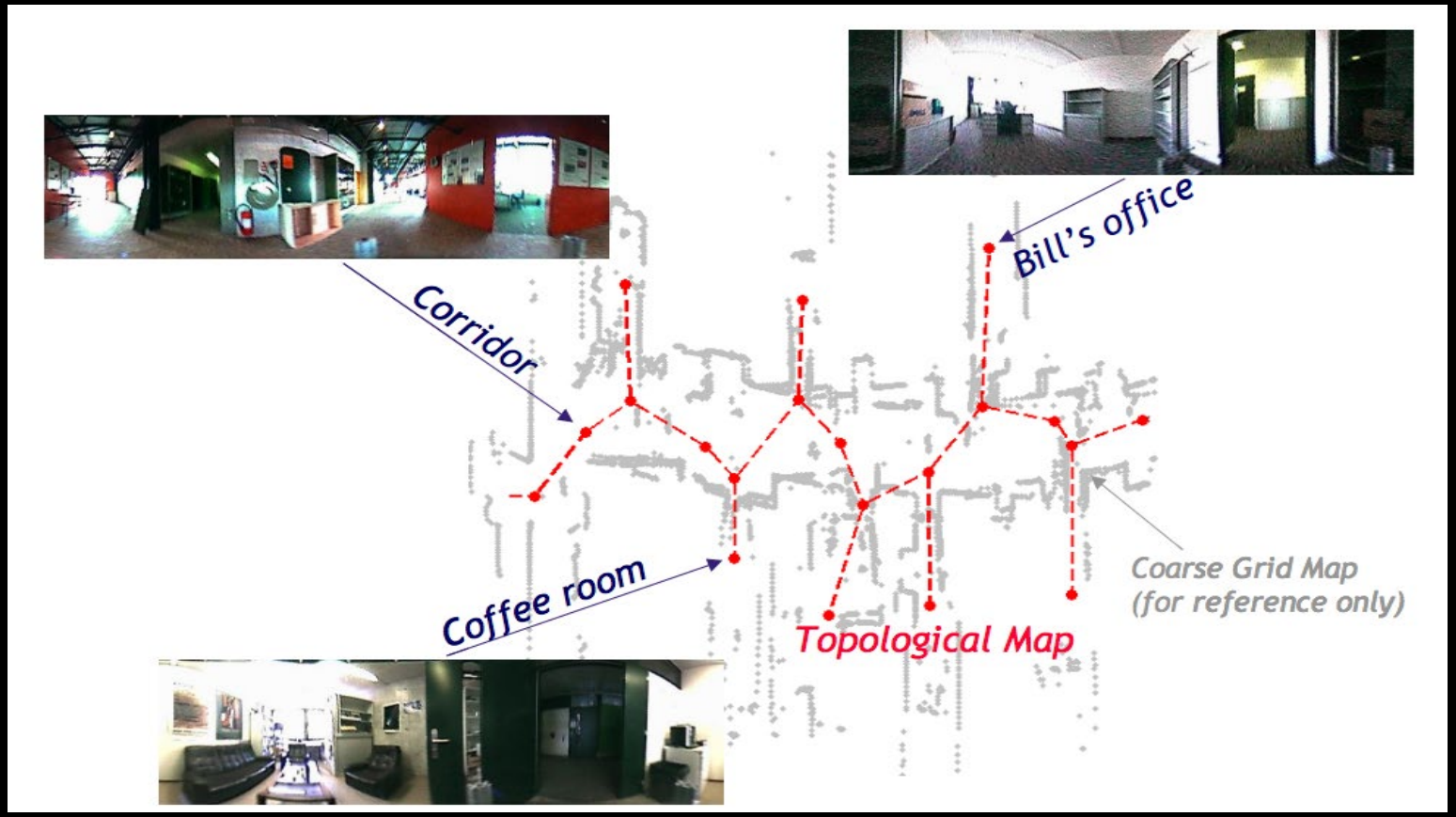


Navigation and Path Planning

- How do you get to your goal?
- No simple answers…
  - 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 computing power do you have?
  - How precise and accurate is your motion control?

Navigation and Path Planning

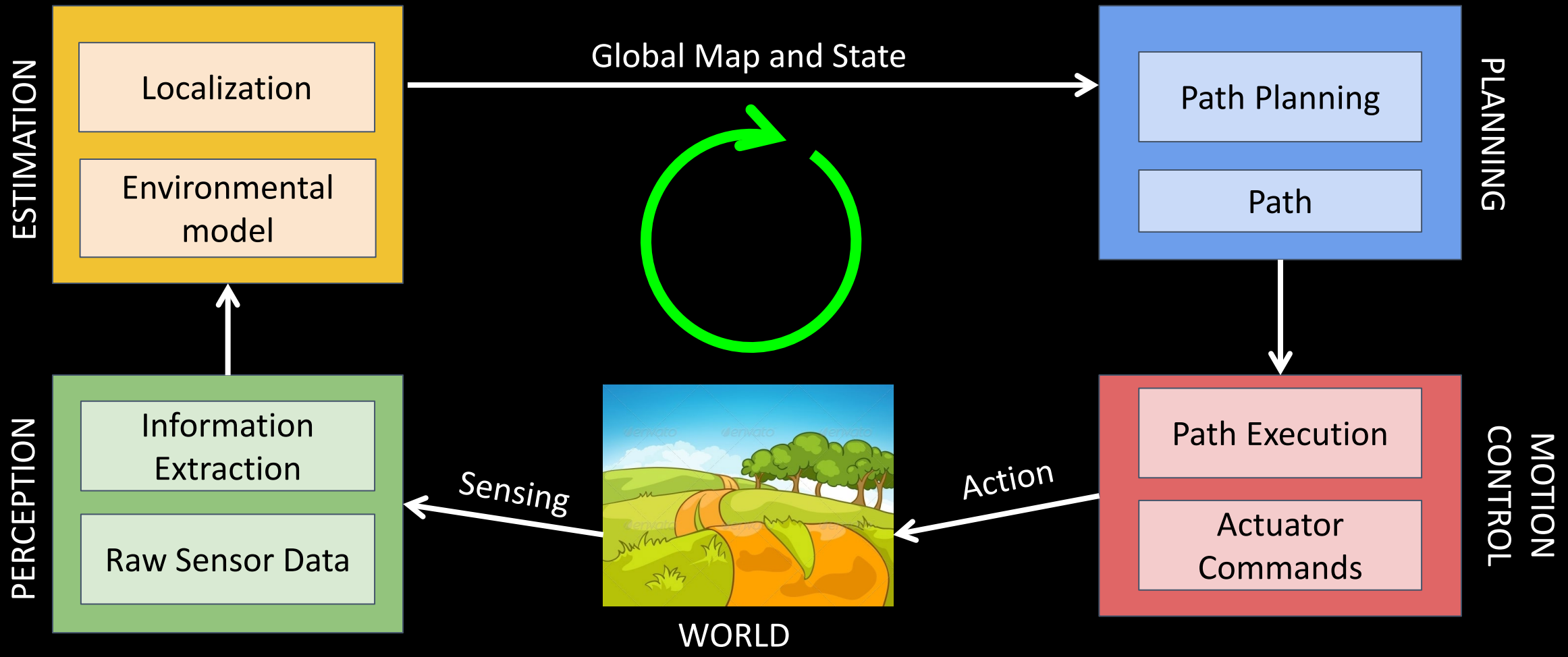
- **Problem:** Find the path in the workspace from an initial location to a goal location, while avoiding collisions
- **Assumption:** There exists a good map of the environment for navigation



- Global navigation
  - Given a map and a goal location, find and execute a trajectory that brings the robot to the goal
  - (Long term plan)
- Local navigation
  - Given real-time sensor readings, modulate the robot trajectory to avoid collisions
  - (Short term plan)

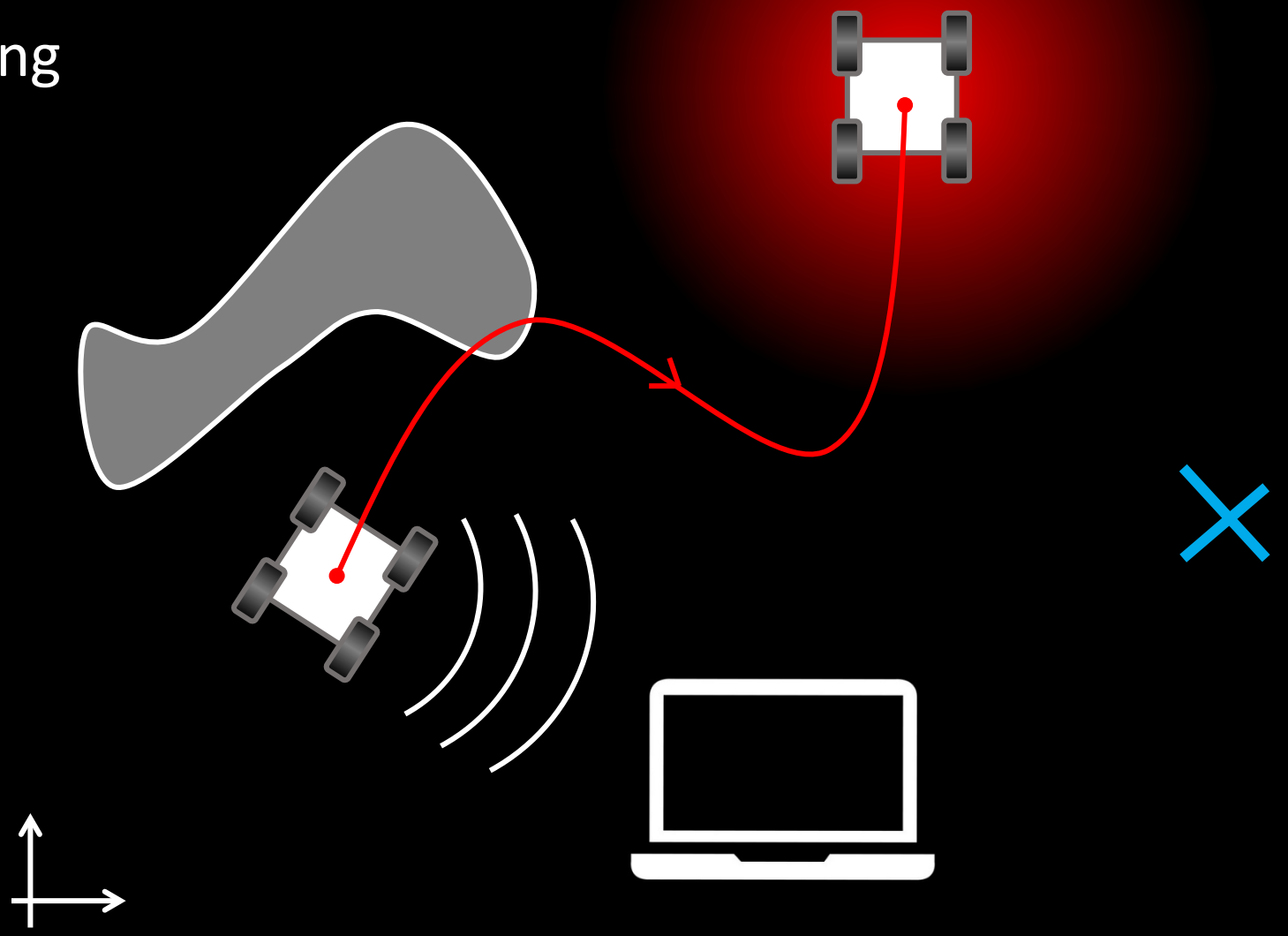
Navigation and Path Planning

- Navigation breaks down to: Localization, Map Building, Path Planning



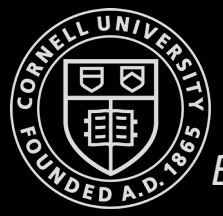
Outline of the next module on Navigation

- 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*



Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

Local Planners



Local Path Planning / Obstacle Avoidance

- Utilize goal position, recent sensor readings, and relative position of robot to goal
  - Implemented as a separate task most of the times
  - Runs at a much faster rate than the global planning

- BUG Algorithms
- Vector Field Histogram (VFH)
- Dynamic Window Approach (DWA)

Wagner, ITS 2015

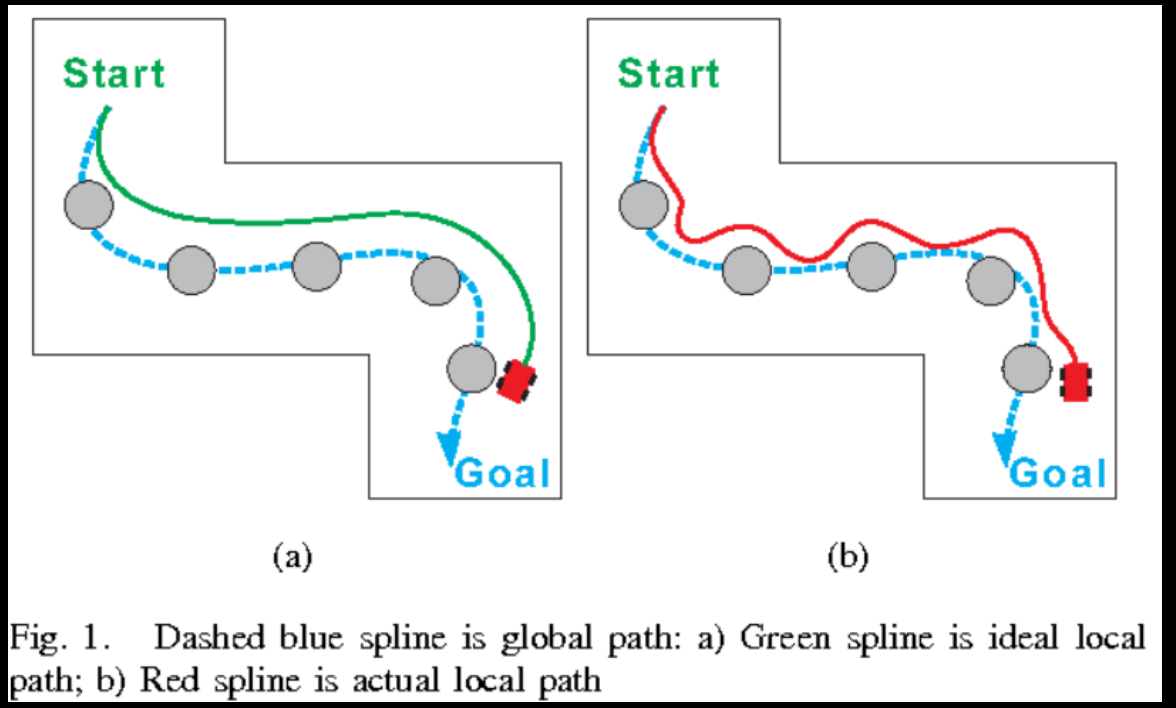
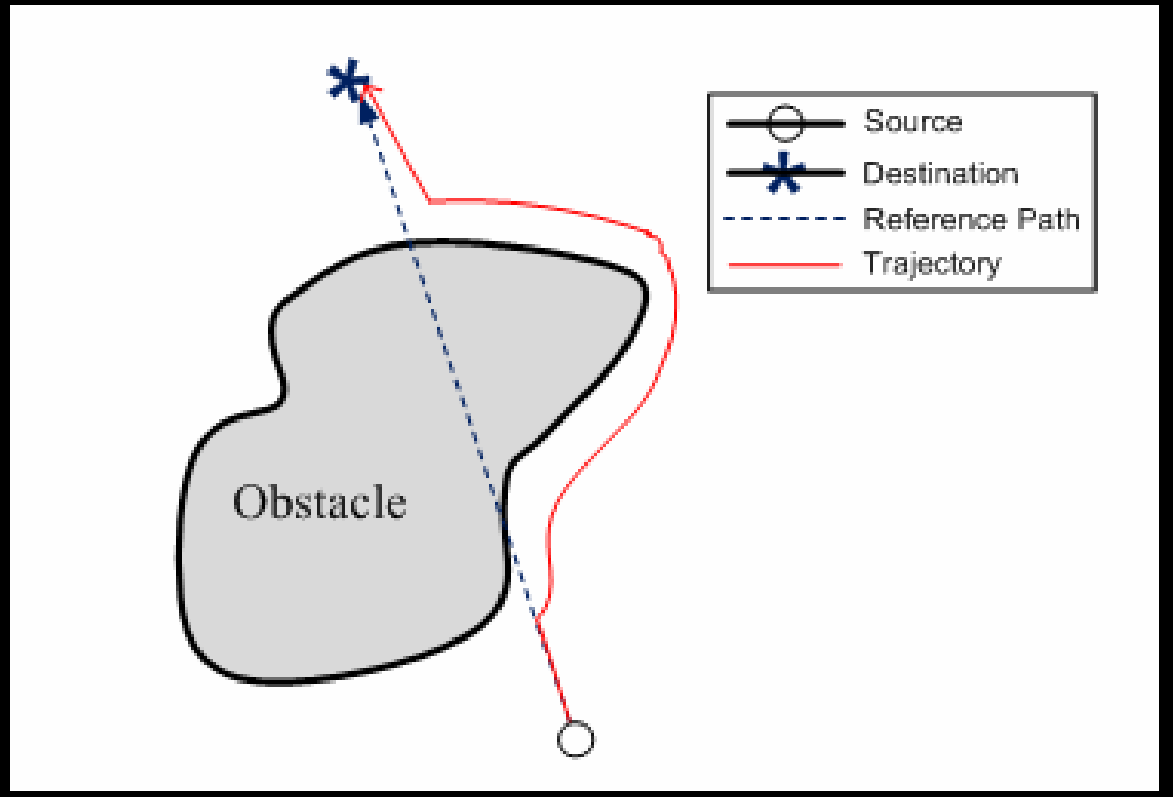


Fig. 1. Dashed blue spline is global path: a) Green spline is ideal local path; b) Red spline is actual local path

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



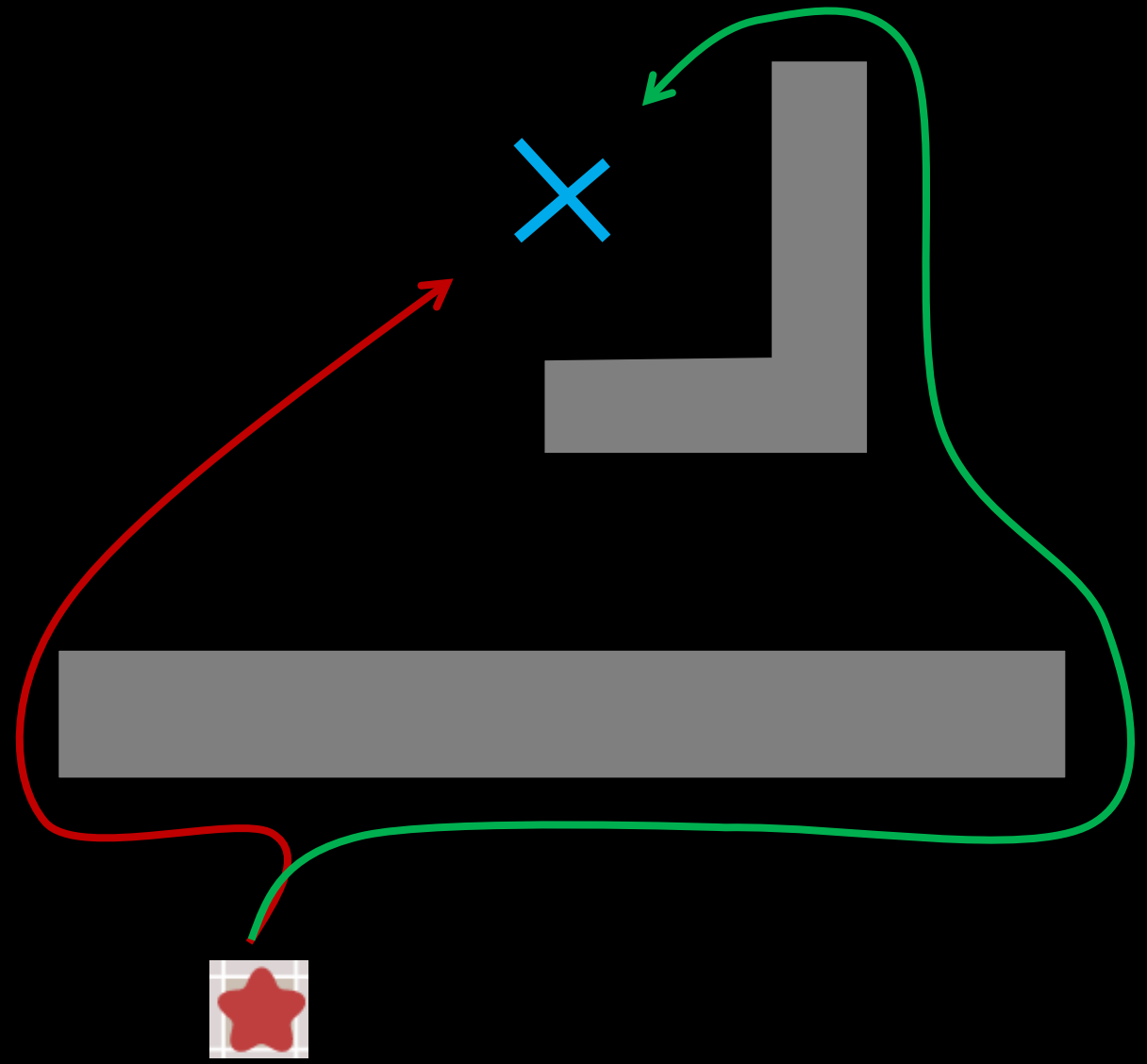
Bug 0

Sensor Assumptions

- Direction to the goal
- Detect walls

Algorithm

1. Go towards goal
2. Follow obstacles until you can go towards goal again
3. Loop



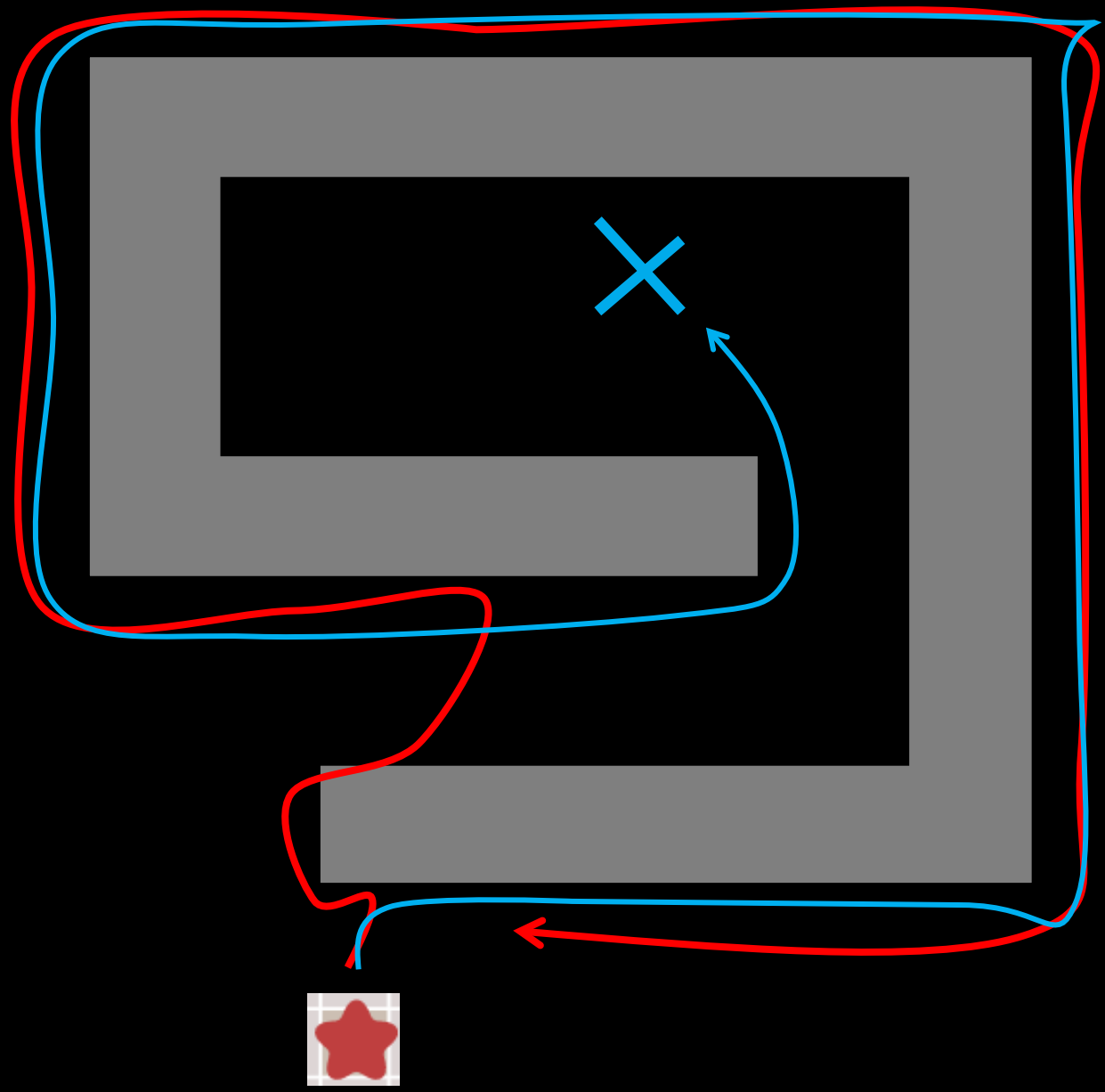
Bug 0

Sensor Assumptions

- Direction to the goal
- Detect walls

Algorithm

1. Go towards goal
2. Follow obstacles until you can go towards goal again
3. Loop



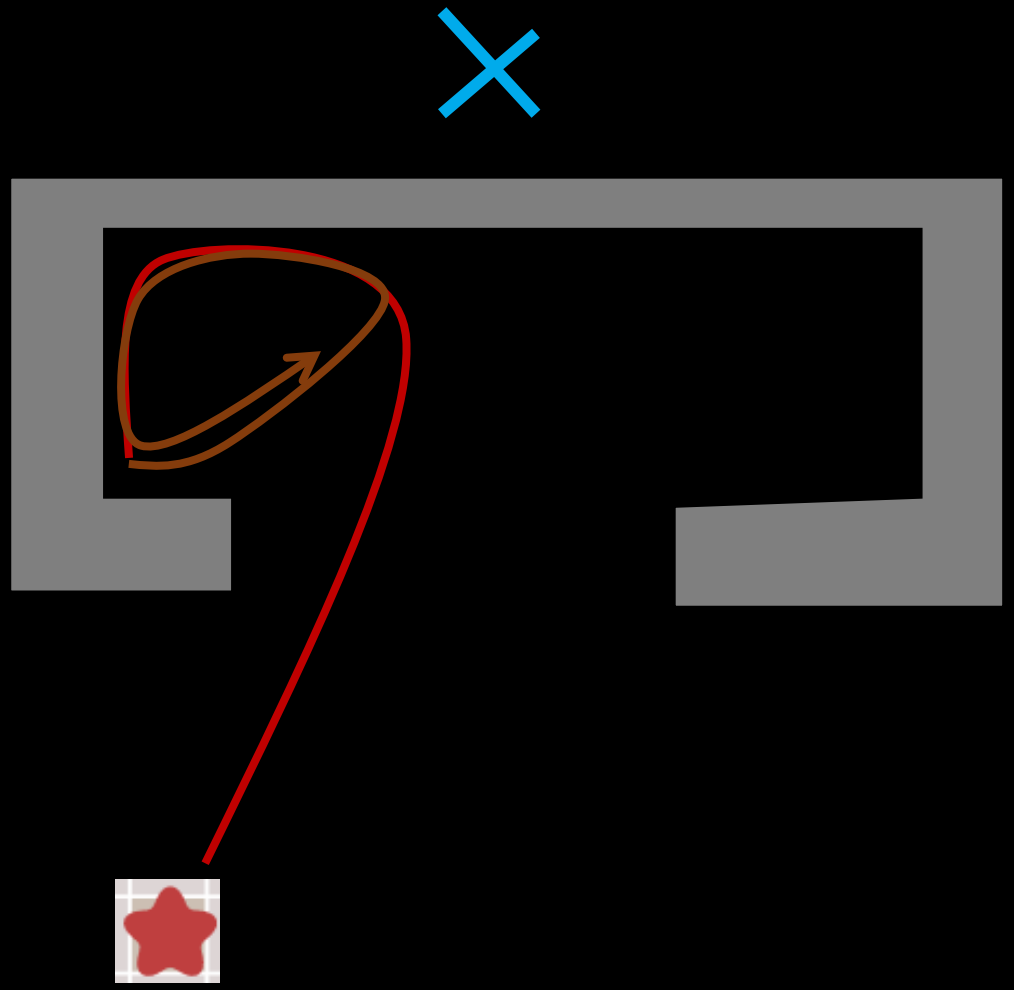
Bug 0

Sensor Assumptions

- Direction to the goal
- Detect walls

Algorithm

1. Go towards goal
2. Follow obstacles until you can go towards goal again
3. Loop



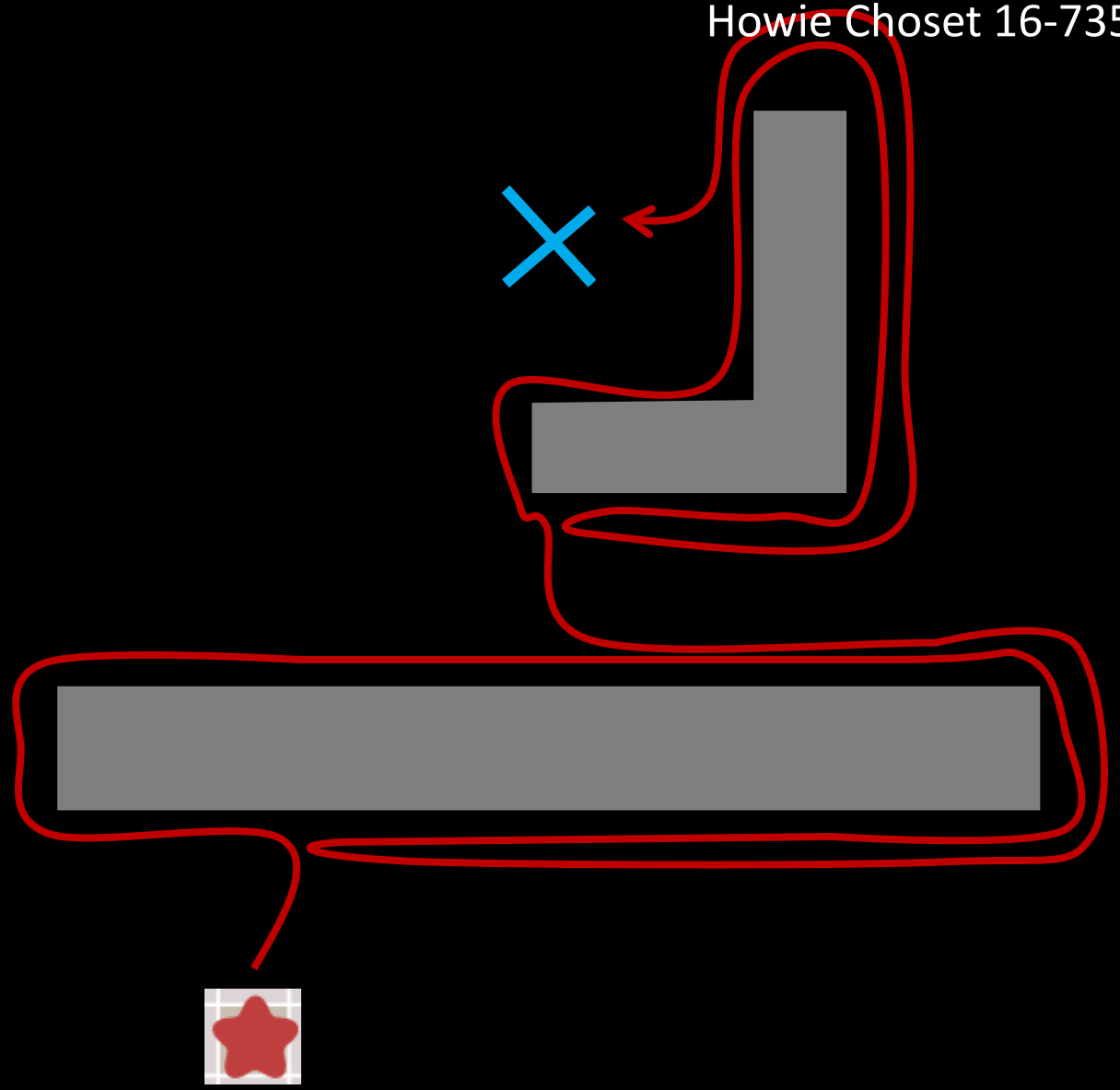
Bug 1

Sensor Assumptions

- Direction to the goal
- Detect walls
- Odometry

Algorithm

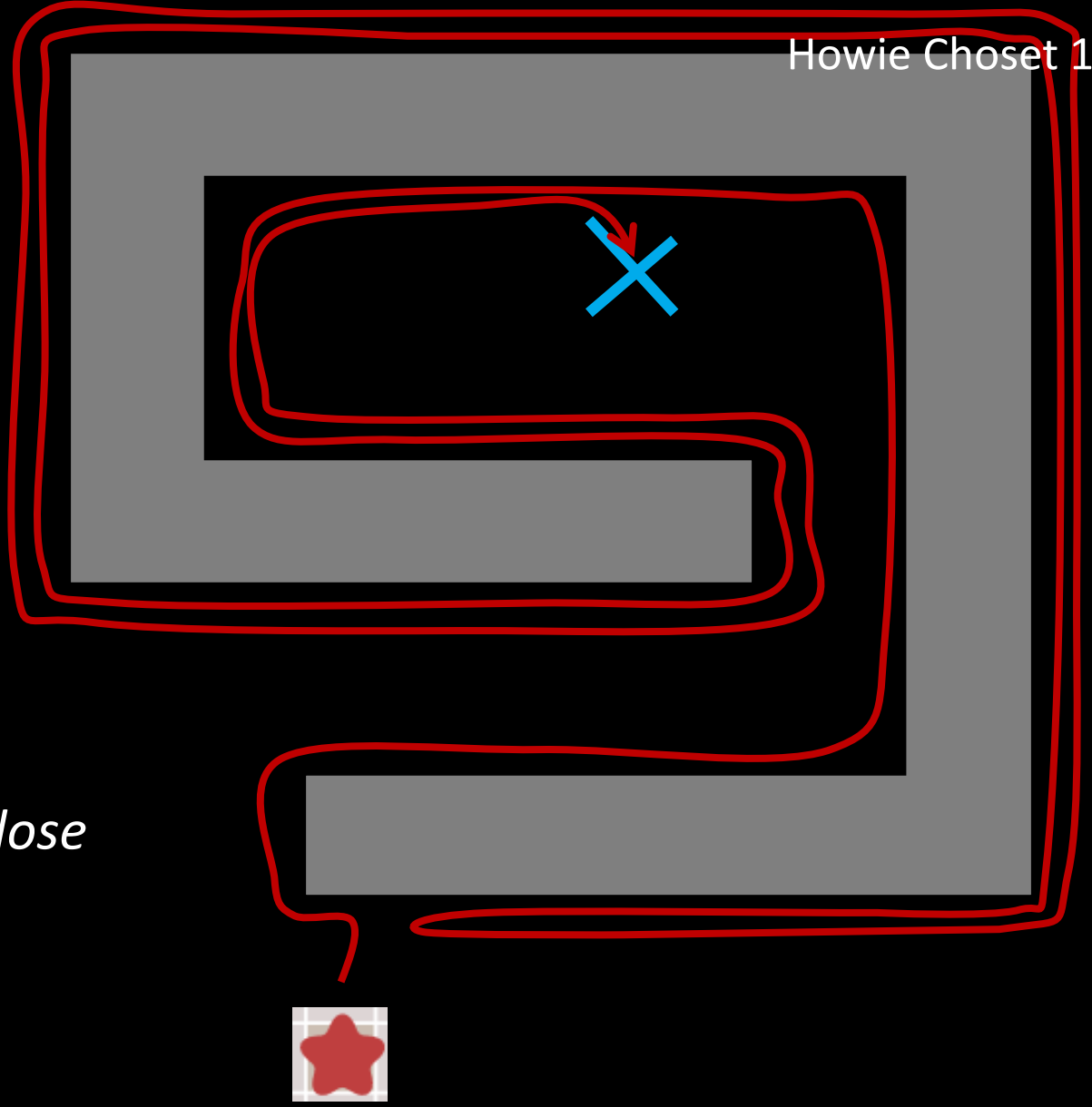
1. Go towards goal
2. Follow obstacles *and remember how close you got to the goal*
3. Return to the closest point, and loop



Bug 1

Sensor Assumptions

- Direction to the goal
- Detect walls
- Odometry



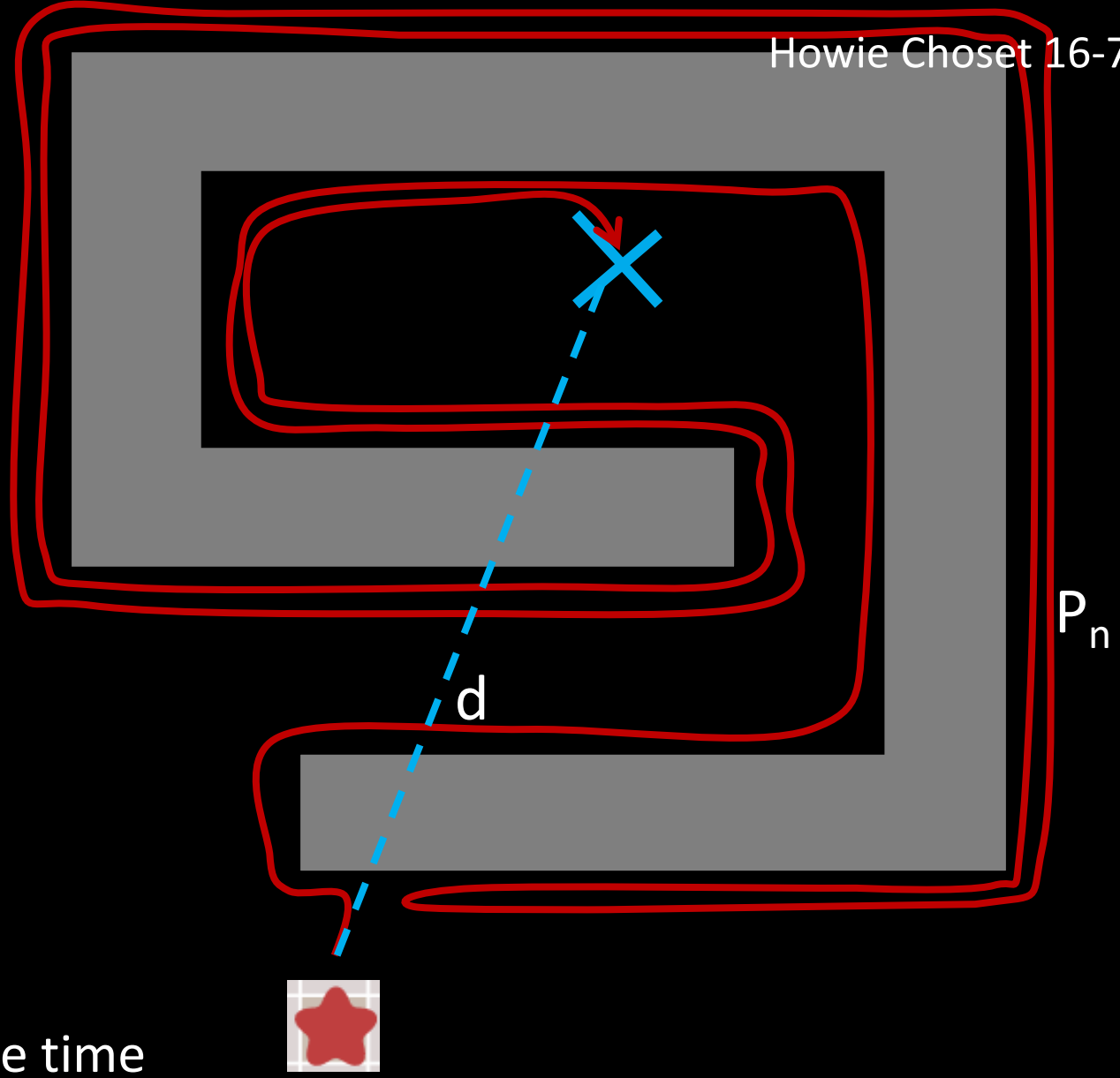
Algorithm

1. Go towards goal
2. Follow obstacles *and remember how close you got to the goal*
3. Return to the closest point, and loop

Bug 1 - formally

Sensor Assumptions

- Direction to the goal
- Detect walls
- Odometry



- ***Lower bound traversal?***
  - d
- ***Upper bound traversal?***
  - $d + 1.5 \cdot \text{Sum}(P_n)$
- ***Pros?***
  - If a path exist, it returns one in finite time
  - AND it knows if *none* exist!

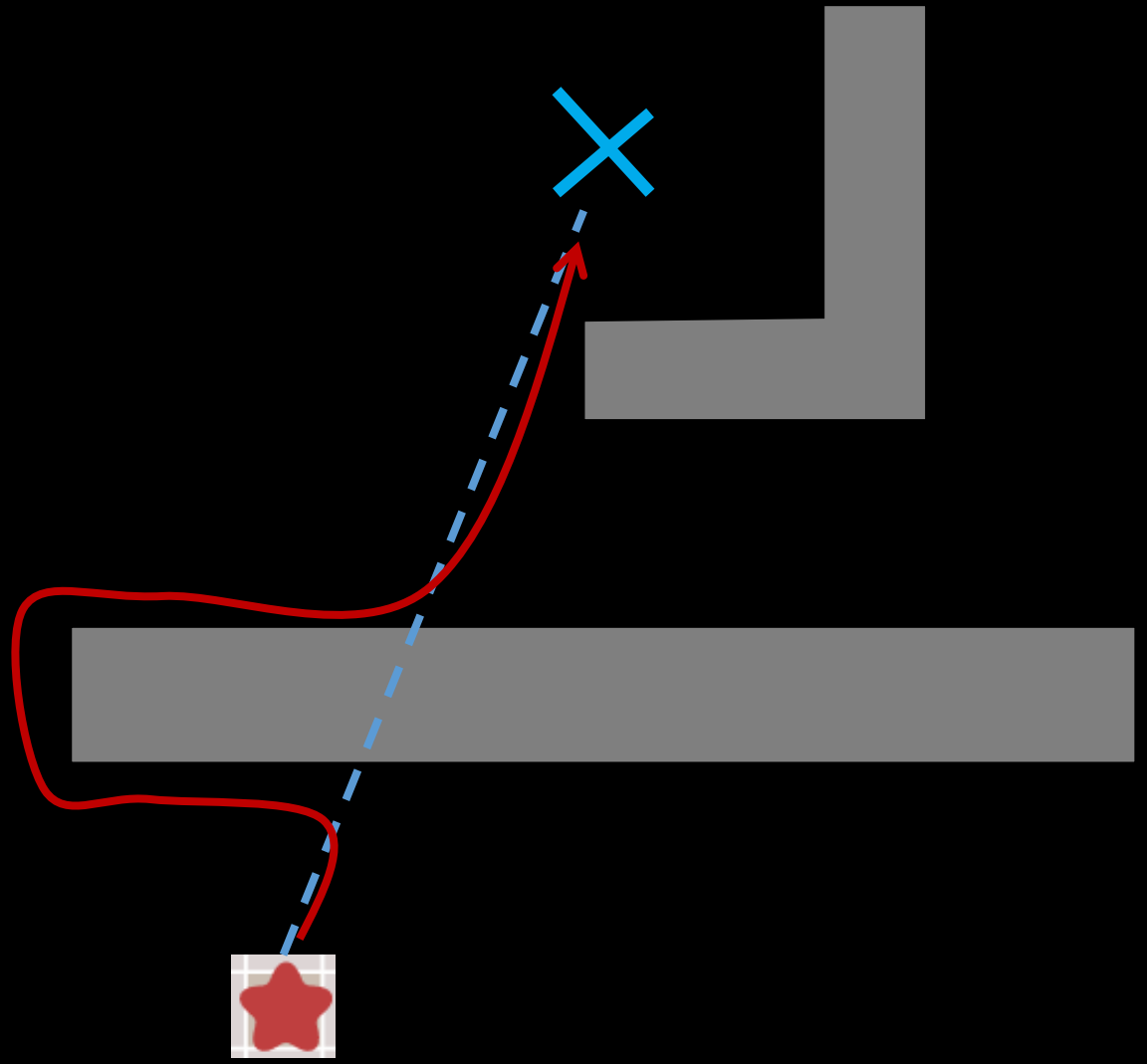
Bug 2

Sensor Assumptions

- Direction to the goal
- Detect walls
- Odometry
- Original vector to the goal

Algorithm

1. Go towards goal on the vector
2. Follow obstacles *until you are back on the vector (and closer to the obstacle)*
3. Loop



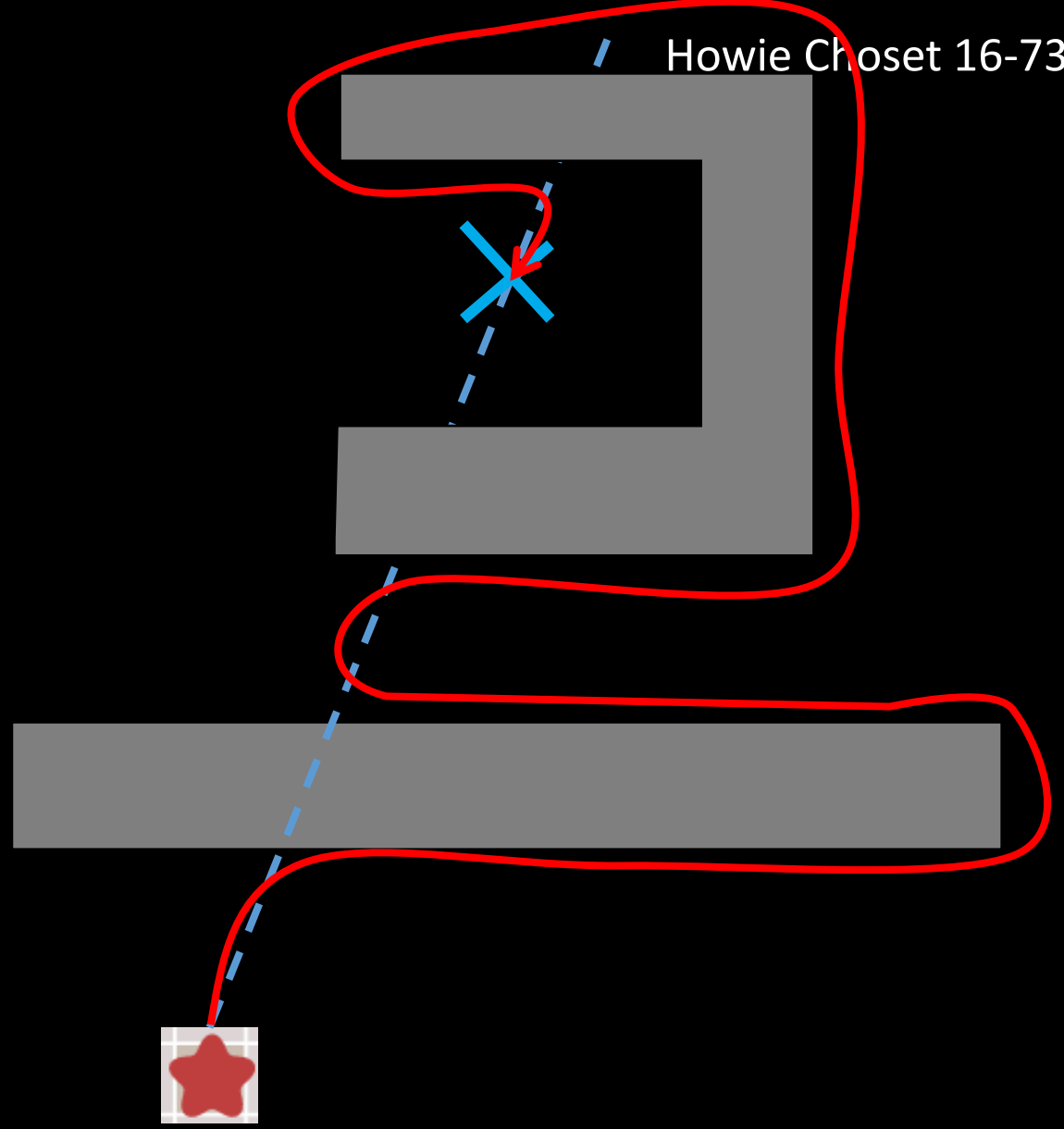
Bug 2

Sensor Assumptions

- Direction to the goal
- Detect walls
- Odometry
- Original vector to the goal

Algorithm

1. Go towards goal on the vector
2. Follow obstacles *until you are back on the vector (and closer to the obstacle)*
3. Loop



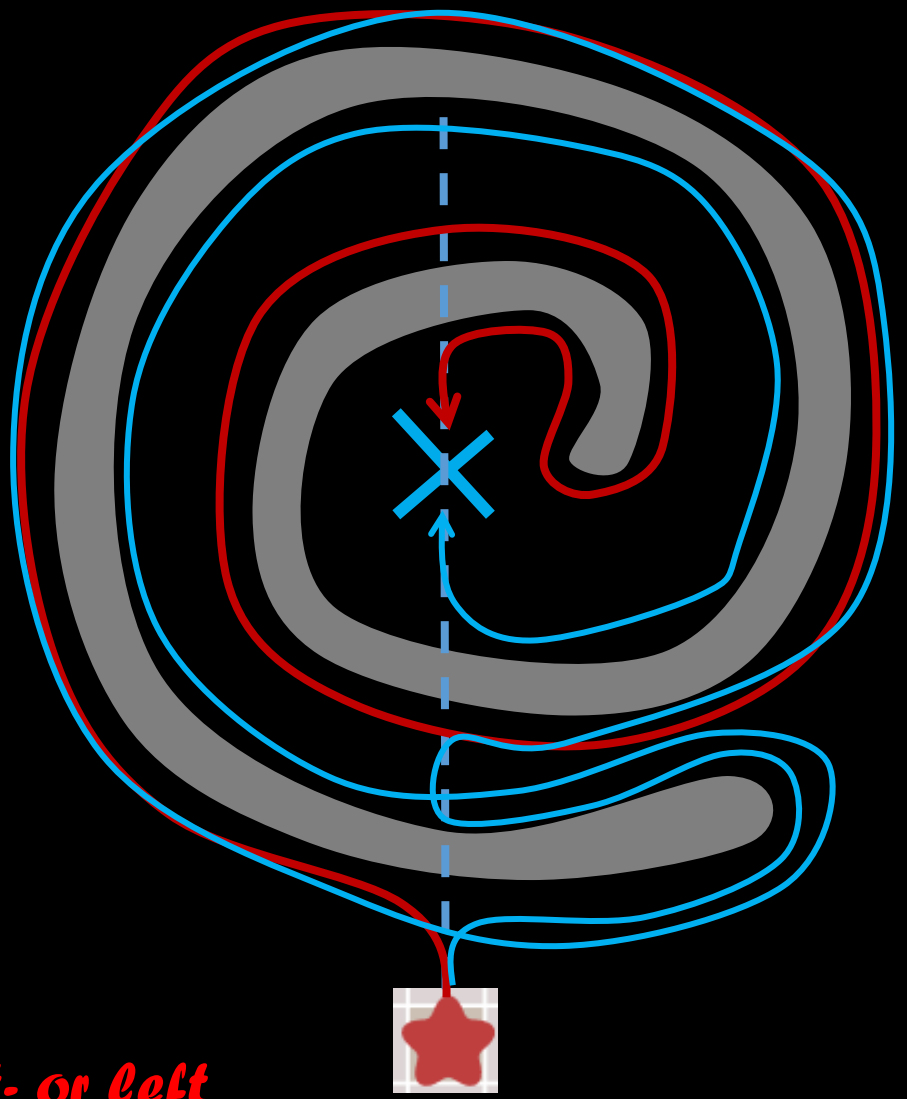
Bug 2

Sensor Assumptions

- Direction to the goal
- Detect walls
- Odometry
- Original vector to the goal

Algorithm

1. Go towards goal on the vector
2. Follow obstacles *until you are back on the vector (and closer to the obstacle)*
3. Loop



What is faster, right- or left wall following?

Battle of the Bugs (1 vs 2)

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

Bug 1
Layout 1

Bug 2
Layout 1

Battle of the Bugs (1 vs 2)

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

Exhaustive Search

Bug 1
Layout 2

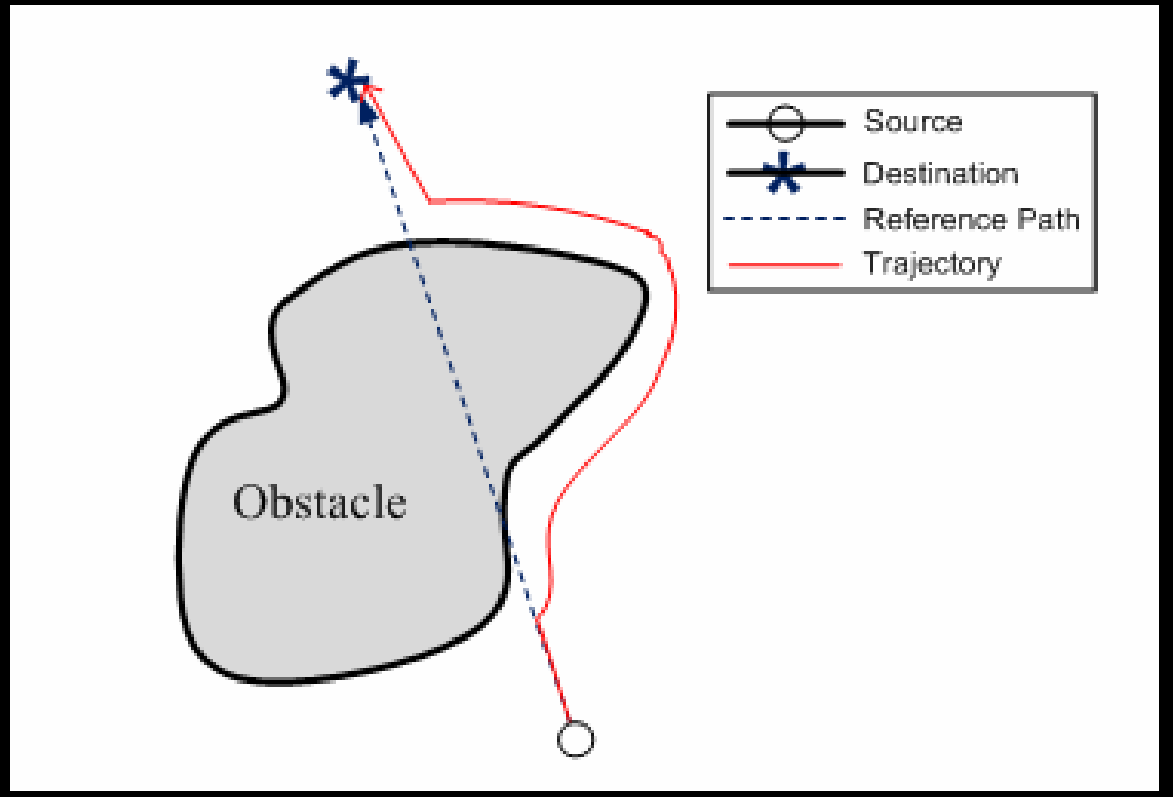
Greedy Search

Bug 2
Layout 2

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

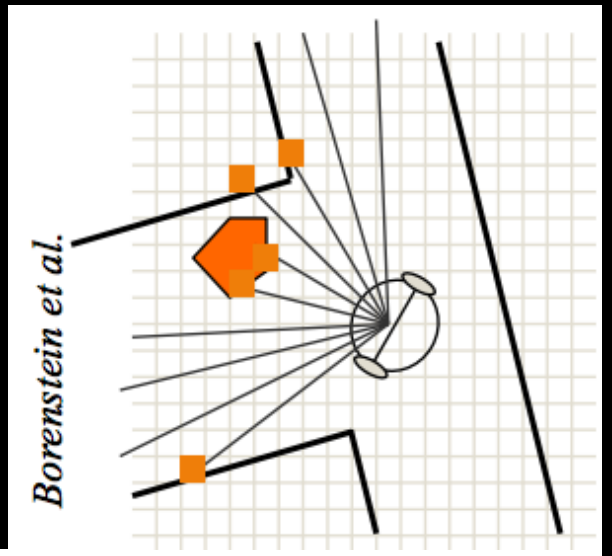
- The robot motion behavior is *reactive*
- Issues if the instantaneous sensor readings do not provide enough information or are noisy

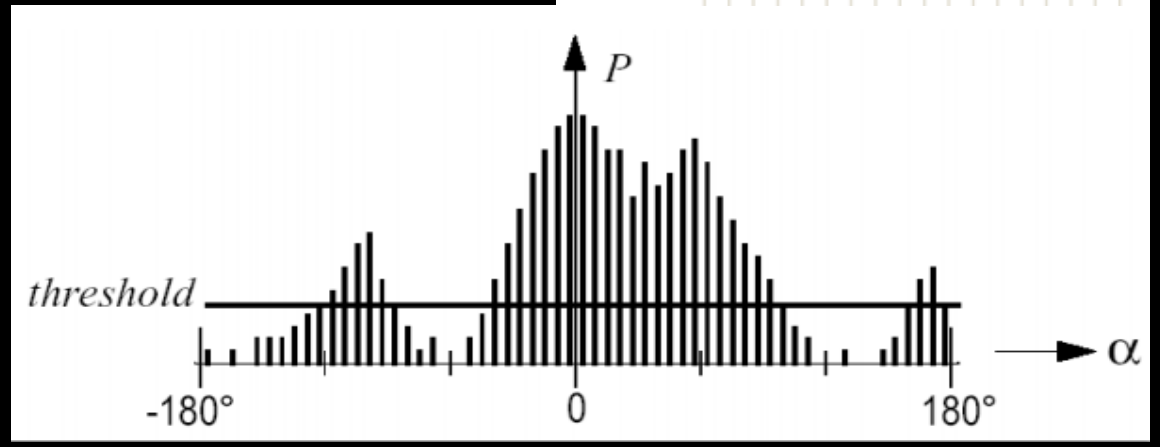


Vector Field Histograms

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

- VFH creates a local map of the environment around the robot populated by “relatively” recent sensor readings
- Build a local 2D grid map → reduce to 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

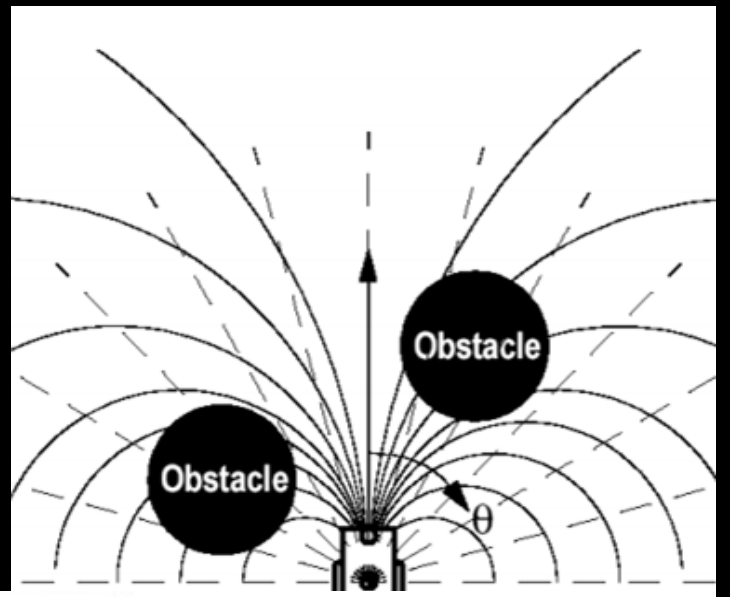


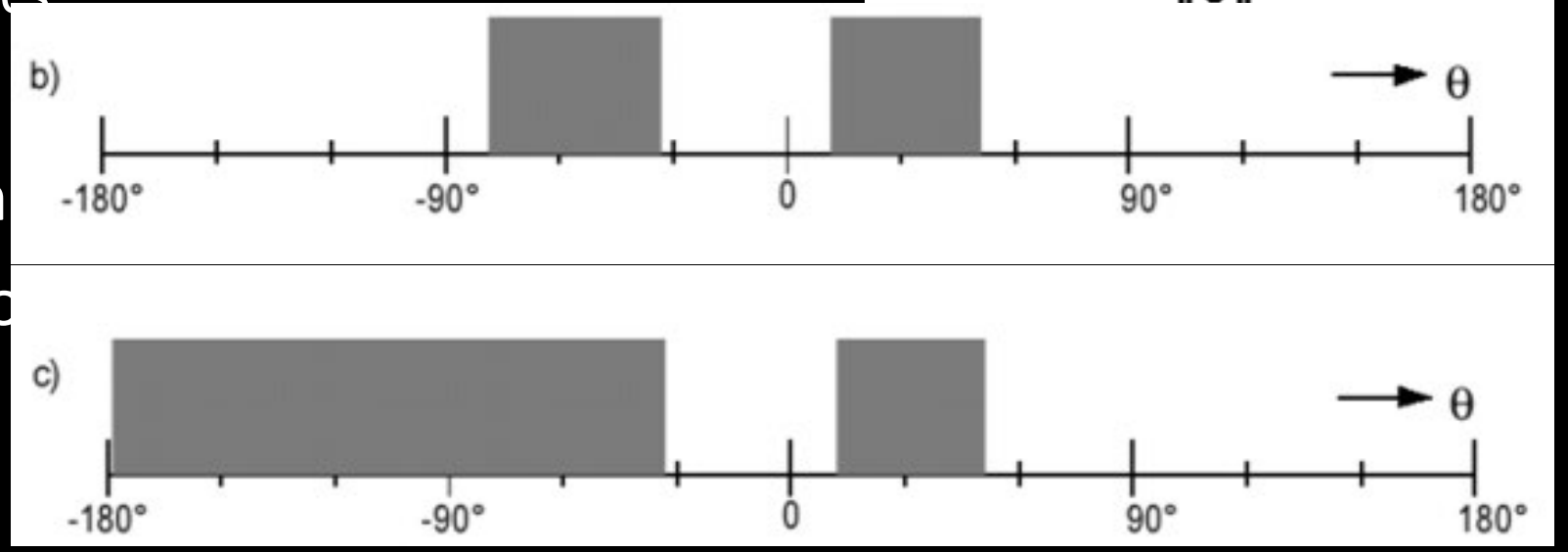


Vector Field Histograms

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

- VFH creates a local map of the environment around the robot populated by “relatively” recent sensor readings
- Build a local 2D grid map → reduce to 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
  - VHF+: Incorporate kinematics
- Limitations
- Does not avoid local minim
- Not guaranteed to reach go





Dynamic Window Approach

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

- Search in the velocity space (robot moves in circular arcs)
  - Takes into account robot acceleration capabilities and update rate
- A dynamic window, V_d, is the set of all tuples (v_d, ω_d) that can be reached
- Admissible velocities, V_a, include those where the robot can stop before collision
- The search sp
- Cost function:

$$G(v,\omega) = \sigma(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot velocity(v,\omega))$$

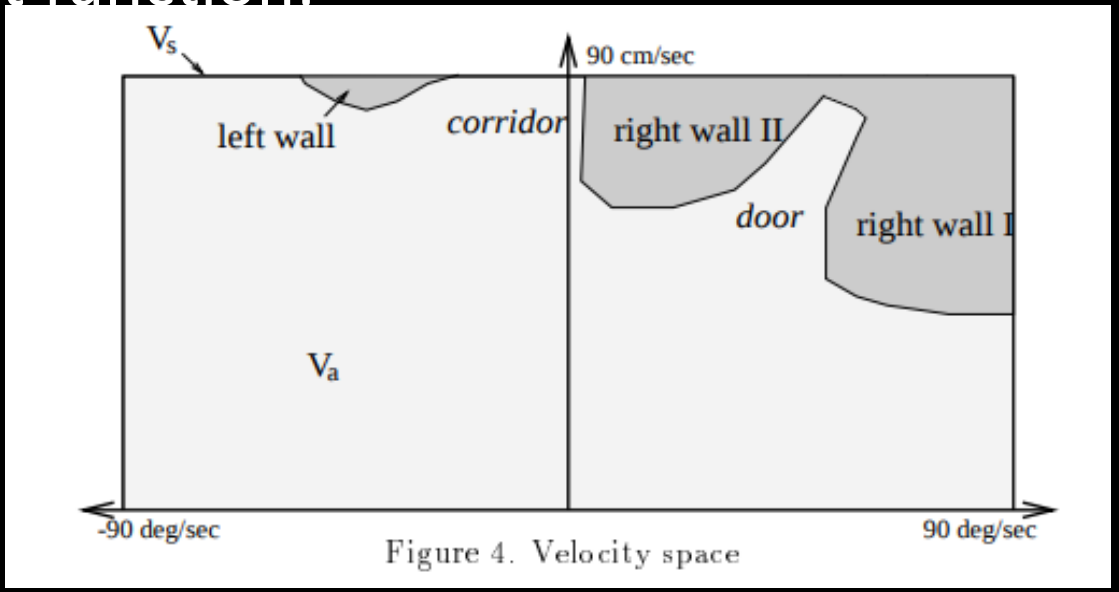
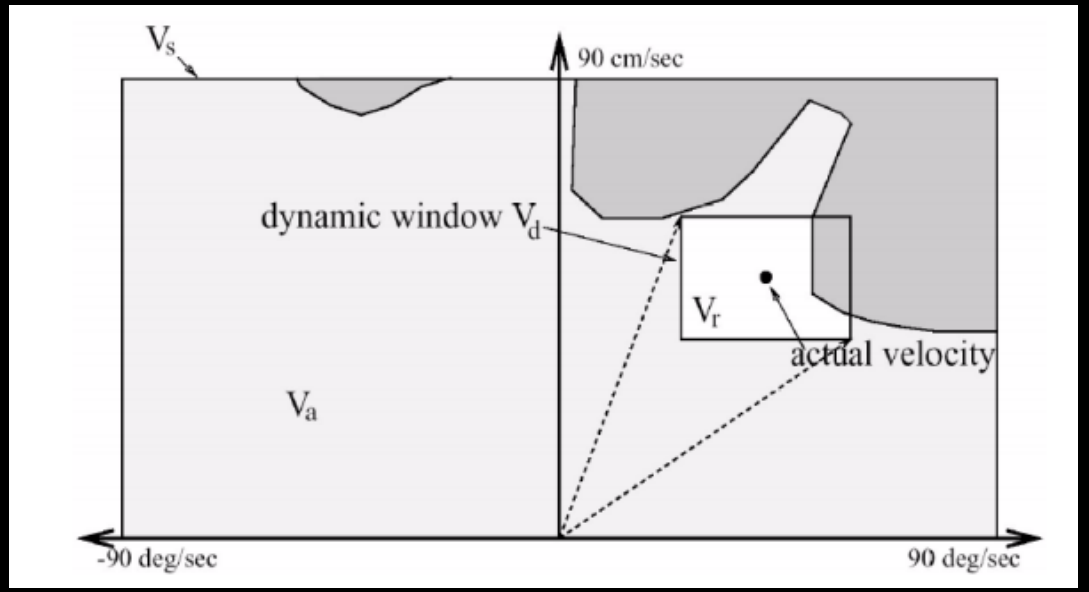


Figure 4. Velocity space



Local Planning Algorithms, Summary

- 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