Autonomous Robotics

Spring 2026

Abhishek Gupta, Siddhartha Srinivasa

TAs: Helen Wang, Sidharth Talia, Rohan Baijal, Christopher Tan

Recap

Discrete Kalman Filter: Matrix Version

Estimates the state ***x*** of a discrete-time controlled process that is governed by the linear stochastic difference equation

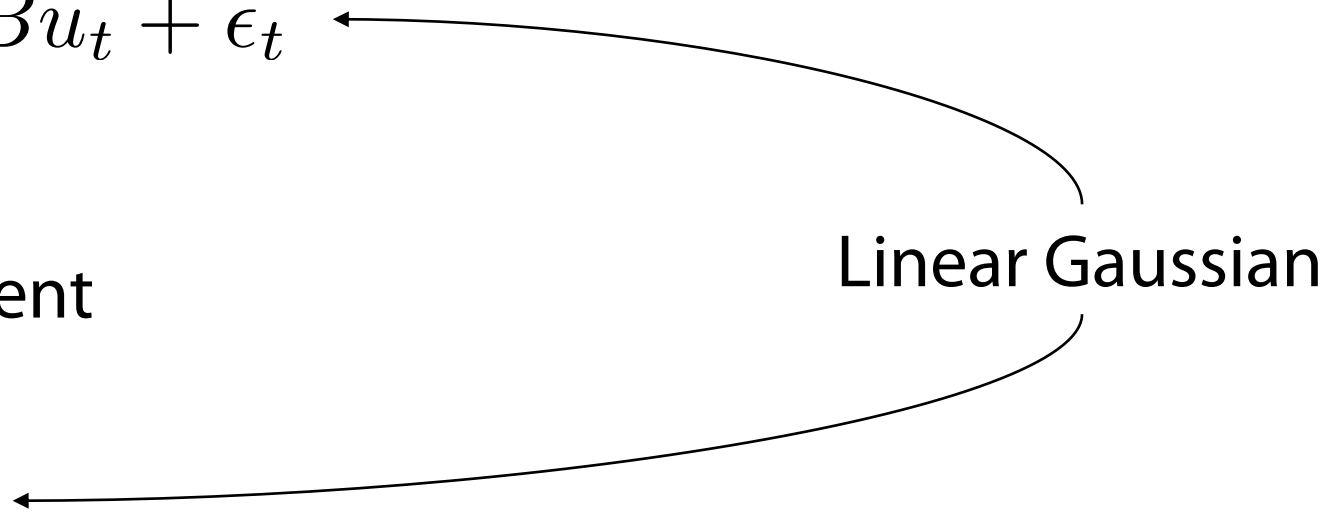
$$x_t = Ax_{t-1} + Bu_t + \epsilon_t$$

$$\epsilon_t \sim \mathcal{N}(0, Q)$$

with a measurement

$$z_t = Cx_t + \delta_t$$

$$\delta_t \sim \mathcal{N}(0, R)$$



Linear Gaussian Systems: Prediction

- Integrate the effect of one action under the dynamics, before measurement comes in

$$p(x_t|u_{0:t}, z_{0:t}) = \mathcal{N}(\mu_{t|0:t}, \Sigma_{t|0:t})$$

$$x_{t+1} = Ax_t + Bu_{t+1} + \epsilon_{t+1}$$

$$\epsilon_{t+1} \sim \mathcal{N}(0, Q_{t+1})$$

$$\begin{cases} X \sim \mathcal{N}(\mu, \Sigma) \\ Y = AX + B + \epsilon \implies Y \sim \mathcal{N}(A\mu + B, A\Sigma A^T + Q) \\ \epsilon \sim \mathcal{N}(0, C) \end{cases}$$

Previous belief	$p(x_t\|u_{0:t}, z_{0:t}) = \mathcal{N}(\mu_{t\|0:t}, \Sigma_{t\|0:t})$
Belief Update	$p(x_{t+1}\|u_{0:t+1}, z_{0:t}) = \mathcal{N}(A\mu_{t\|0:t} + Bu_{t+1}, A\Sigma_{t\|0:t}A^T + Q_{t+1})$

Intuition: Scale and shift the mean according to dynamics, uncertainty grows quadratically!

Linear Gaussian Systems: Observations

- Integrate the effect of an observation using sensor model, after dynamics

$$p(x_{t+1}|u_{0:t+1}, z_{0:t}) = \mathcal{N}(\mu_{t+1|0:t}, \Sigma_{t+1|0:t})$$
$$z_{t+1} = Cx_{t+1} + \delta_{t+1}$$
$$\delta_{t+1} \sim \mathcal{N}(0, R_{t+1})$$

$$\begin{cases} X \sim \mathcal{N}(\mu, \Sigma) \\ Y = CX + B + \delta \\ \delta \sim \mathcal{N}(0, R) \end{cases} \implies X|Y = y_0 \sim \mathcal{N}(\mu + K(y_0 - C\mu), (I - KC)\Sigma)$$
$$K = \Sigma C^T (C\Sigma C^T + R)^{-1}$$

Previous belief $p(x_{t+1}|u_{0:t+1}, z_{0:t}) = \mathcal{N}(\mu_{t+1|0:t}, \Sigma_{t+1|0:t})$ Computed from dynamics step

Updated belief $p(x_{t+1}|u_{0:t+1}, z_{0:t+1})$

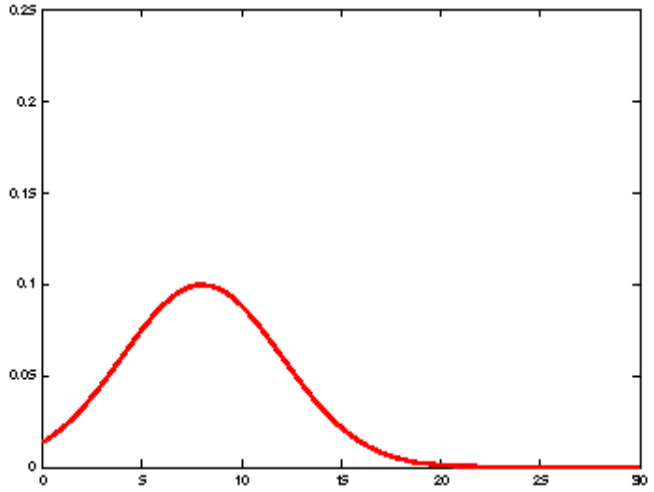
$$= \mathcal{N}(\mu_{t+1|0:t} + K_{t+1}(z_{t+1} - C\mu_{t+1|0:t}), (I - K_{t+1}C)\Sigma_{t+1|0:t})$$

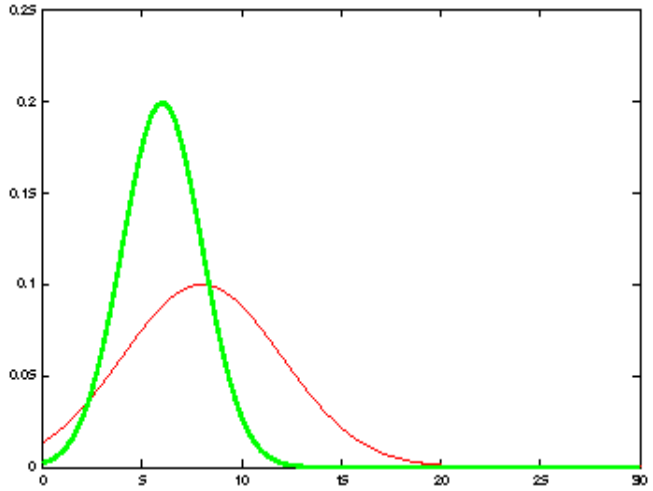
$$K_{t+1} = \Sigma_{t+1|0:t} C^T (C\Sigma_{t+1|0:t} C^T + R)^{-1}$$

Intuition Behind Correction Step

Previous belief

New Measurement

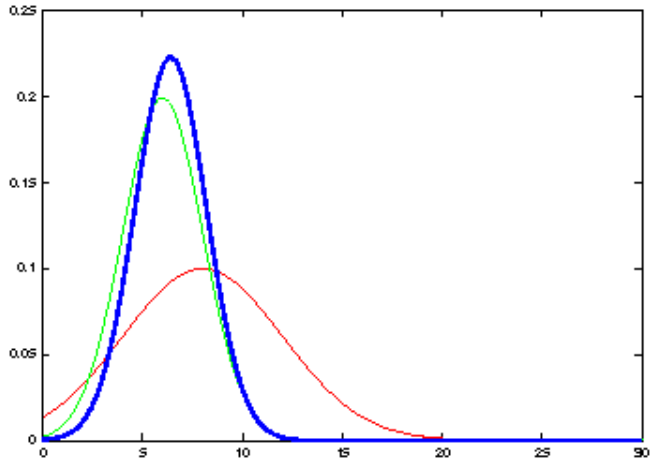




$$p(x_{t+1}|u_{0:t+1}, z_{0:t+1}) = \mathcal{N}(\mu_{t+1|0:t} + K_{t+1}(z_{t+1} - C\mu_{t+1|0:t}), (I - K_{t+1}C)\Sigma_{t+1|0:t})$$
$$K_{t+1} = \Sigma_{t+1|0:t}C^T(C\Sigma_{t+1|0:t}C^T + R)^{-1}$$

For the sake of simplicity, let's say C = I

$$K_{t+1} = \frac{\Sigma_{t+1|0:t}}{\Sigma_{t+1|0:t} + R}$$



Corrects belief based on measurement

→ Average between mean and measurement based on K

→ Scale down uncertainty based on K

Unpacking the Kalman Gain

Previous belief $p(x_{t+1}|u_{0:t+1}, z_{0:t}) = \mathcal{N}(\mu_{t+1|0:t}, \Sigma_{t+1|0:t})$ Computed from dynamics step

Updated belief $p(x_{t+1}|u_{0:t+1}, z_{0:t+1})$

$$= \mathcal{N}(\mu_{t+1|0:t} + K_{t+1}(z_{t+1} - C\mu_{t+1|0:t}), (I - K_{t+1}C)\Sigma_{t+1|0:t})$$

$$K_{t+1} = \Sigma_{t+1|0:t}C^T(C\Sigma_{t+1|0:t}C^T + R)^{-1}$$

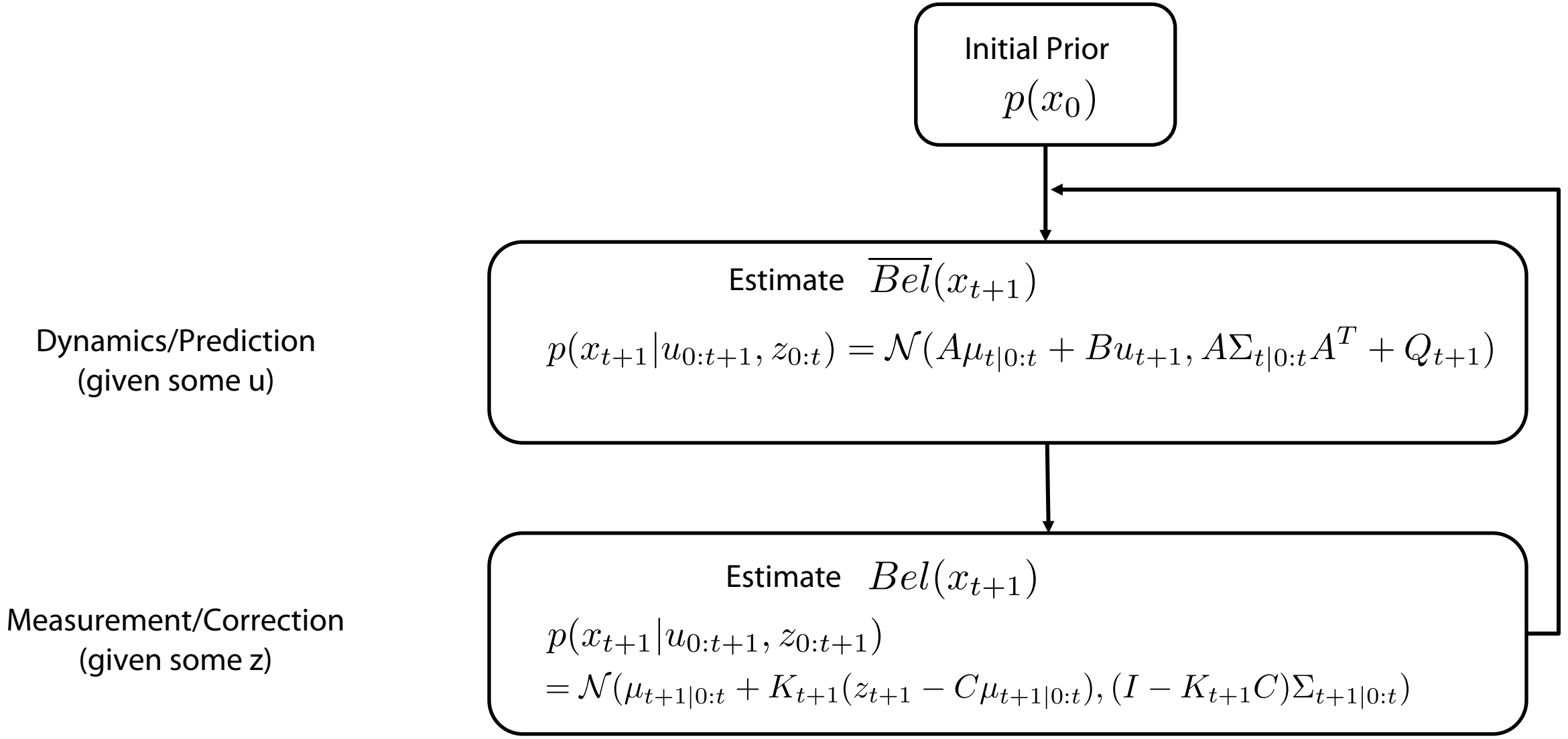
Case 1: Very noisy sensor, R>>Σ

For the sake of simplicity, let's say C = I

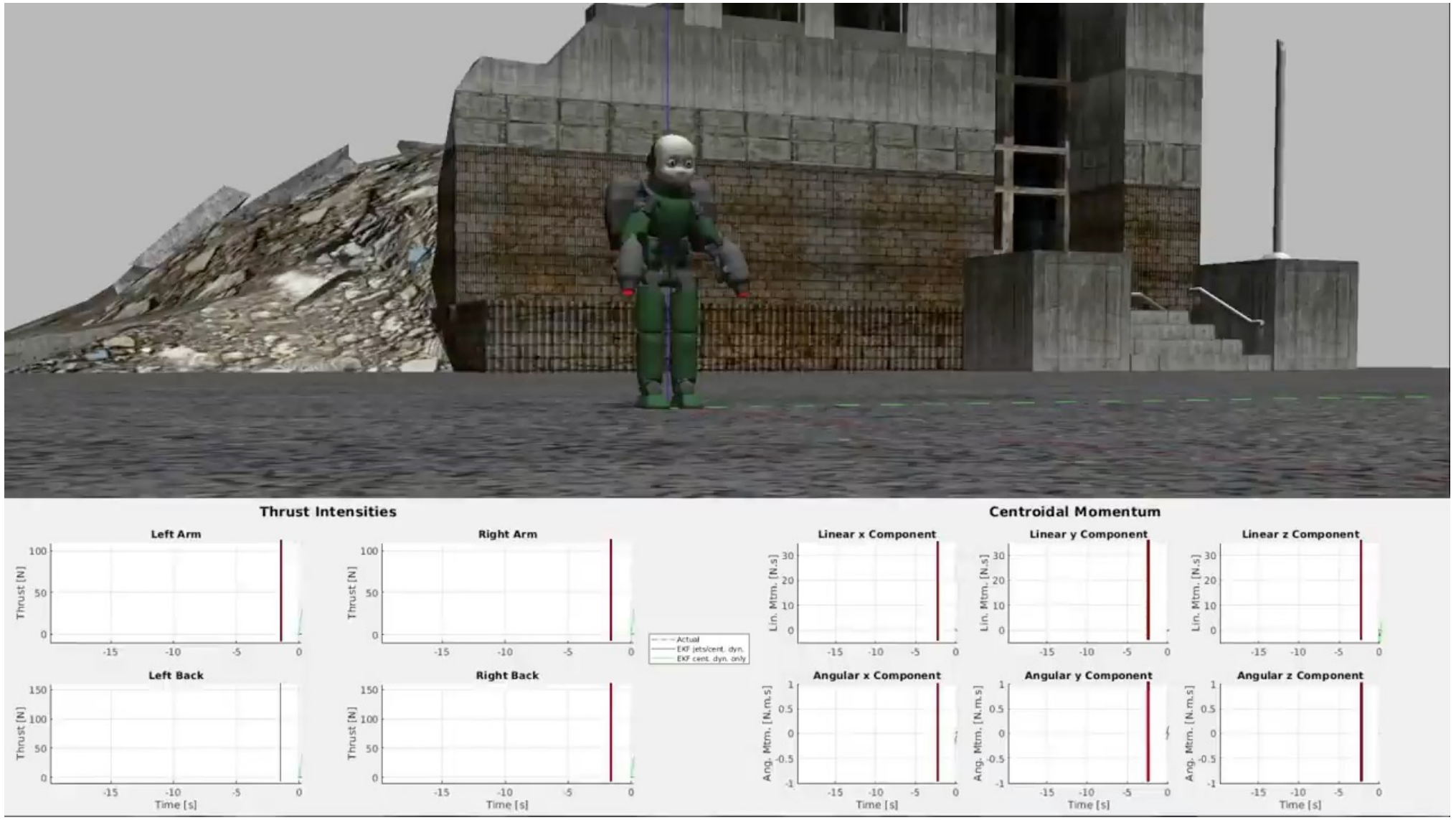
$$K_{t+1} = \frac{\Sigma_{t+1|0:t}}{\Sigma_{t+1|0:t} + R}$$

Case 2: Deterministic sensor, R = 0

Kalman Filter Algorithm



Kalman Filter in Action



Kalman Filter Summary

- Highly efficient: Polynomial in measurement dimensionality ***k*** and state dimensionality ***n***:

  $O(k^{2.376} + n^2)$

Matrix Inversion (Correction)

$$K_{t+1} = \Sigma_{t+1|0:t} C^T (C \Sigma_{t+1|0:t} C^T + R_{t+1})^{-1}$$

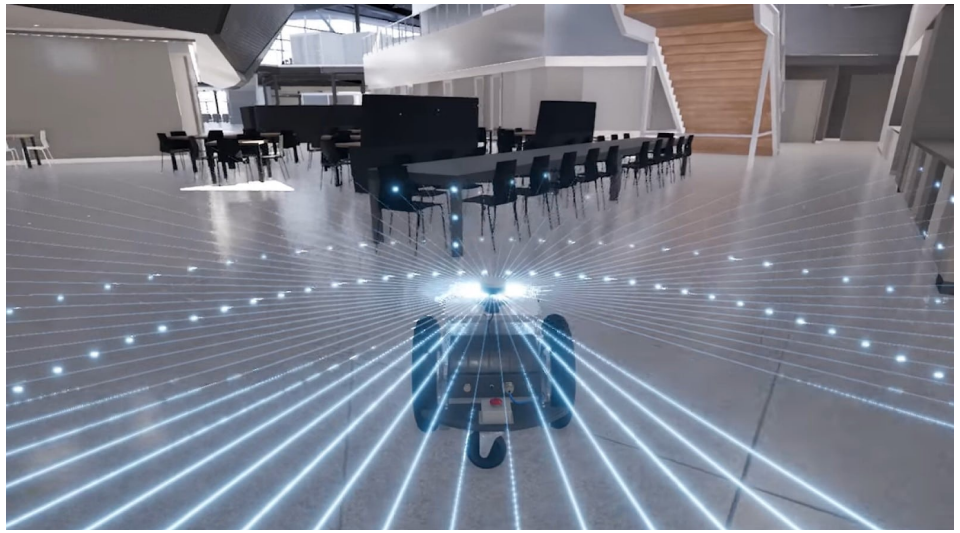
Matrix Multiplication (Prediction)

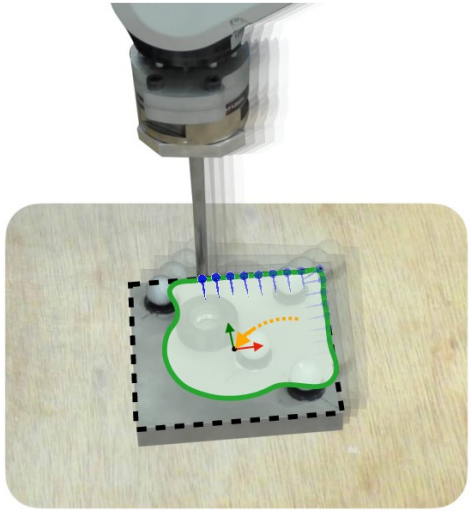
$$p(x_{t+1}|z_{0:t}, u_{0:t+1}) \sim \mathcal{N}(A\mu_{t|0:t} + Bu_t, A\Sigma_{t|0:t}A^T + Q_t)$$

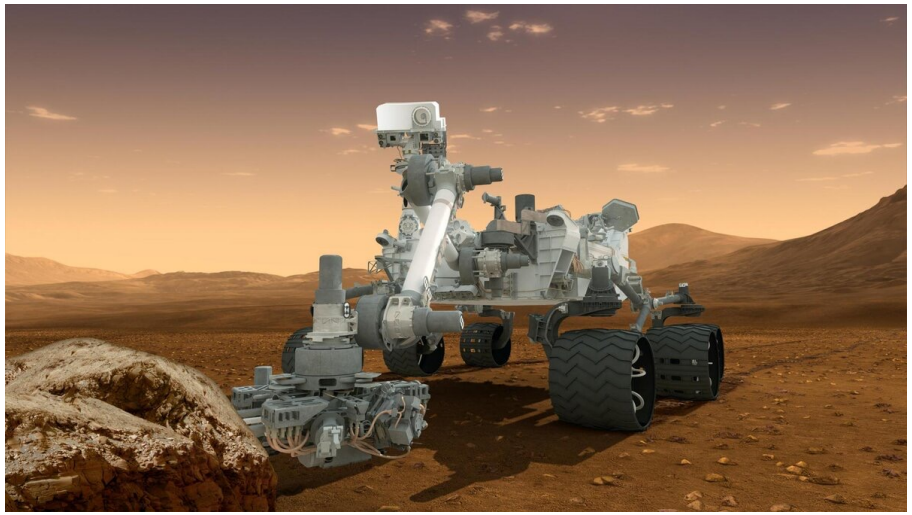
- Optimal for linear Gaussian systems!
- Most robotics systems are nonlinear!

Why should we care?

Still a very widely used technique for estimation/localization/mapping in real problems







Lecture Outline

Kalman Filtering

Extended Kalman Filter

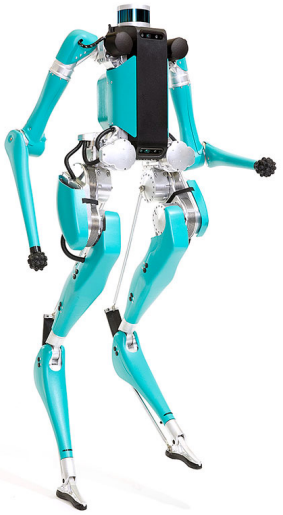
Intro to Control

Realistic Robotic Systems

- Most realistic robotic problems involve nonlinear functions

$$x_{t+1} = Ax_t + Bu_t + \epsilon_t$$
$$\epsilon_t \sim \mathcal{N}(0, Q)$$

$$z_{t+1} = Cx_{t+1} + \delta_t$$
$$\delta_t \sim \mathcal{N}(0, R)$$



Nonlinear Dynamic Systems

- Most realistic robotic problems involve nonlinear functions

$$x_{t+1} = g(x_t, u_t) + \epsilon_t$$

Non-linear system

$$z_t = h(x_t) + \delta_t$$

$$\epsilon_t \sim \mathcal{N}(0, Q)$$

Additive Gaussian noise

$$\delta_t \sim \mathcal{N}(0, R)$$

More reasonable assumption than linear Gaussian. More on non-Gaussian systems next time

How do we deal with non-linearity?

- Differentiable non-linear functions can be expressed via their Taylor expansion

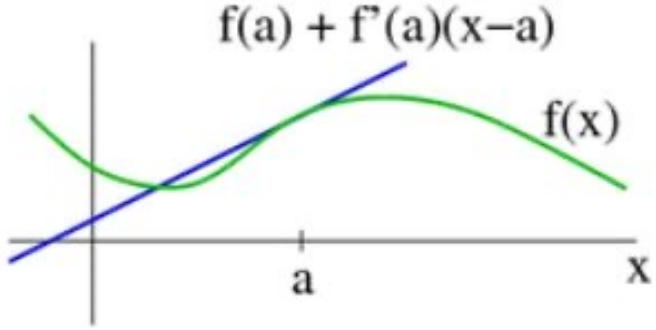
$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots,$$

$$f(x) \approx f(a) + \frac{f'(a)}{1!}(x-a)$$

Dropping higher order terms, when x-a is small enough

Linear function in x

Pretend that your function is linear in this neighborhood
→ Reapprox in a new neighborhood



EKF Linearization: First Order Taylor Series Expansion

- Idea behind EKF: Linearize the dynamics and measurement around current μ_t
- Dynamics Model (linearize around previous belief):

$$x_{t+1} = g(x_t, u_t) + \epsilon_t \approx g(\mu_t, u_t) + \left.\frac{\partial g(x_t, u_t)}{\partial x_t}\right|_{x_t=\mu_t} (x_t - \mu_t) + \epsilon_t$$

$$= g(\mu_t, u_t) + G(x_t - \mu_t) + \epsilon_t$$

- Measurement Model (linearize around post dynamics belief):

$$z_t = h(x_t) + \delta_t \approx h(\bar{\mu}_t) + \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{x_t=\bar{\mu}_t} (x_t - \bar{\mu}_t) + \delta_t \approx h(\bar{\mu}_t) + H(x_t - \bar{\mu}_t) + \delta_t$$

Now everything is linear → back to Kalman filtering!

Modified System under EKF Linearization

- Start by linearizing dynamics model under current belief
- Dynamics Model (linearize around previous belief):

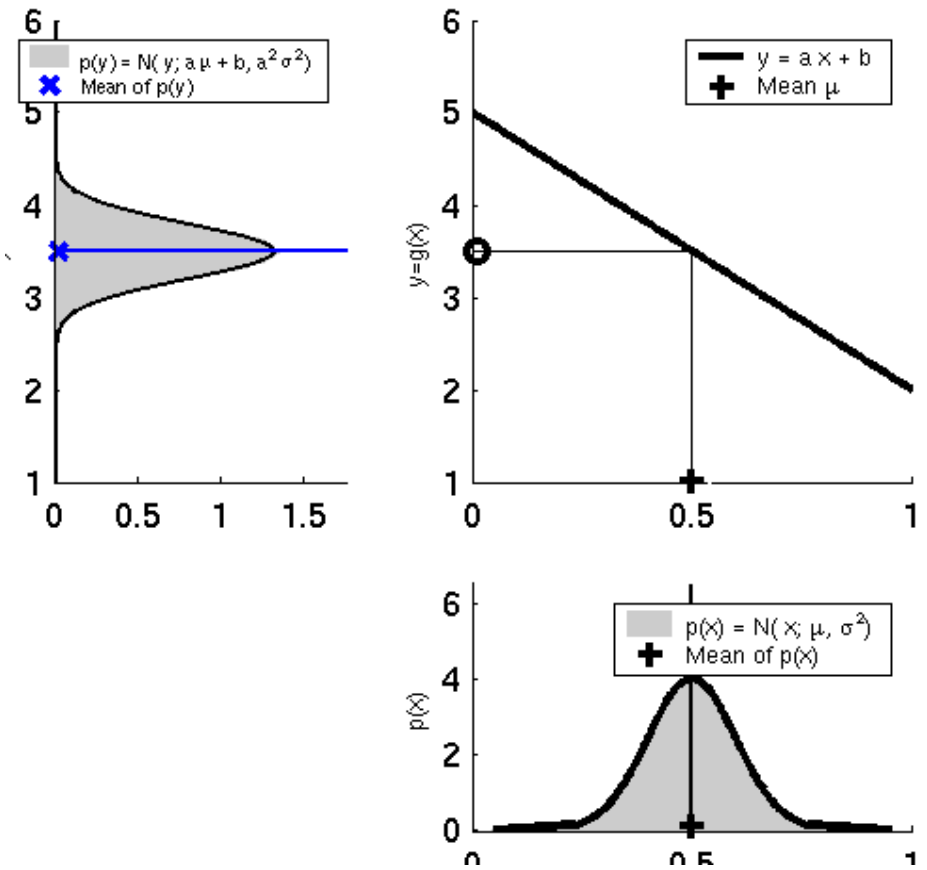
$$x_{t+1} = g(x_t, u_t) + \epsilon_t \qquad \approx g(\mu_t, u_t) + \left.\frac{\partial g(x_t, u_t)}{\partial x_t}\right|_{x_t=\mu_t} (x_t - \mu_t) + \epsilon_t$$

- Perform dynamics update
- Linearize measurement around post dynamics belief

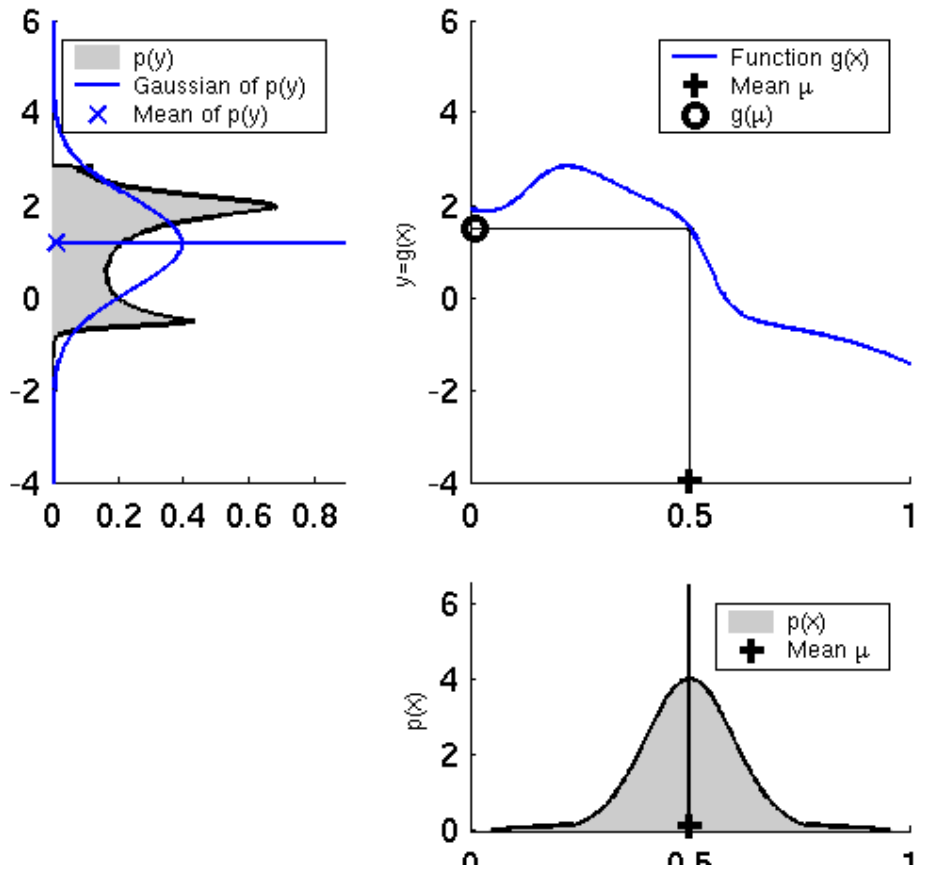
$$z_t = h(x_t) + \delta_t \approx h(\bar{\mu}_t) + \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{x_t=\bar{\mu}_t} (x_t - \bar{\mu}_t) + \delta_t \approx h(\bar{\mu}_t) + H(x_t - \bar{\mu}_t) + \delta_t$$

- Perform measurement update
- Repeat

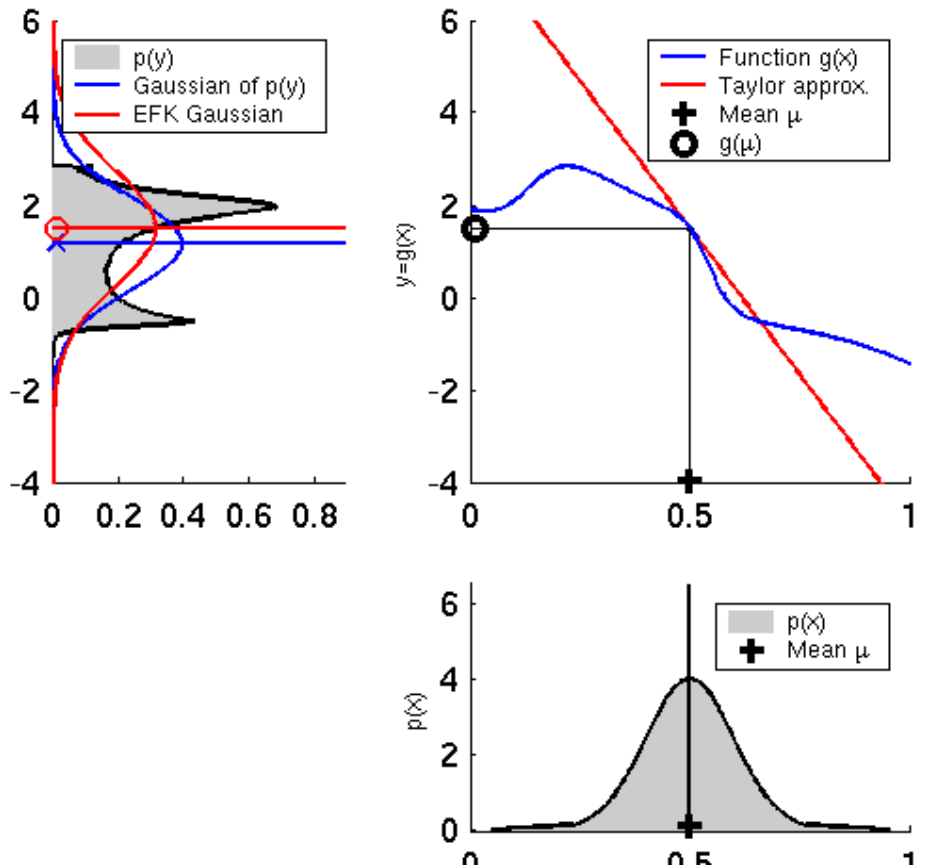
Linearity Assumption Revisited



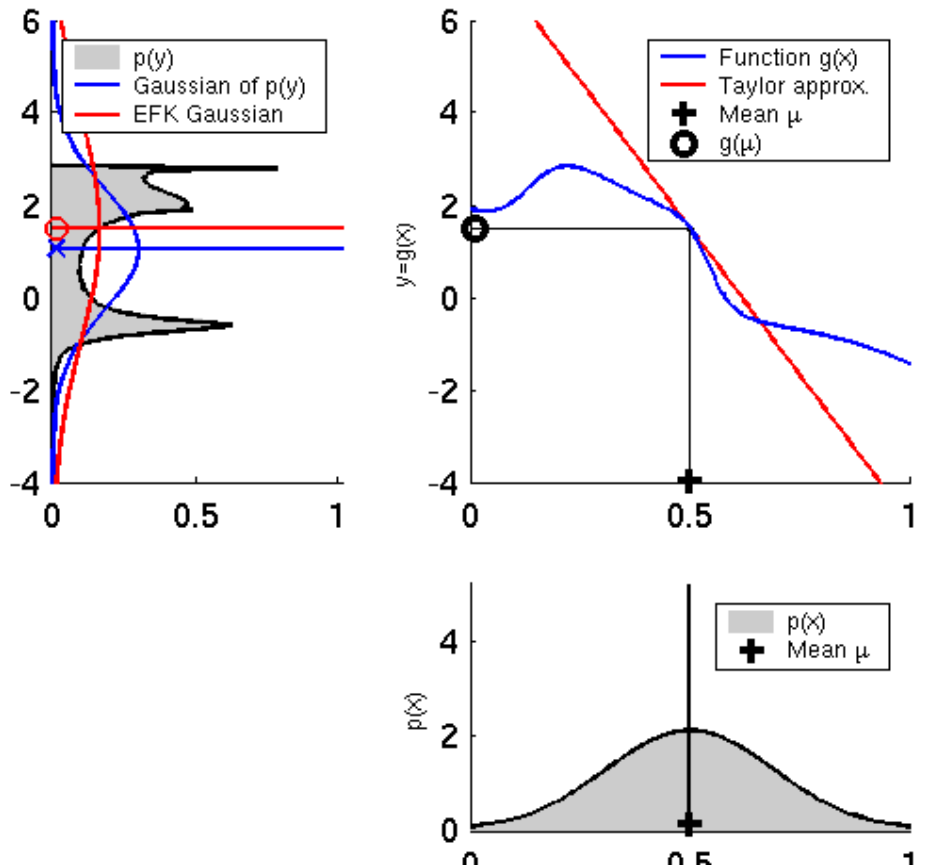
Non-linear Function



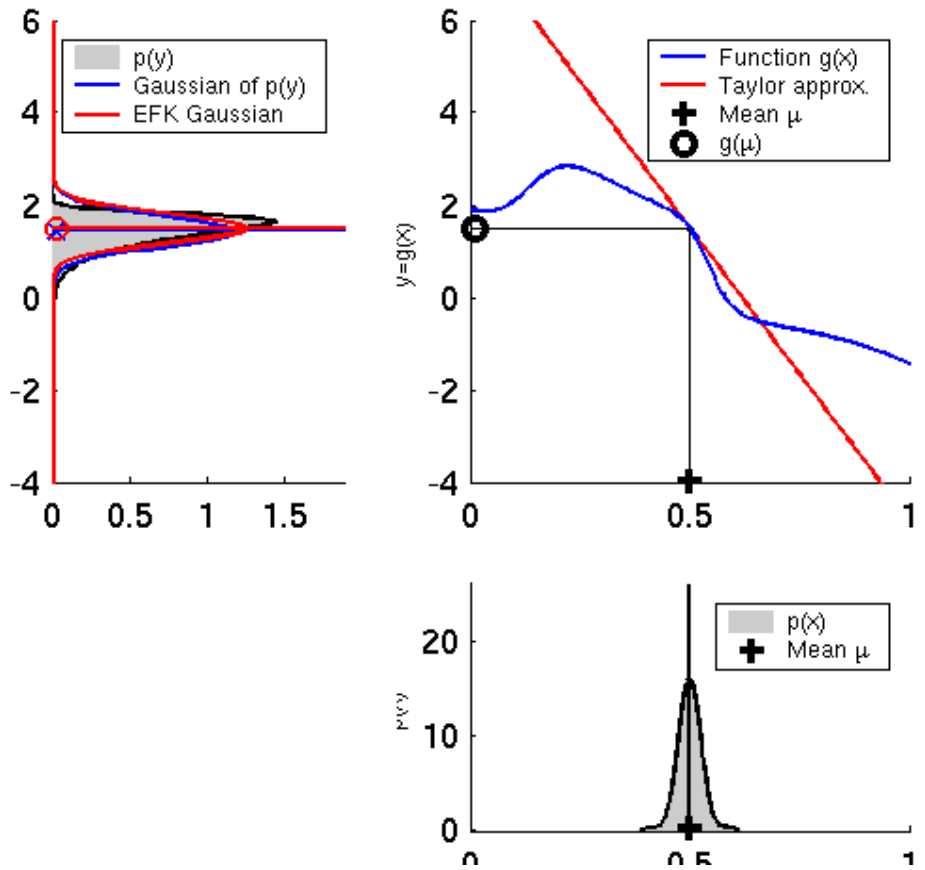
EKF Linearization (1)



EKF Linearization (2)



EKF Linearization (3)



EKF Dynamics Step

- Linearize dynamics around current mean belief

$$x_{t+1} = g(x_t, u_t) + \epsilon_t \approx g(\mu_t, u_t) + \left.\frac{\partial g(x_t, u_t)}{\partial x_t}\right|_{x_t=\mu_t} (x_t - \mu_t) + \epsilon_t$$

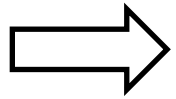
$$= g(\mu_t, u_t) + G(x_t - \mu_t) + \epsilon_t$$

- Run standard Kalman filter with A = G and Bu = $g(\mu_t, u_t)$

$$p(x_{t+1}|z_{0:t}, u_{0:t})$$

$$\sim \mathcal{N}(A\mu_{t|0:t} + Bu_t, A\Sigma_{t|0:t}A^T + Q_t)$$

Standard Kalman filter



$$p(x_{t+1}|z_{0:t}, u_{0:t})$$

$$\sim \mathcal{N}(g(\mu_t, u_t), G\Sigma_{t|0:t}G^T + Q_t)$$

Extended Kalman filter

EKF Measurement Step

- Linearize measurement around post dynamics belief:

$$z_t = h(x_t) + \delta_t \approx h(\bar{\mu}_t) + \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{x_t=\bar{\mu}_t} (x_t - \bar{\mu}_t) + \delta_t \approx h(\bar{\mu}_t) + H(x_t - \bar{\mu}_t) + \delta_t$$

- Run standard Kalman filter with C = H

$$p(x_{t+1}|z_{0:t+1}, u_{0:t})$$
$$= \mathcal{N}(\mu_{t+1|0:t} + K_{t+1}(z_{t+1} - C\mu_{t+1|0:t}), (I - K_{t+1}C)\Sigma_{t+1|0:t})$$
$$K_{t+1} = \Sigma_{t+1|0:t}C^T(C\Sigma_{t+1|0:t}C^T + R)^{-1}$$

Standard Kalman Filter

$$p(x_{t+1}|z_{0:t+1}, u_{0:t})$$
$$= \mathcal{N}(\mu_{t+1|0:t} + K_{t+1}(z_{t+1} - h(\bar{\mu}_t), (I - K_{t+1}H)\Sigma_{t+1|0:t})$$
$$K_{t+1} = \Sigma_{t+1|0:t}H^T(H\Sigma_{t+1|0:t}H^T + R)^{-1}$$

Extended Kalman Filter

EKF Algorithm

Initial Prior

$$p(x_0)$$

Linearize dynamics

$$x_{t+1} = g(x_t, u_t) + \epsilon_t \approx g(\mu_t, u_t) + \left.\frac{\partial g(x_t, u_t)}{\partial x_t}\right|_{x_t=\mu_t} (x_t - \mu_t) + \epsilon_t$$

Dynamics/Prediction
(given some u)

Estimate $\overline{Bel}(x_t)$

$$p(x_{t+1}|z_{0:t}, u_{0:t}) \sim \mathcal{N}(g(\mu_t, u_t), G\Sigma_{t|0:t}G^T + Q_t)$$

Linearize measurement

$$z_t = h(x_t) + \delta_t \approx h(\bar{\mu}_t) + \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{x_t=\bar{\mu}_t} (x_t - \bar{\mu}_t) + \delta_t$$

Measurement/Correction
(given some z)

Estimate $Bel(x_t)$

$$p(x_{t+1}|z_{0:t+1}, u_{0:t}) = \mathcal{N}(\mu_{t+1|0:t} + K_{t+1}(z_{t+1} - h(\bar{\mu}_t), (I - K_{t+1}H)\Sigma_{t+1|0:t})$$

EKF Pseudocode

1. def Extended_Kalman_filter($\mu_{t|0:t}, \Sigma_{t|0:t}, u_t, z_{t+1}$):
2. Linearize dynamics: $G = \left.\frac{\partial g(x_t, u_t)}{\partial x_t}\right|_{x_t = \mu_{t|0:t}}$
3. Prediction:

$$\mu_{t+1|0:t} = g(\mu_{t|0:t}, u_t)$$
$$\Sigma_{t+1|0:t} = G\Sigma_{t|0:t}G^T + Q$$

4. Linear measurement: $H = \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{x_t = \mu_{t+1|0:t}}$
5. Correction: C = H

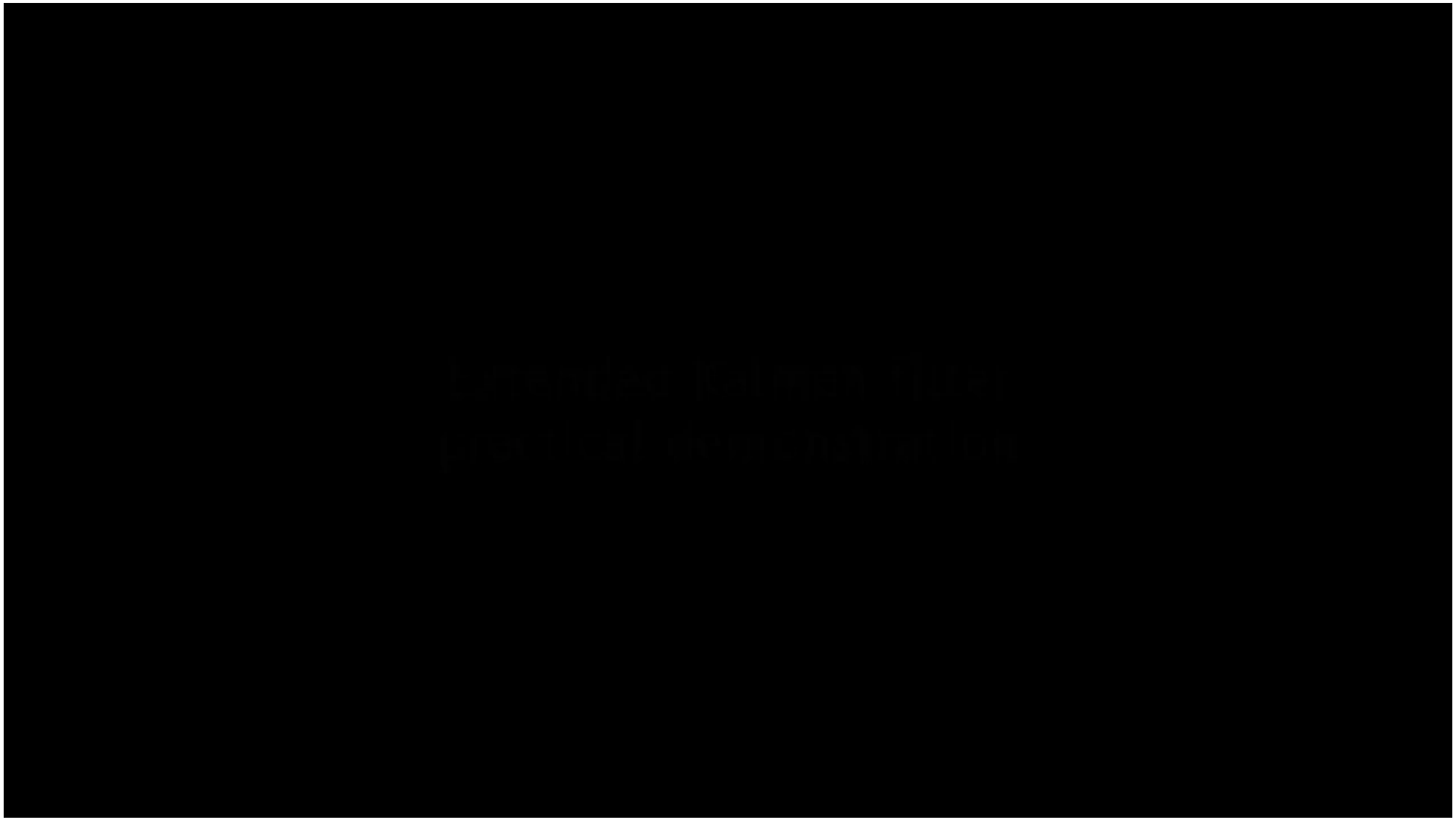
$$K_{t+1} = \Sigma_{t+1|0:t}H^T(H\Sigma_{t+1}H^T + R)^{-1}$$
$$\mu_{t+1|0:t+1} = \mu_{t+1|0:t} + K_{t+1}(z_{t+1} - h(\mu_{t+1|0:t}))$$
$$\Sigma_{t+1|0:t+1} = (I - K_{t+1}H)\Sigma_{t+1|0:t}$$

6. Return $\mu_{t+1|0:t+1}, \Sigma_{t+1|0:t+1}$

$$x_{t+1} = g(x_t, u_t) + \epsilon_t$$
$$z_t = h(x_t) + \delta_t$$
$$\epsilon_t \sim \mathcal{N}(0, Q)$$
$$\delta_t \sim \mathcal{N}(0, R)$$

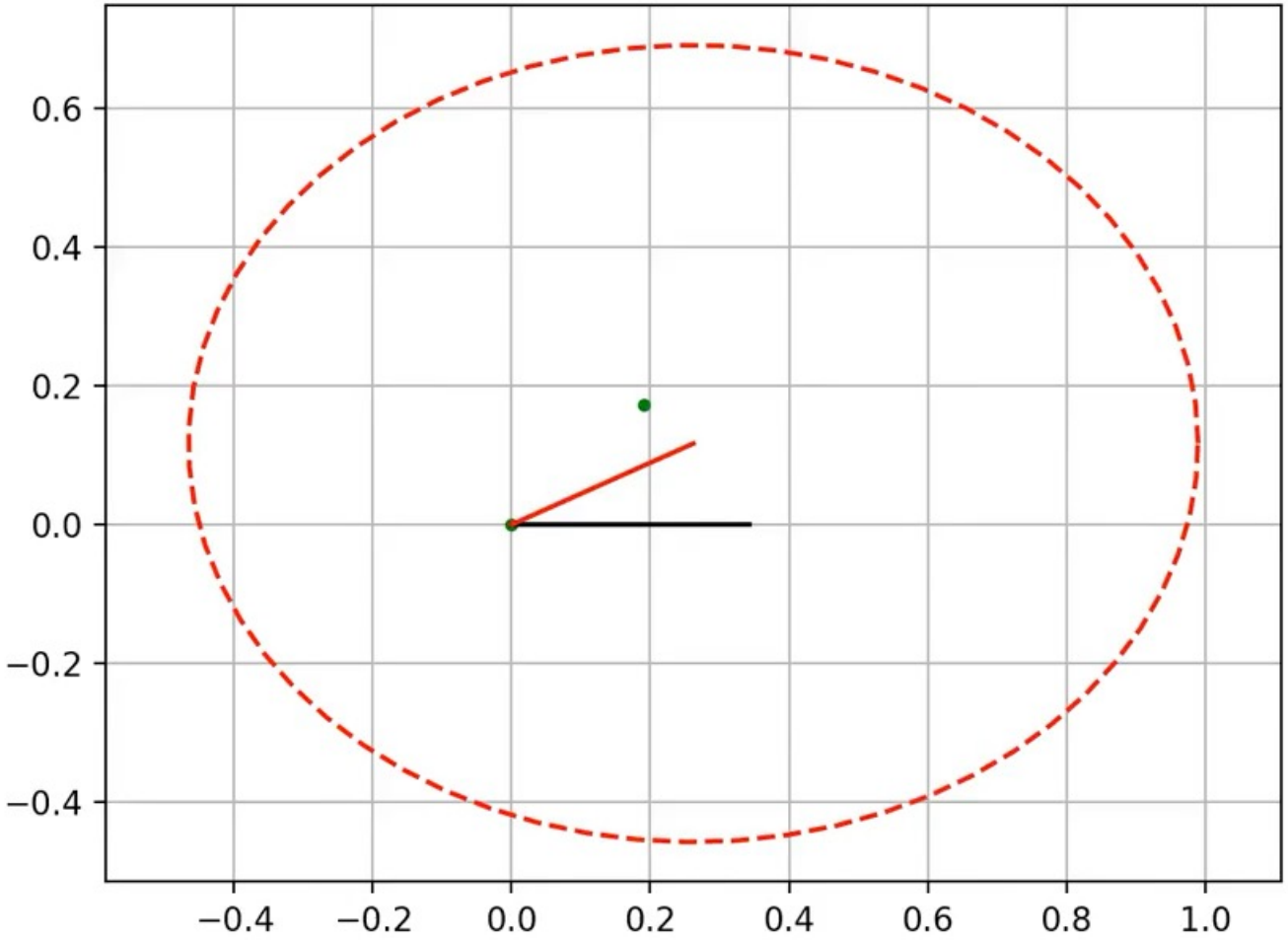
Reminder of the model

EKF in Action



https://www.youtube.com/watch?v=dEd4cNfmi0s&ab_channel=ThePoorEngineer

EKF in Action



https://www.youtube.com/watch?v=ju27V142D2o&ab_channel=AtsushiSakai

EKF Summary

- Highly efficient: Polynomial in measurement dimensionality ***k*** and state dimensionality ***n***:

  $$O(k^{2.376} + n^2)$$

- Not optimal!
- Can diverge if nonlinearities are large!
- Works surprisingly well even when all assumptions are violated!

When does the EKF struggle?

- With discontinuous dynamics, the linearization will not be valid
- For very non-linear functions, the first order Taylor approximation is poor
- The EKF can drift over time because of growing linearization errors
- Jacobian may be very expensive to compute and invert

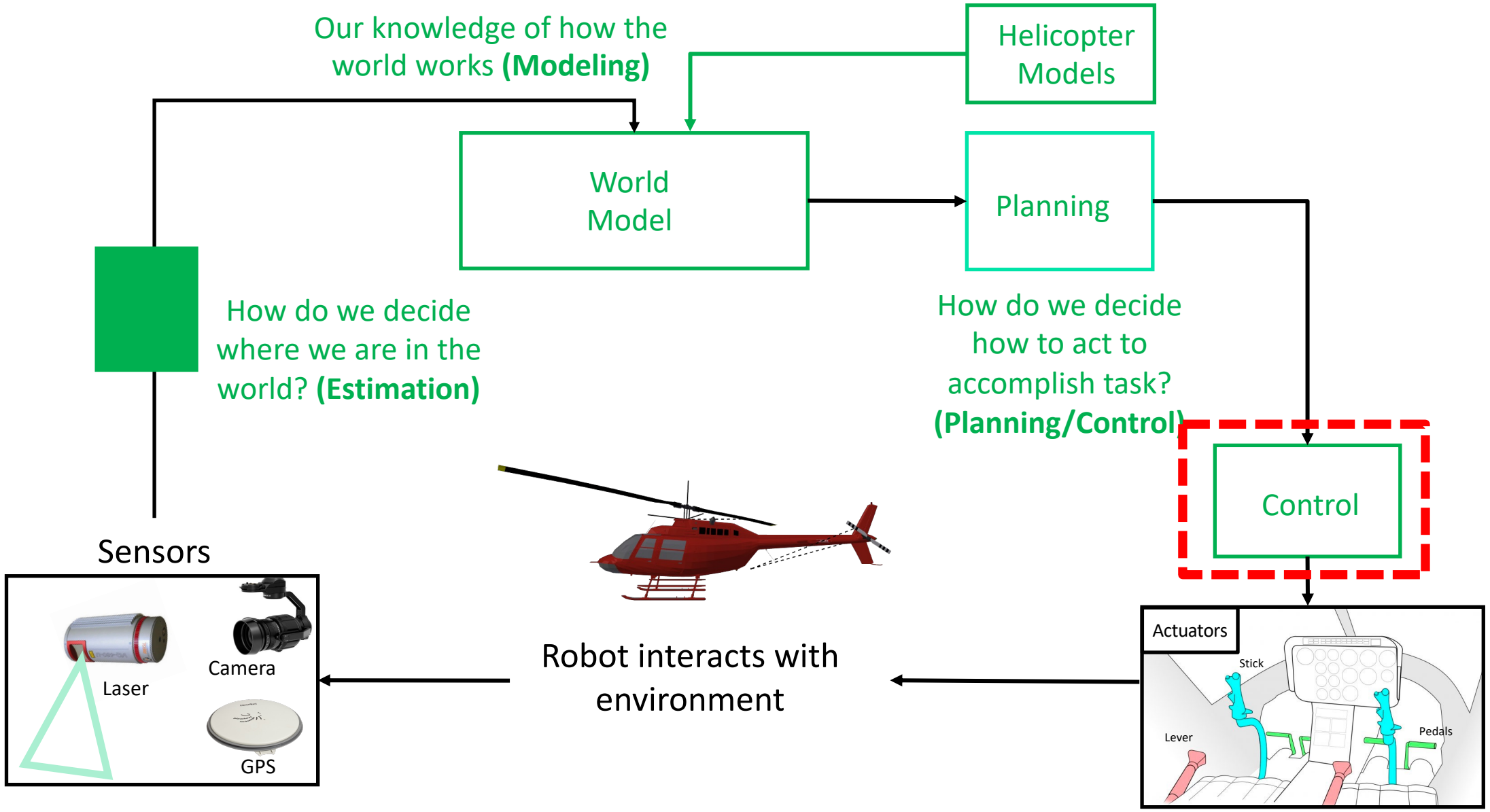
Lecture Outline

Kalman Filtering

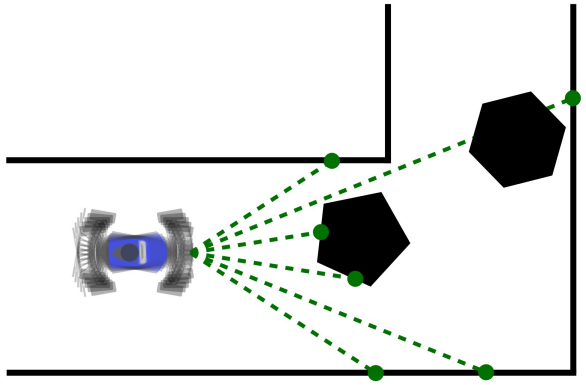
Extended Kalman Filter

Intro to Control

Let's zoom back out

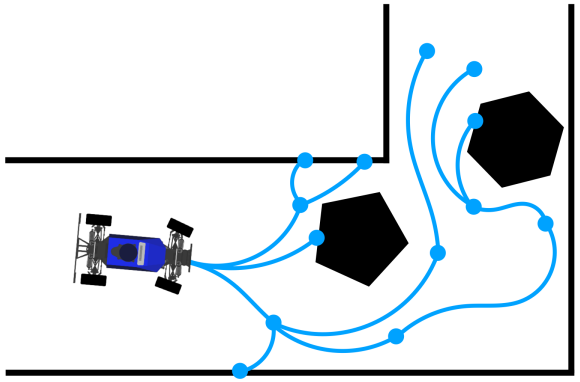


The Sense-Plan-Act Paradigm



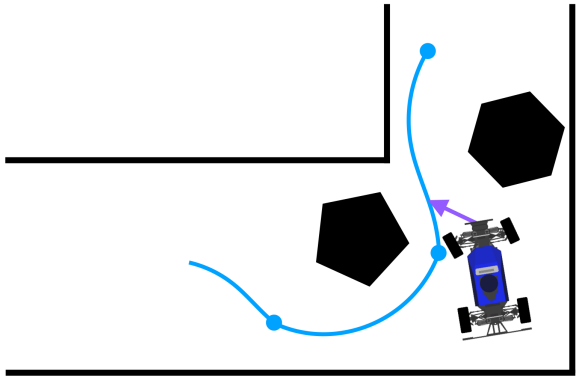
Estimate robot state

Solved over last 2.5 weeks



Plan sequence of motions

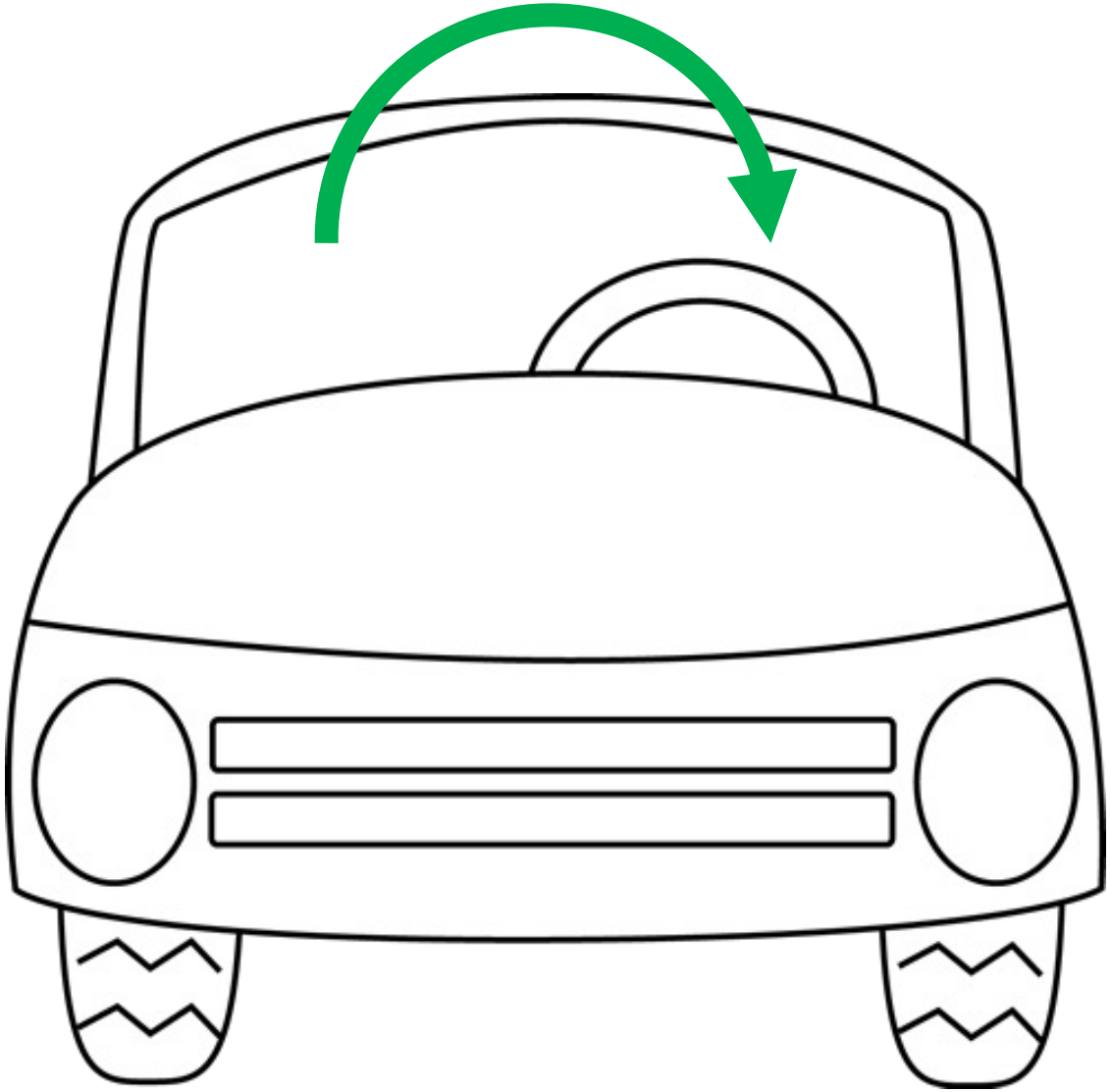
Assume to be solved for now



Control robot to follow plan

??

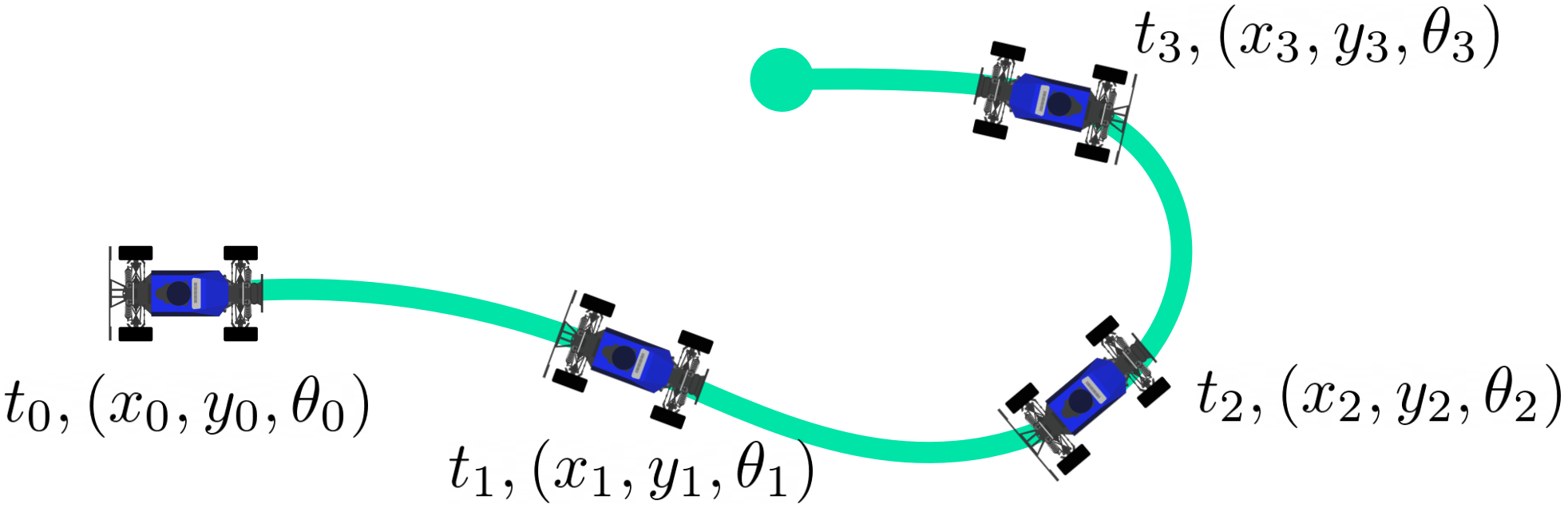
From perception to control …



When I think about control …

What is Control?

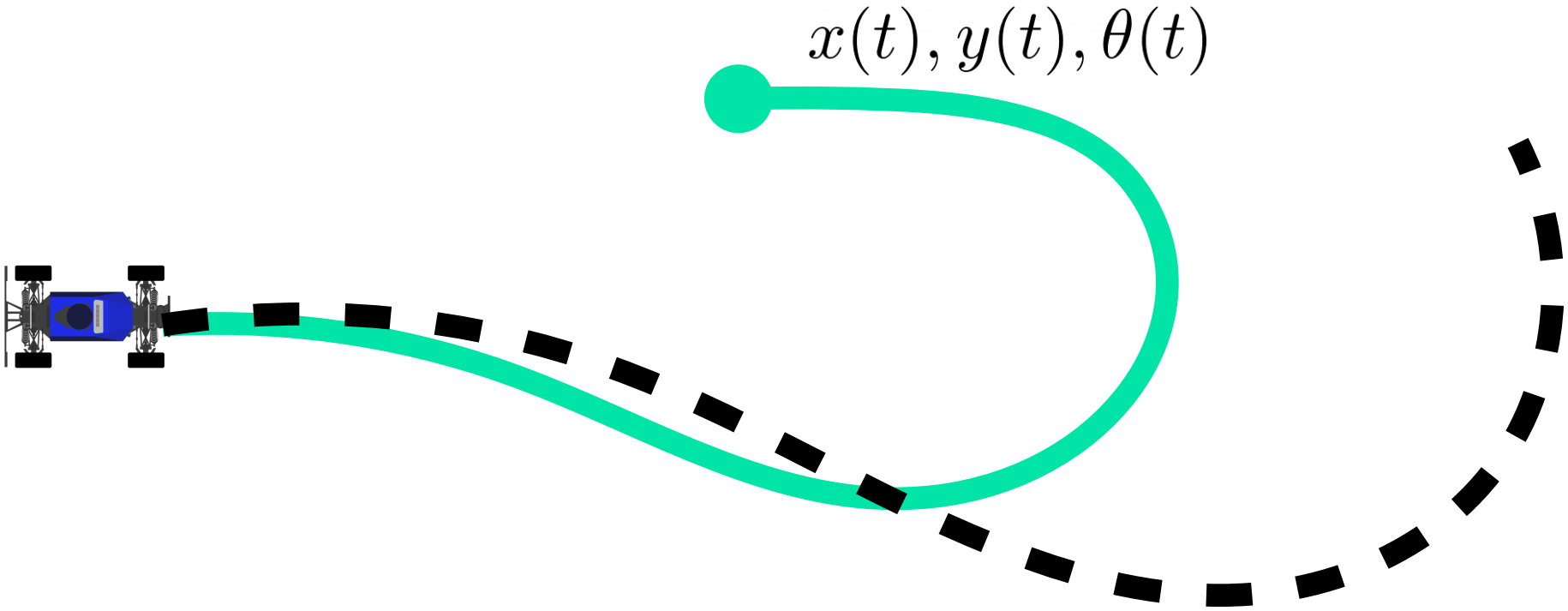
What is a Plan?



Can express this problem as **tracking a reference** trajectory

$$x(t), y(t), \theta(t)$$

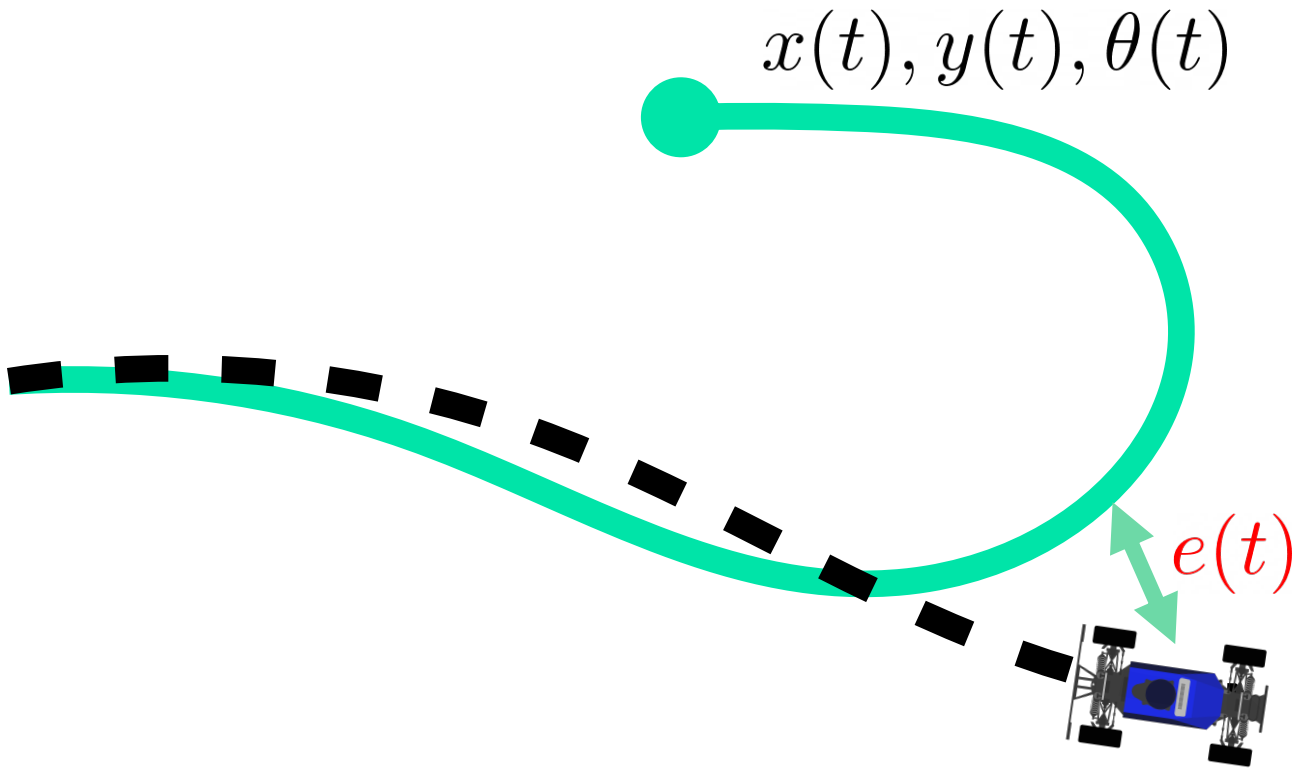
Why Feedback Control?



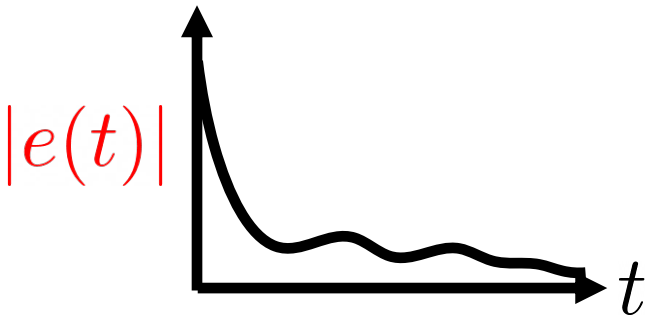
What if we send out controls $u(t)$ from kinematic car model?

Open-loop control leads to **accumulating errors**!

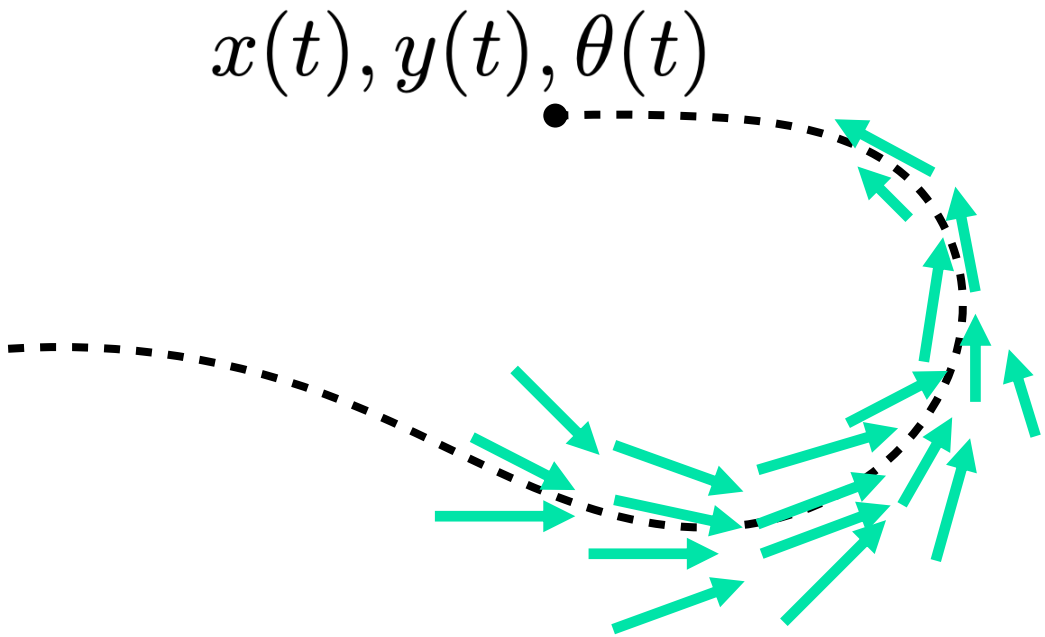
Feedback Control



1. Measure error between reference and current state.
2. Take actions to minimize this error.

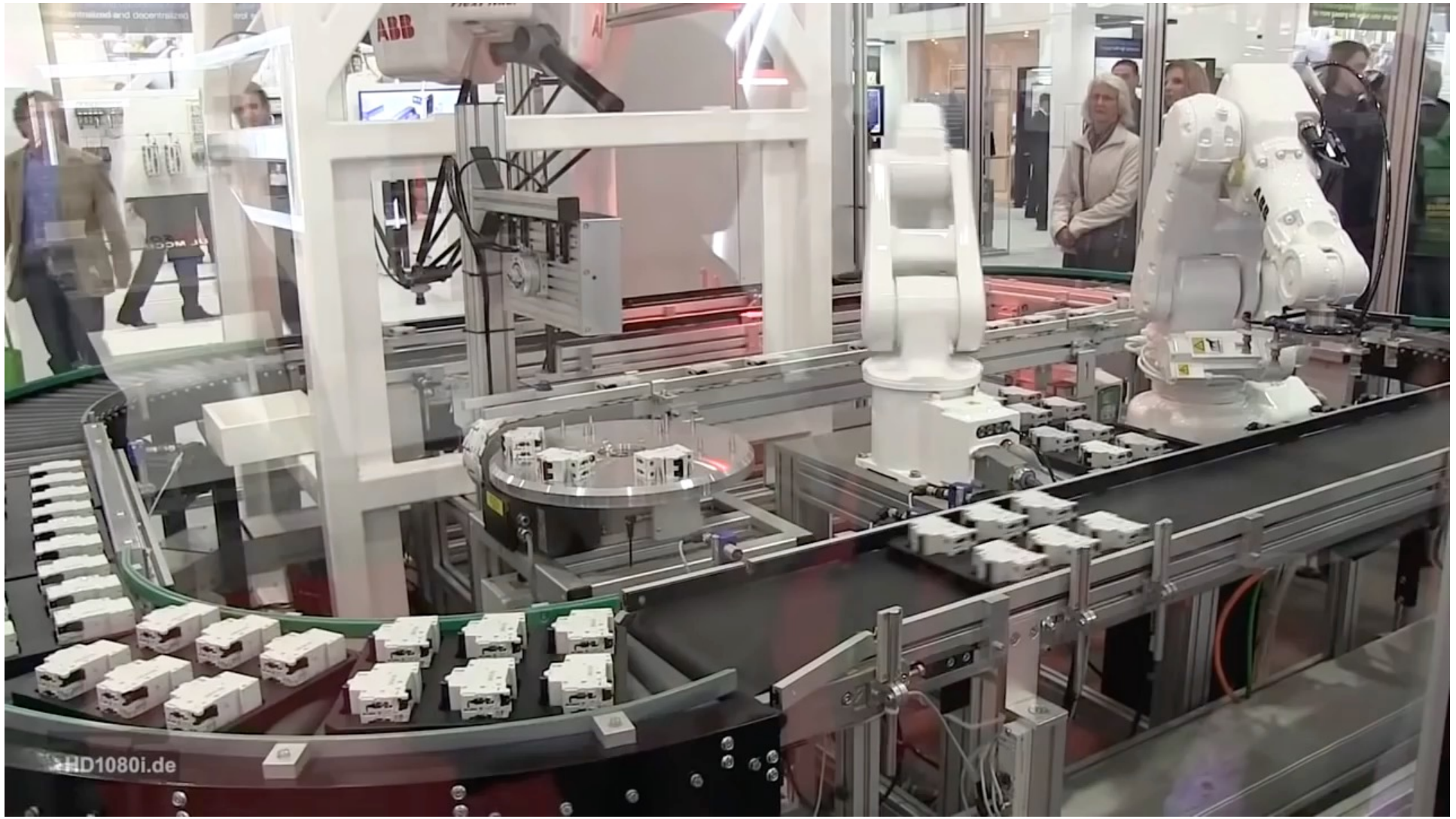


Useful to think of control laws as vector fields



Is this still a research problem?

Industrial robots hard at work



https://www.youtube.com/watch?v=J_8OnDsQVZE&t=315s

Assumptions made by such controllers

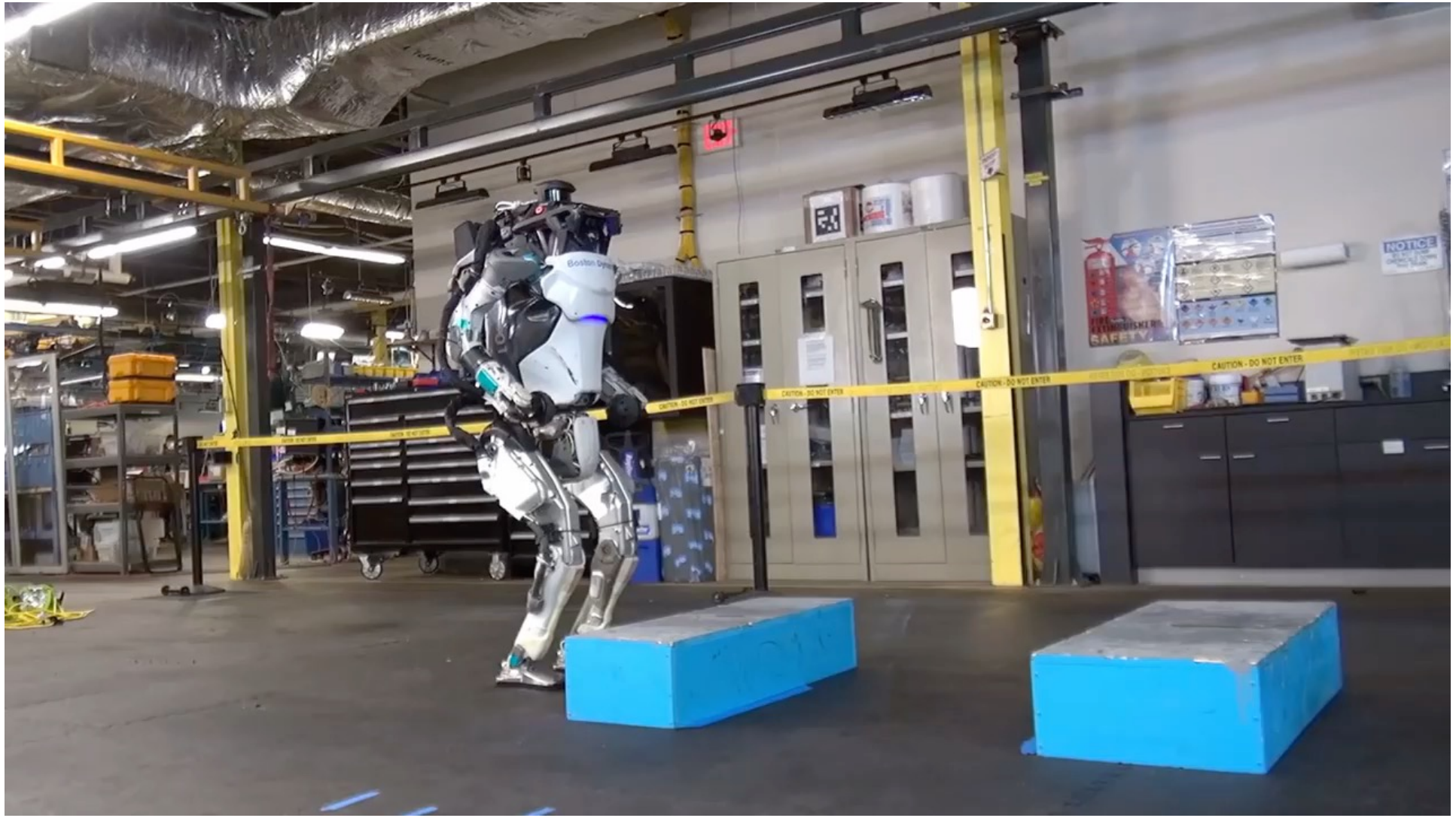
1. Fully actuated: There exists an inverse mapping from reference to control actions

$$\sigma(t) \rightarrow u(t)$$

2. Almost no execution error or state estimation error

3. Enough control authority to clamp down errors / overcome disturbances

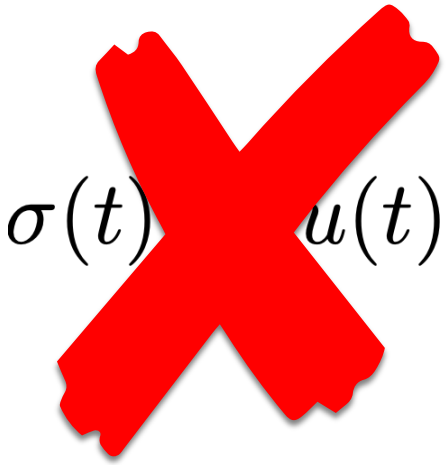
The Atlas robot hard at … play?



https://www.youtube.com/watch?v=fRj34o4hN4I

Challenge 1: Underactuated systems

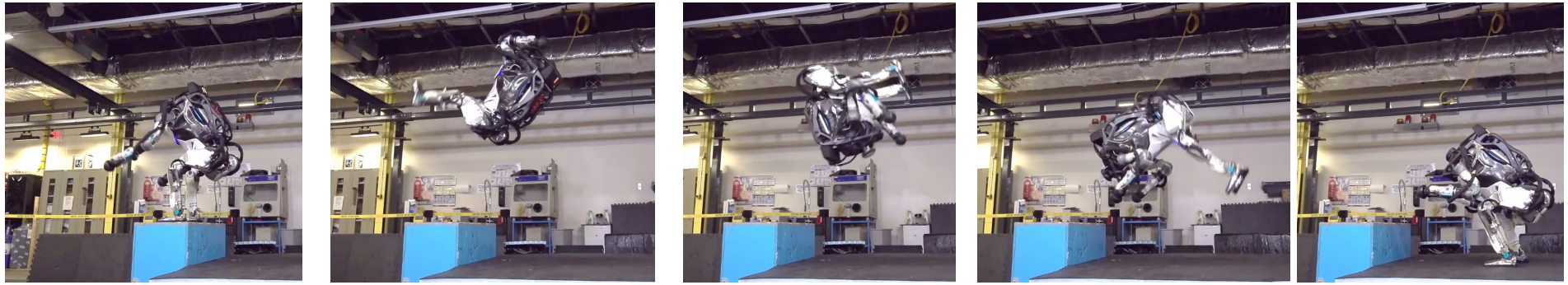
Fully actuated: There exists an inverse mapping from reference to control actions



We don’t have full authority to move the system along arbitrary trajectories

Challenge 1: Underactuated systems

What affects the error between robot state and reference?



Some initial motor thrust …

Whole lot of gravity!

Whole lot of momentum!

… some precise control adjustments

Question:

If we know the model of our robot, can't we solve a huge optimization problem to figure out control?

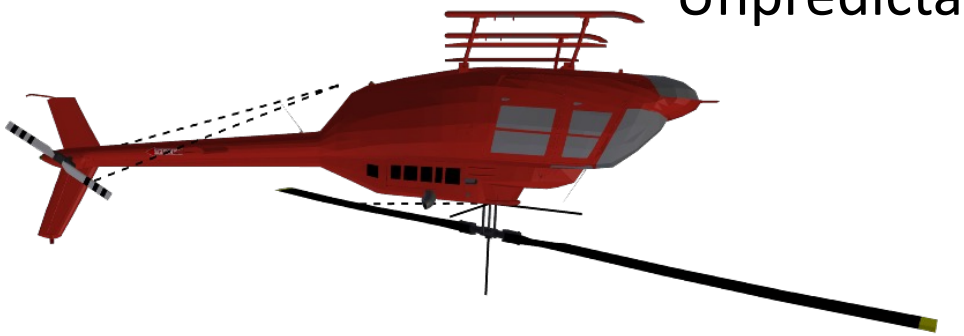
Doing backflips with a helicopter

Redbull Eurocopter BO-105

https://www.youtube.com/watch?v=RGu45s1_QPU

And just what is this model ?!?

Nothing countering gravity!

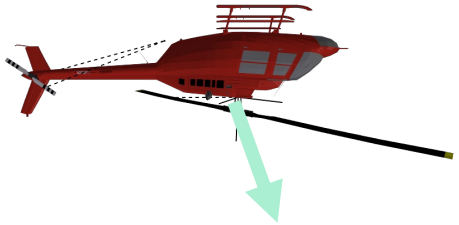


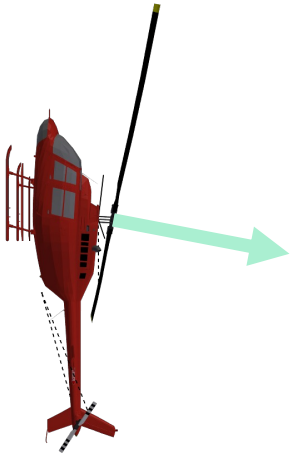
Unpredictable drag forces!

Chaotic vortex around blades!

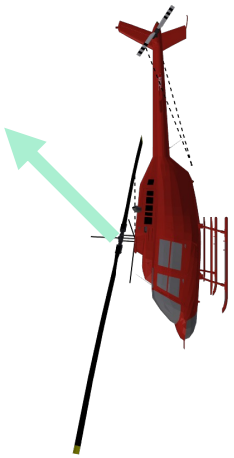
Hopeless to assume we know exactly how the helicopter will behave upside down…

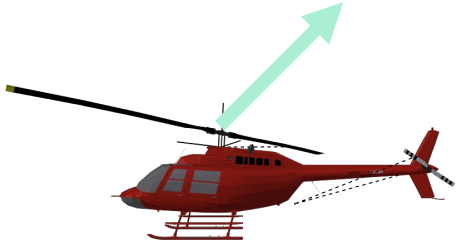
Challenge 2: Choosing good closed-loop models





Closed-loop system =
Point mass
with a planar
thrust vector





Chaotic dynamics + Feedback control law = Well-behaved system

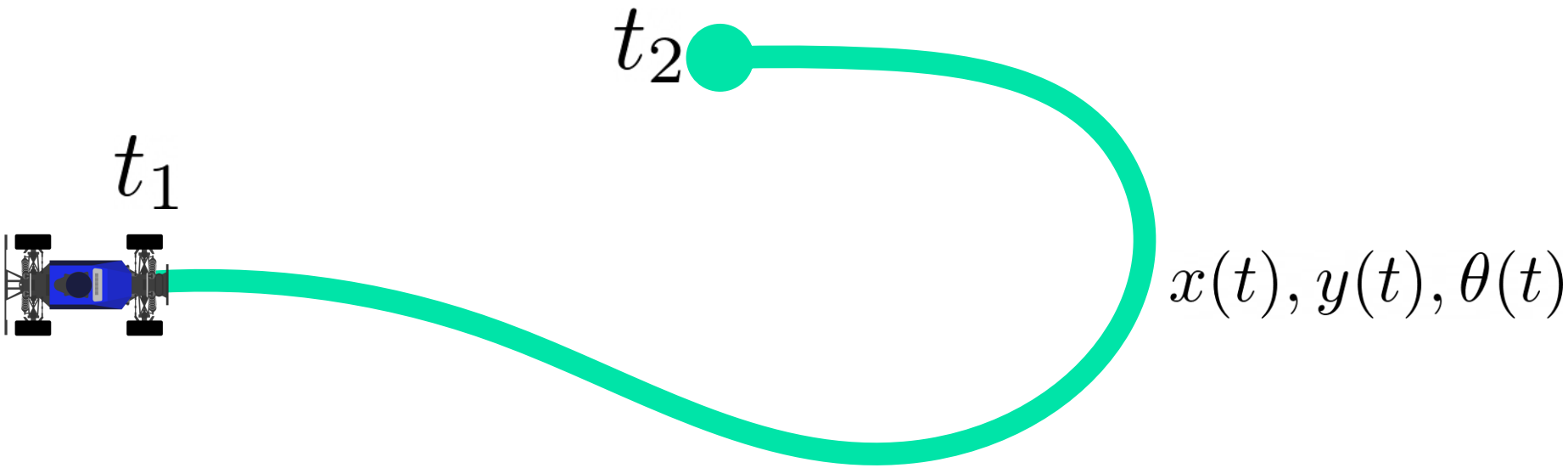
Challenge 3: Model changing on the fly!

Run real-time estimators for wheel characteristics

Need control laws for all possible model parameters

Ok let’s control racecars!

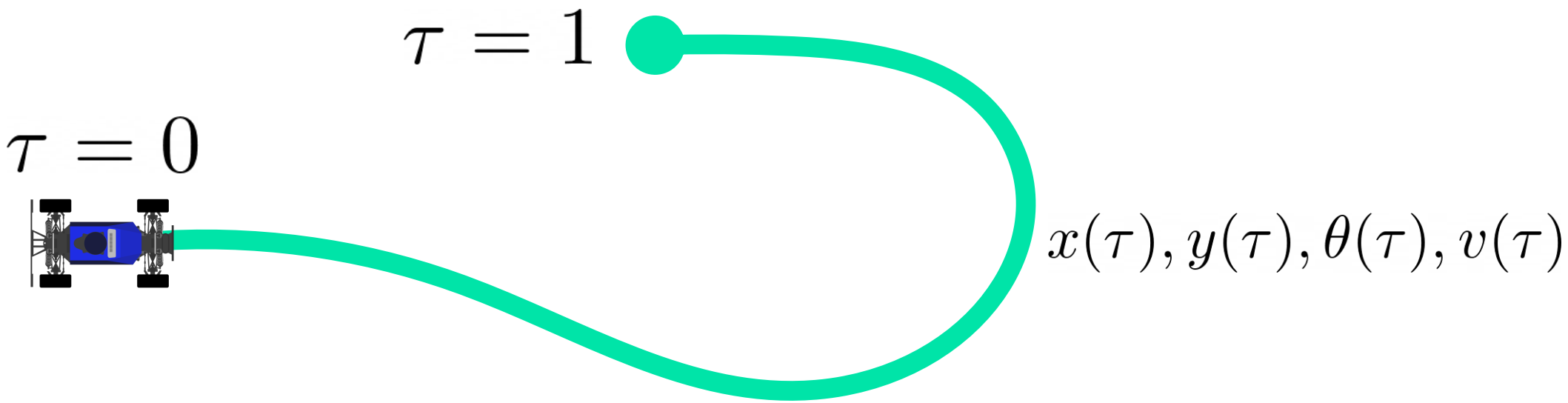
Reference Parameterizations



Option 1: **Time**-parameterized trajectory

Pro: Useful if we want the robot to respect time constraints
Con: Sometimes we only care about deviation from reference

Reference Parameterizations



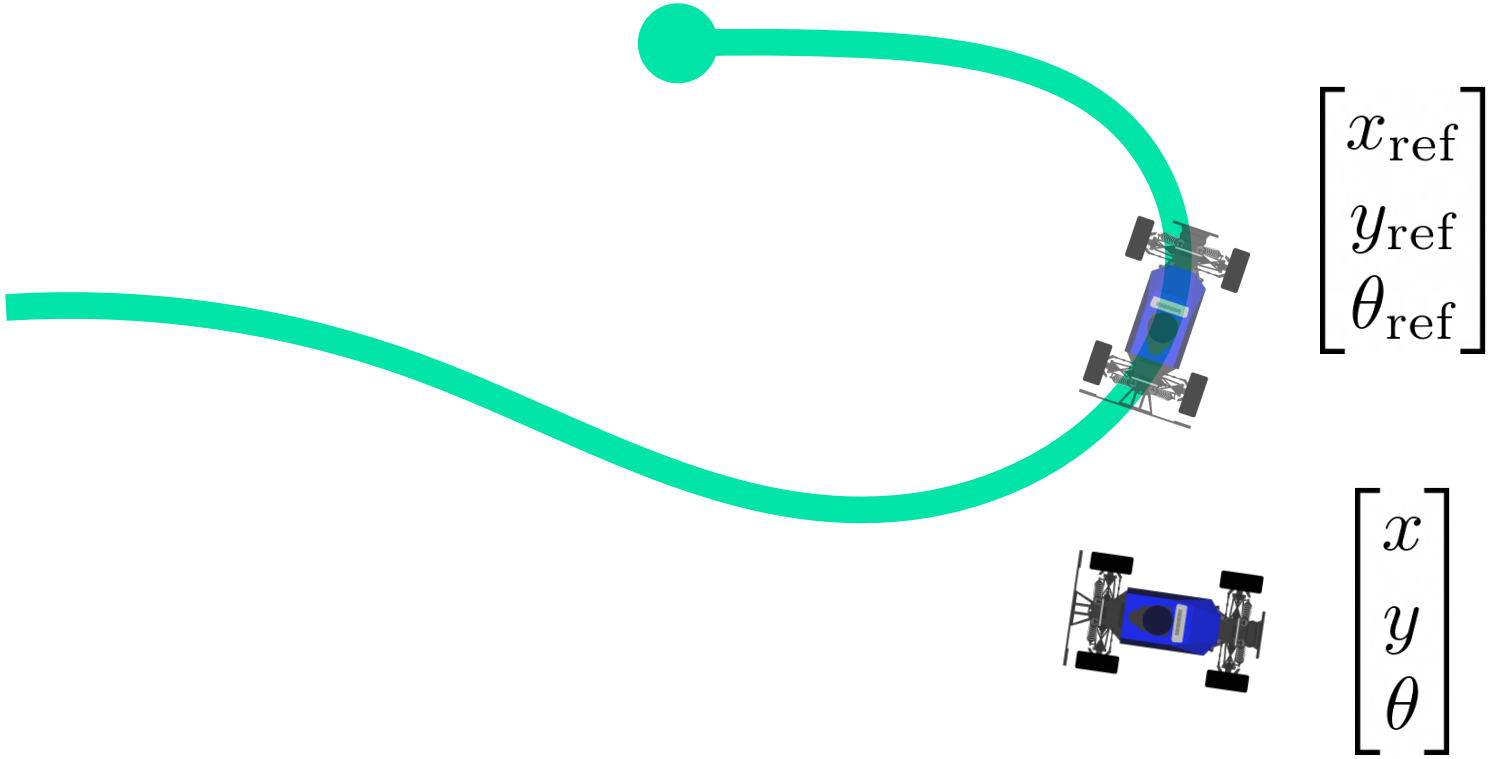
Option 2: **Index**-parameterized geometric path (untimed)

Pro: Useful for conveying shape for the robot to follow
Con: Can’t control when robot will reach a point

Controller Design Decisions

1. Get a reference path/trajectory to track
2. Pick a reference state from the reference path/trajectory
3. Compute error to reference state
4. Compute control law to minimize error

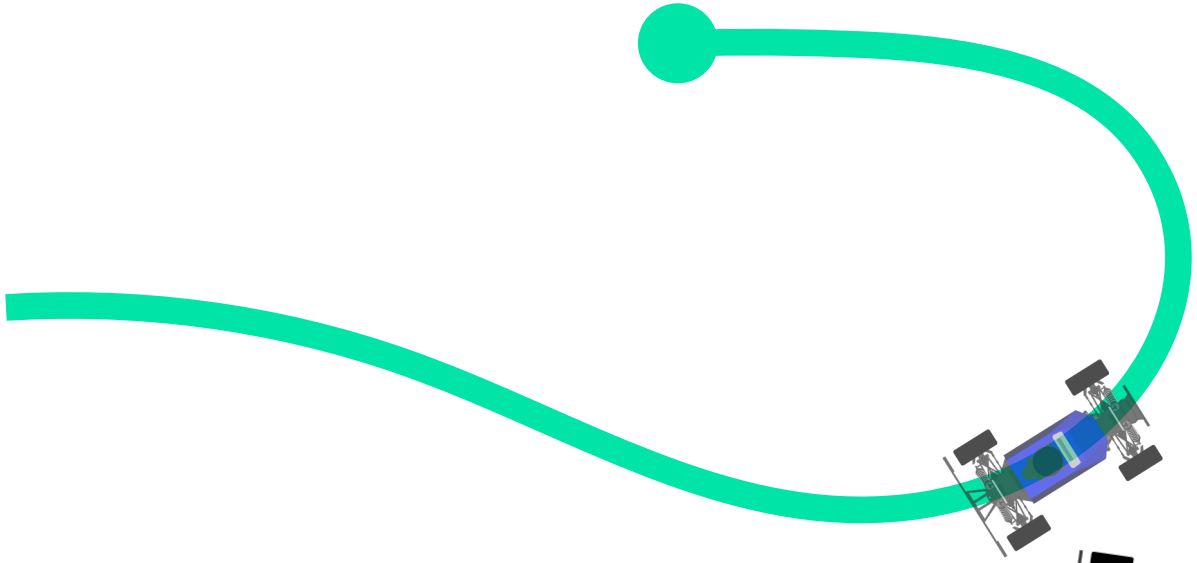
Step 2: Pick a reference (desired) state



$$\begin{bmatrix} x_{\text{ref}} \\ y_{\text{ref}} \\ \theta_{\text{ref}} \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

How do we choose a reference state?



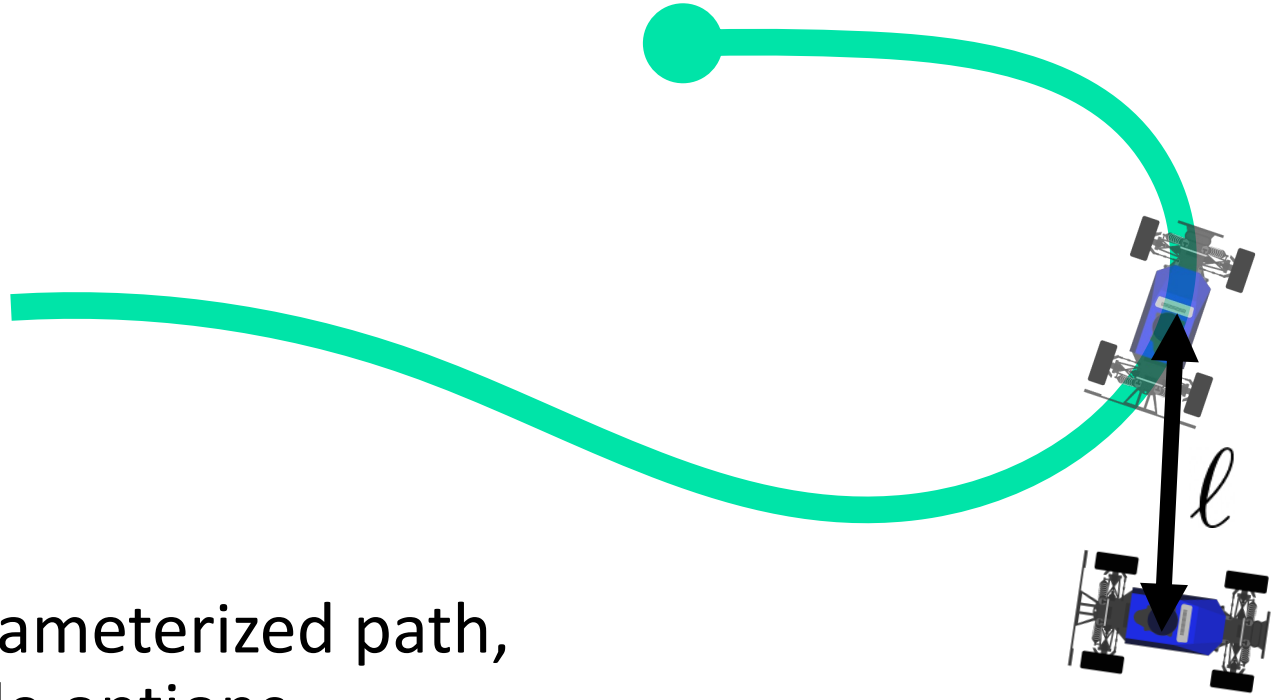
$$\begin{bmatrix} x(\tau_{\text{ref}}) \\ y(\tau_{\text{ref}}) \\ \theta(\tau_{\text{ref}}) \\ v(\tau_{\text{ref}}) \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

For an **index**-parameterized path, there are multiple options.

Closest point $\tau_{\text{ref}} = \arg\min_{\tau} \| \begin{bmatrix} x & y \end{bmatrix}^\top - \begin{bmatrix} x(\tau) & y(\tau) \end{bmatrix}^\top \|$

How do we choose a reference state?



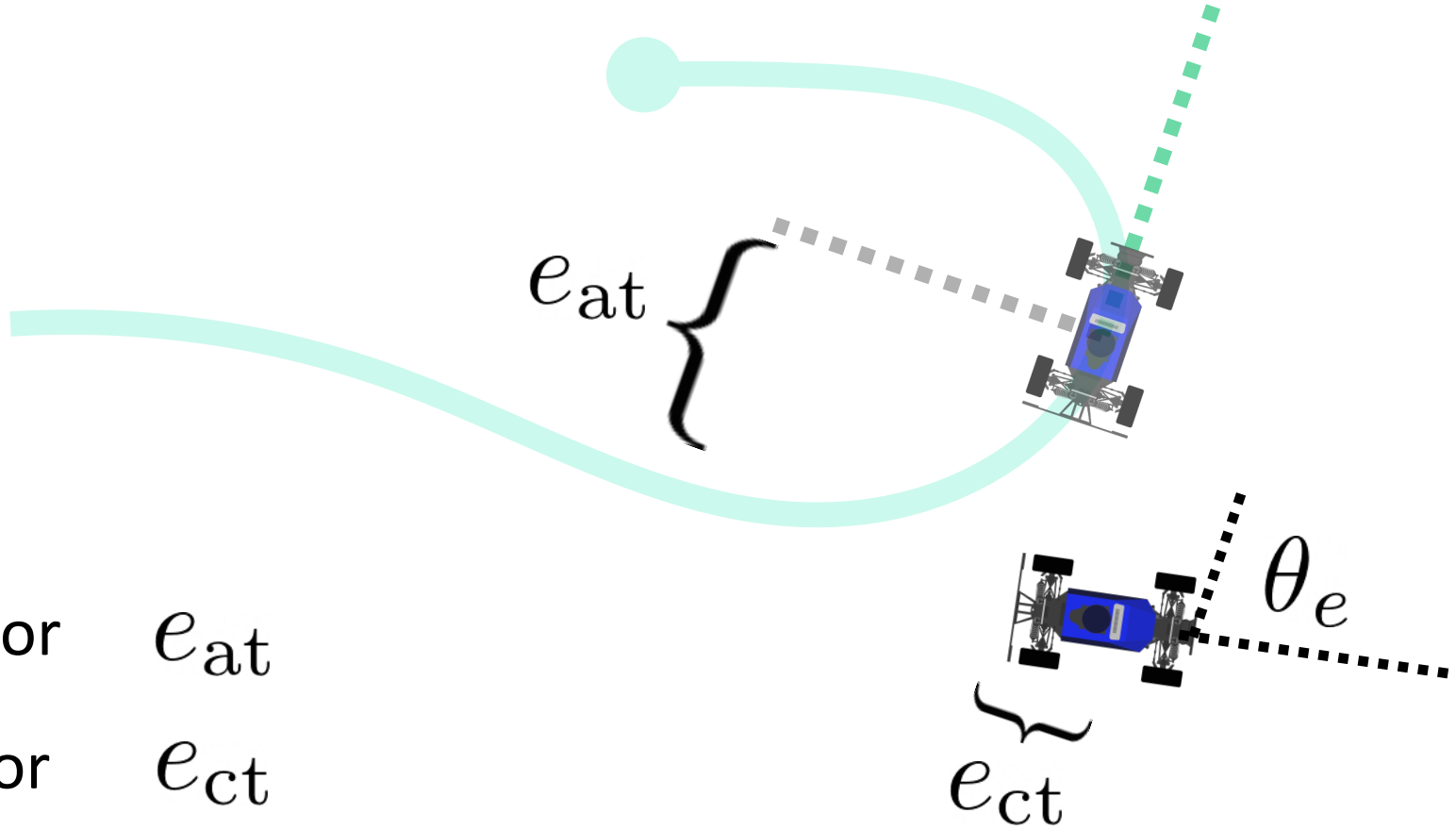
$$\begin{bmatrix} x(\tau_{\text{ref}}) \\ y(\tau_{\text{ref}}) \\ \theta(\tau_{\text{ref}}) \\ v(\tau_{\text{ref}}) \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

For an **index**-parameterized path, there are multiple options.

Lookahead $$\tau_{\text{ref}} = \arg\min_{\tau} \left(\| \begin{bmatrix} x & y \end{bmatrix}^\top - \begin{bmatrix} x(\tau) & y(\tau) \end{bmatrix}^\top \| - \ell \right)^2$$

Step 3: Compute error to reference state

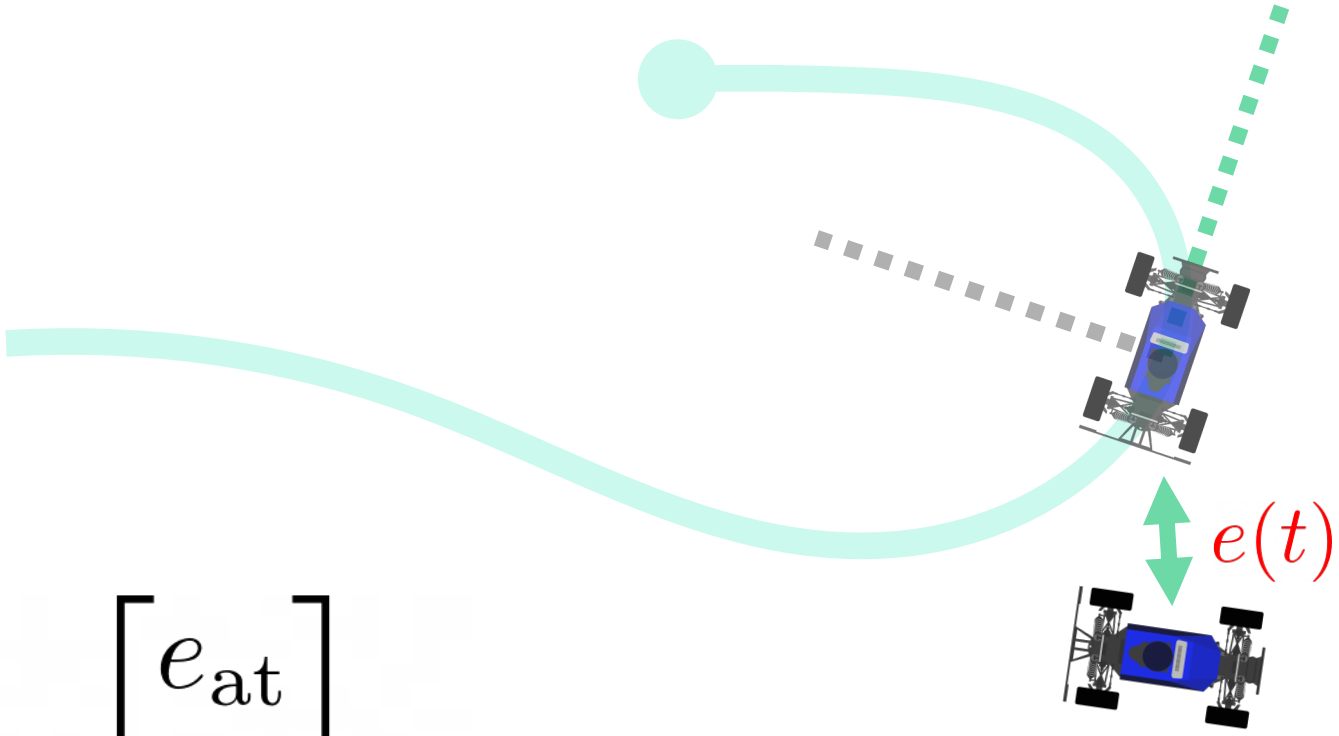


Along-track error e_{at}

Cross-track error e_{ct}

Heading error θ_e

Step 3: Compute error to reference state

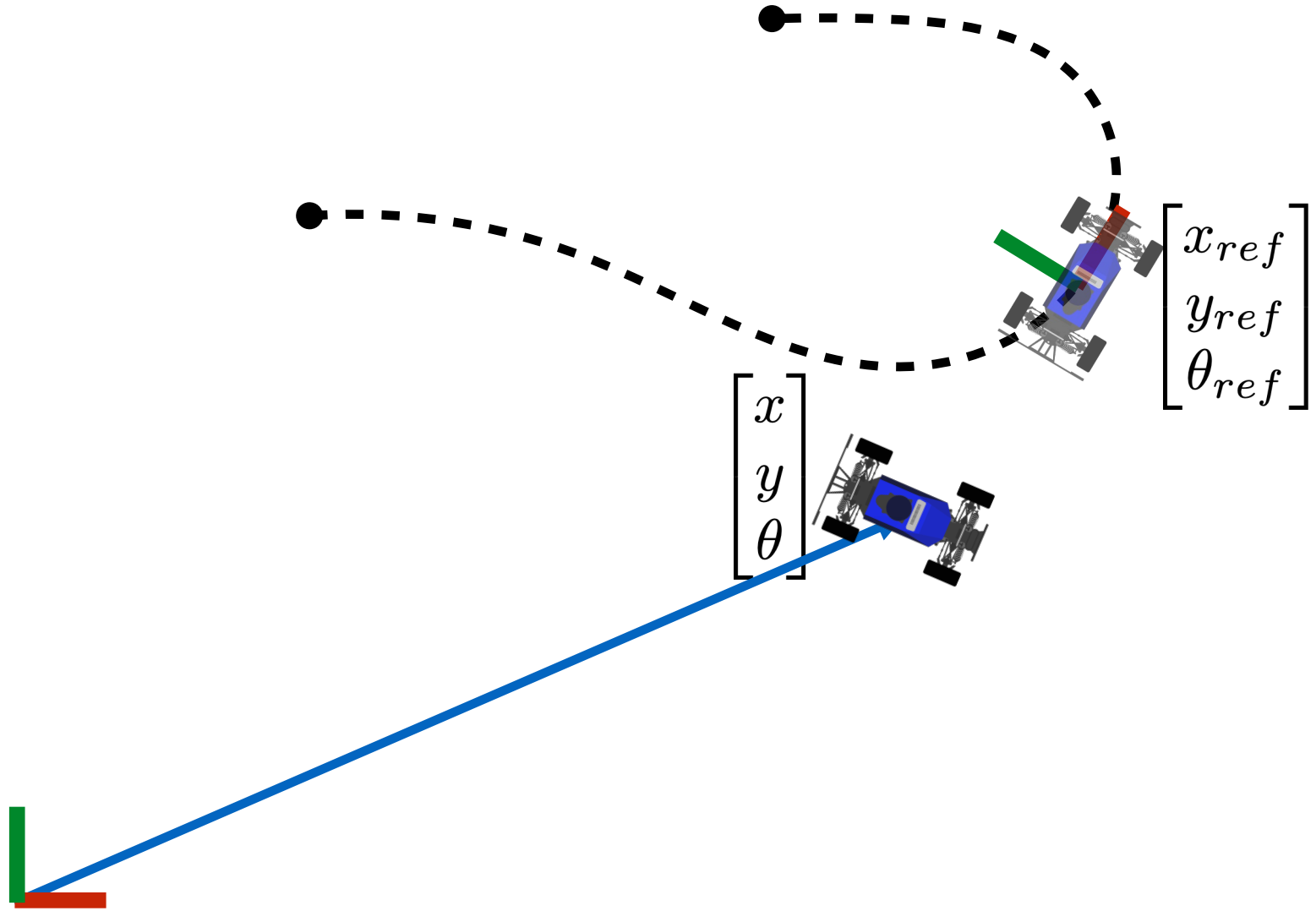


$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_G \rightarrow \begin{bmatrix} e_{\text{at}} \\ e_{\text{ct}} \\ \theta_e \end{bmatrix}_{\text{ref}}$$

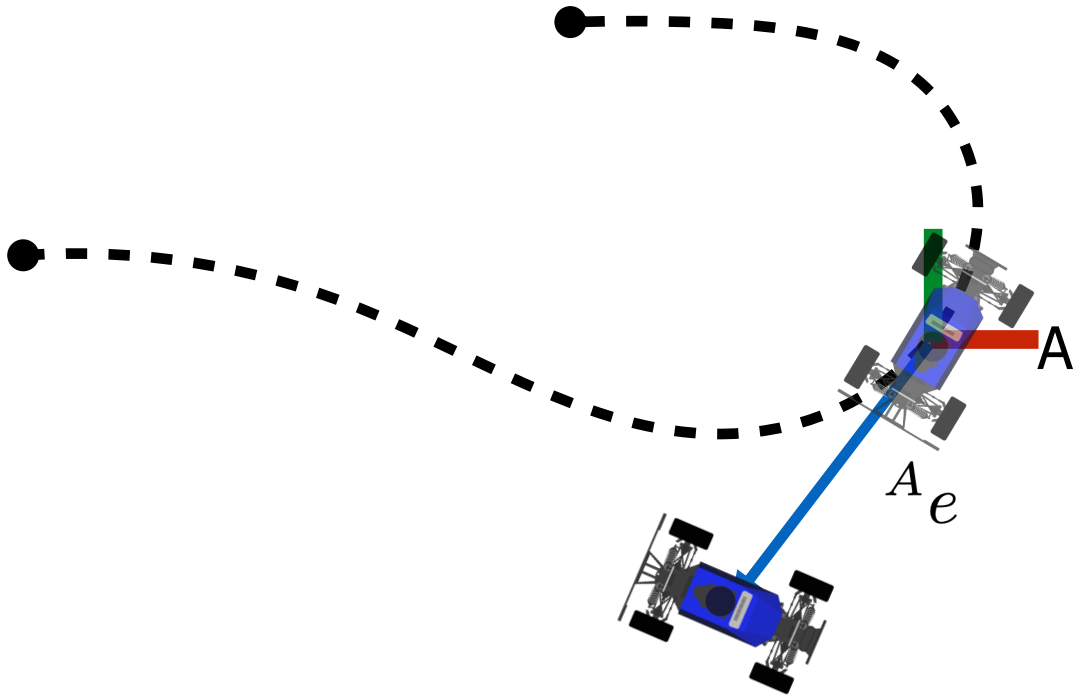
Aside: Rotation Matrices (Plane)

$$R = R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Step 3: Compute error to reference state



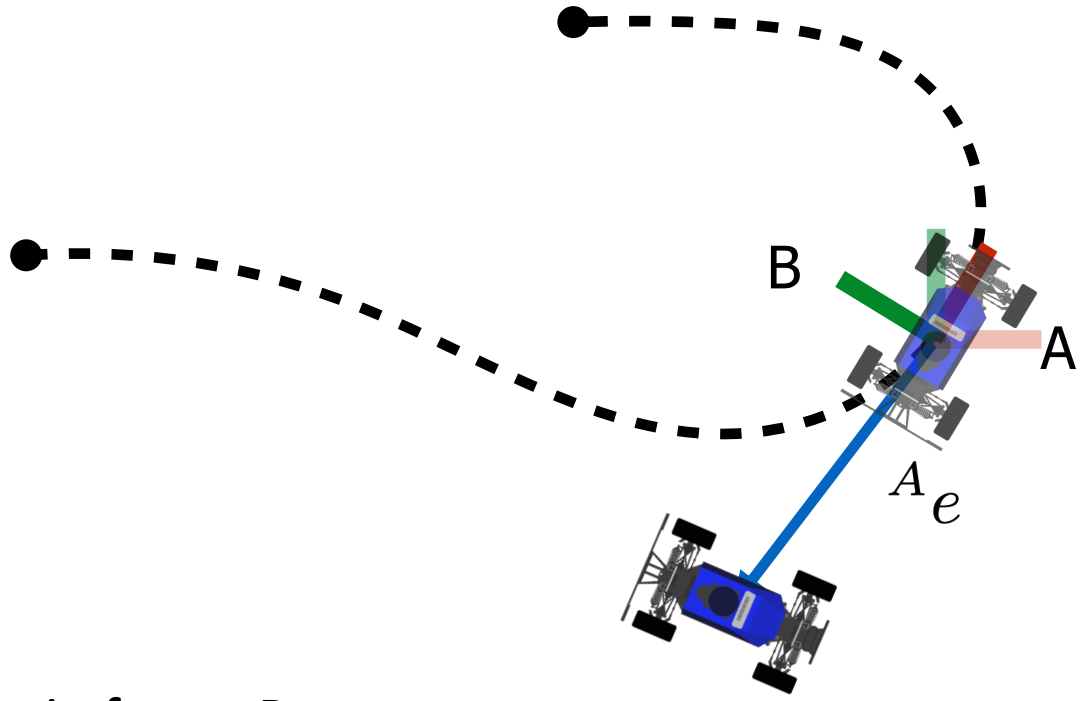
Step 3: Compute error to reference state



Position in frame A

$$^{A}e = \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{ref} \\ y_{ref} \end{bmatrix}$$

Step 3: Compute error to reference state



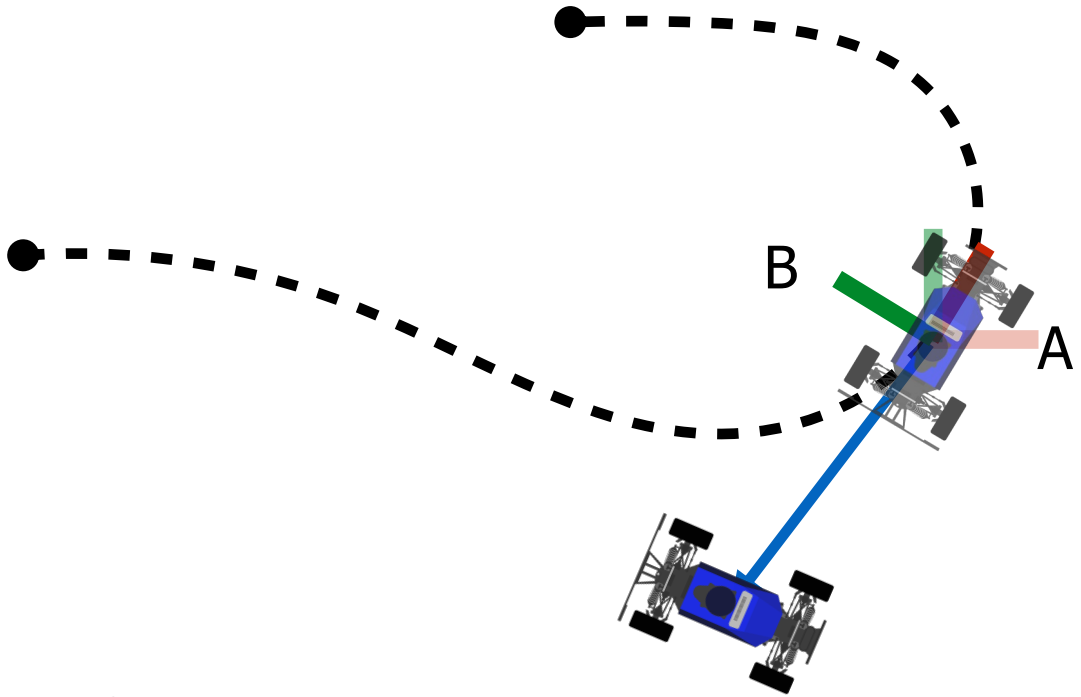
We want position in frame B

$$^{B}e = {}^{B}_{A}R \; {}^{A}e = R(-\theta_{ref}) \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{ref} \\ y_{ref} \end{bmatrix} \right)$$

(rotation of A w.r.t B)

(rotation of A w.r.t B)

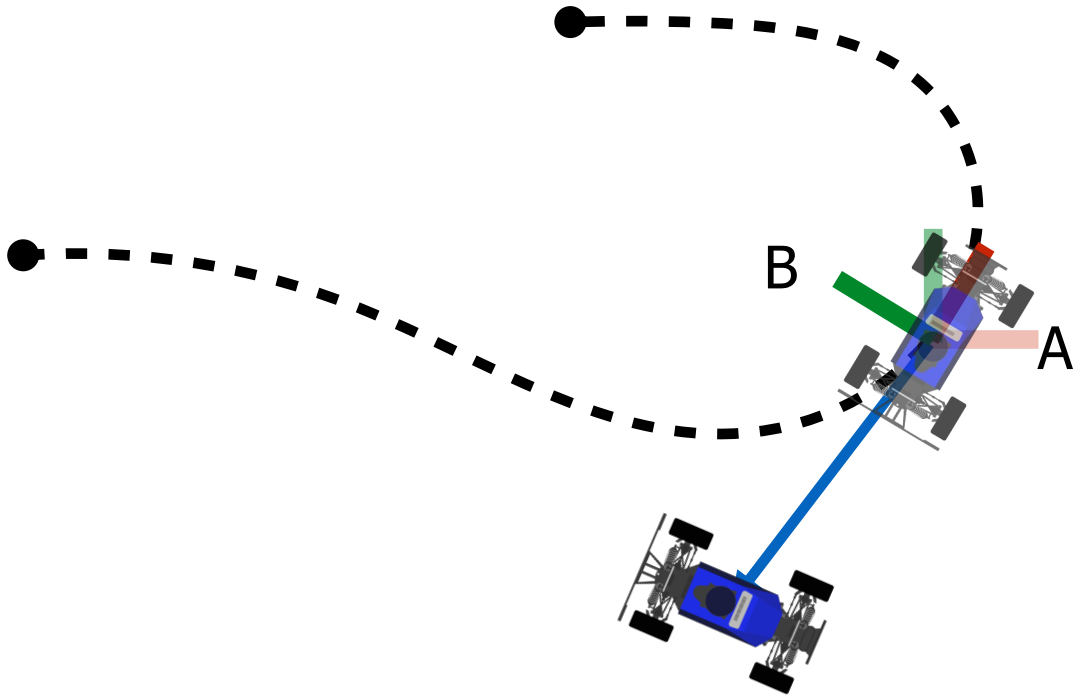
Step 3: Compute error to reference state



We want position in frame B

$$ {}^{B}e = \begin{bmatrix} e_{at} \\ e_{ct} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{ref}) & \sin(\theta_{ref}) \\ -\sin(\theta_{ref}) & \cos(\theta_{ref}) \end{bmatrix} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{ref} \\ y_{ref} \end{bmatrix} \right) $$

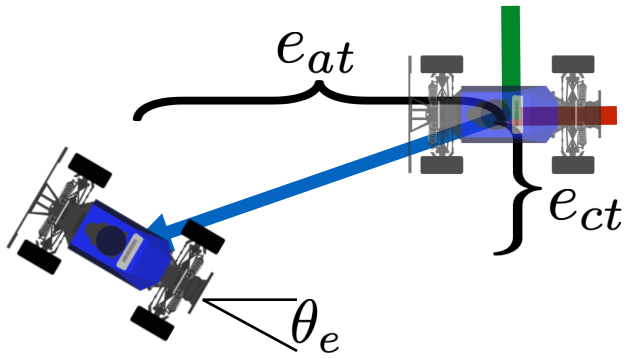
Step 3: Compute error to reference state



Heading error

$$\theta_e = \theta - \theta_{ref}$$

Step 3: Compute error to reference state



(Along-track) $$e_{at} = \cos(\theta_{ref})(x - x_{ref}) + \sin(\theta_{ref})(y - y_{ref})$$

(Cross-track) $$e_{ct} = -\sin(\theta_{ref})(x - x_{ref}) + \cos(\theta_{ref})(y - y_{ref})$$

(Heading) $$\theta_e = \theta - \theta_{ref}$$

Step 4: Compute control law

We will only control steering angle; fixed constant speed

As a result, no real control for along-track error

Some control laws will only minimize cross-track error, others will also minimize heading

$$u = K(e)$$

Different Control Laws

Proportional-integral-derivative (PID) control

Pure-pursuit control

Model-predictive control (MPC)

Linear-quadratic regulator (LQR)

And many many more!

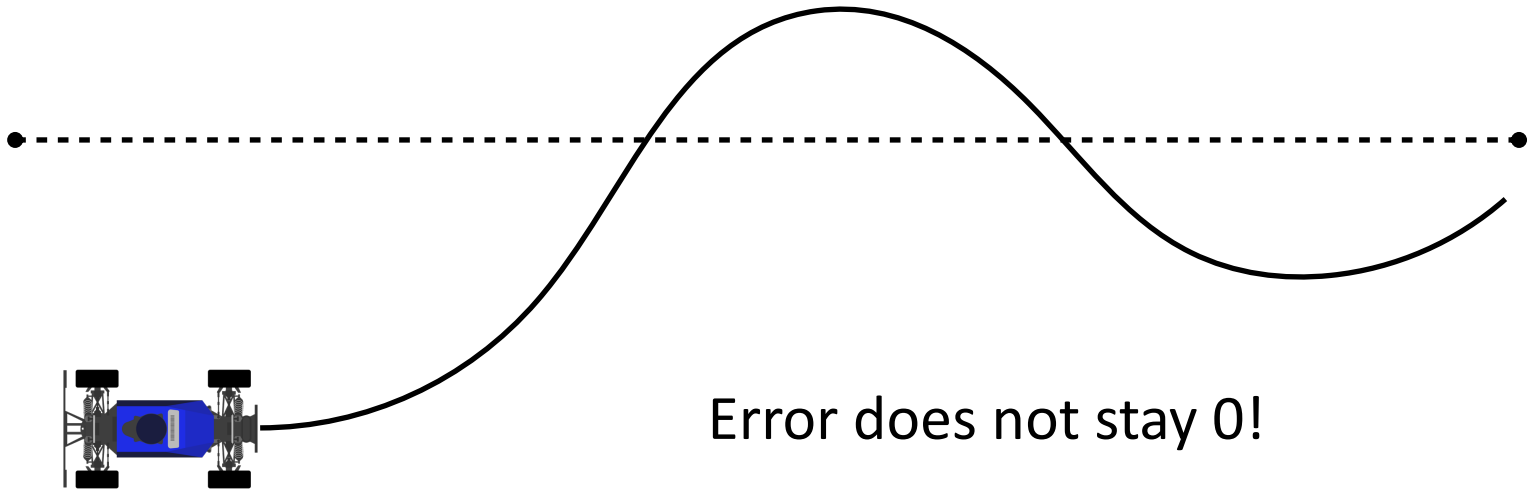
Bang-bang control

Simple control law - choose between hard left and hard right

$$u = \begin{cases} u_{max} & \text{if } e_{ct} < 0 \\ -u_{max} & \text{otherwise} \end{cases}$$

Bang-bang control

What happens when we run this control?



Error does not stay 0!

Need to adapt the magnitude of control proportional to the error …

This clearly sucks! Come back on Monday to find out more