# ESE 680-004
# Learning and Control

**Instructor:** Nikolai Matni

**Class:** Tu/Th 3-4.30pm in Moore 212

**Office hours:** Levine 374, by appointment

**Website:** https://nikolaimatni.github.io/courses/ese680-fall2019/

**E-mail:** nmatni@seas.upenn.edu (start subject line with ESE680)

# Course overview

- We will provide a broad overview of the emerging area at the intersection of machine learning and control, with a focus on *continuous* control problems.

- **Tentative topics list:** system identification, statistical learning theory, model based control of learned systems, model free methods, safe learning and control.

- **Official prereqs:** ESE 500 (Linear Systems) and one of CIS 520 (Machine Learning) or ESE 605 (Convex Optimization).

- **Unofficial prereqs:** mathematical maturity in linear algebra, functional analysis, probability theory, and optimization. Exposure to control and machine learning a plus.

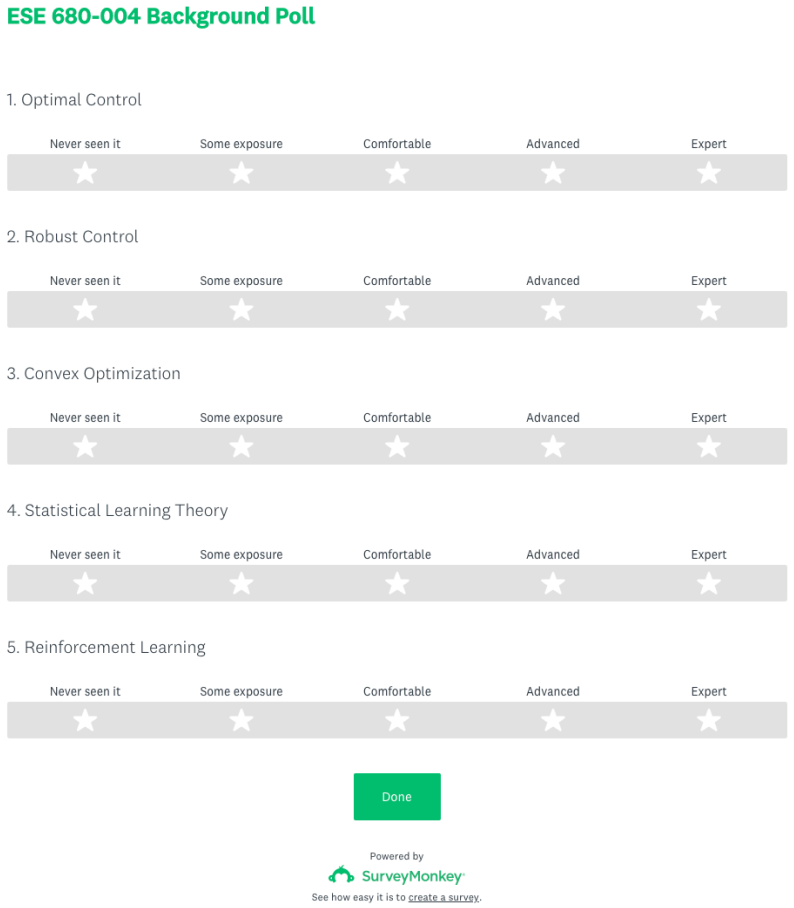
- **Main focus:** algorithms/results with *strong theoretical guarantees* about stability, performance, robustness, and sample efficiency.

# Background poll

Please fill out background poll at
https://www.surveymonkey.com/r/XFSR26N

**If you are sitting in, please still fill this out.**

**Also please send me an e-mail so I can add you to class mailing list.**



# Logistics and grading

- **Class structure:** Roughly 50% lectures, 50% student led discussions
- **Grading:** this is a graduate level topics class, so if you show up prepared, participate, and take the assignments seriously, you will get an A.
- **Assignments:** roughly, each of you will have to scribe one lecture, and/or present one paper in class (this may change depending on numbers)
- **Project:** done individually or in groups of two. Students are encouraged to propose a topic that connects class material to aspects of their research.
  - Proposal (1-2 pages), Midterm update (2-3 pages), Final presentation, Final report (6-8 pages).
  - Meant to mimic a conference submission: intro, lit review, problem formulation, main results, experiments, discussion/conclusion

# Scribing

- Google doc signup sheet: https://docs.google.com/spreadsheets/d/1Kqa-ZnnqO-r7-c0rTKNSaCNiz_hTQARH9V_Ug8blp3Q/edit?usp=sharing
- Produce a high quality set of notes
- Go beyond what was presented in class: summarize both the lecture and the required reading. Document should be self contained.
- Use LaTeX template from course website
- **If you are sitting in, you may be asked to scribe, depending on numbers**

# Paper presentations

- Google doc signup sheet: https://docs.google.com/spreadsheets/d/1Kqa-ZnnqO-r7-c0rTKNSaCNiz_hTQARH9V_Ug8blp3Q/edit?usp=sharing

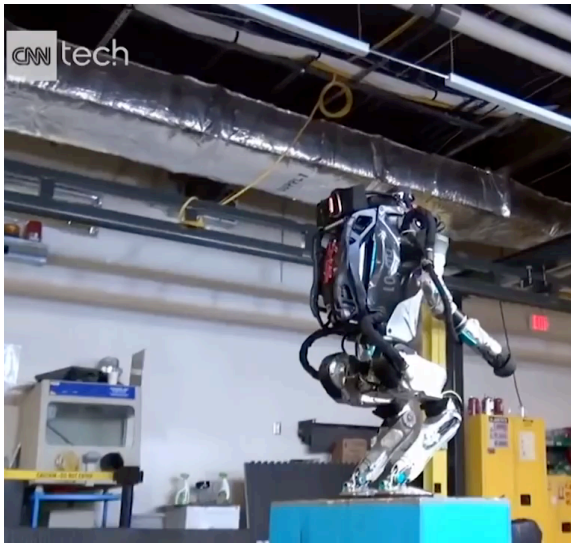
- Sign up to present a paper at least 2 weeks in advance, and then e-mail to schedule a meeting with me 1 week in advance to go over your presentation. Plan for about 45min of presenting and 45min of discussion.

- **Problem statement:** what problem is the paper solving, and why is it meaningful?
- **Prior work:** how does this paper fit into the current research landscape.
- **Key idea:** what is the main takeaway from the paper?
- **Key technical tools:** what are the main technical contributions of the paper? Be prepared to present and explain these in detail.
- **Points of confusion:** are there technical or conceptual arguments that are unclear or incomplete? Bring these up so that we can try to work through them in class.
- **Shortcomings/areas for improvement:** what are some of the flaws of the paper? What are possible directions for future work to address these?

- **If you are sitting in, you may be asked to present, depending on numbers**

# Motivation and hand-wavy overview

# The future is now

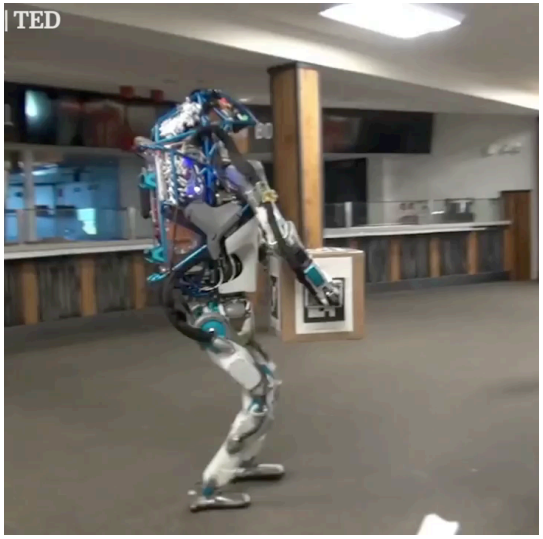



Boston Dynamics

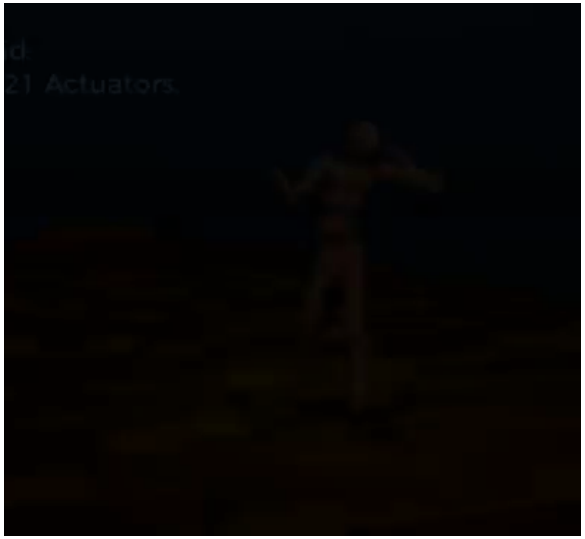
Open AI

Waymo

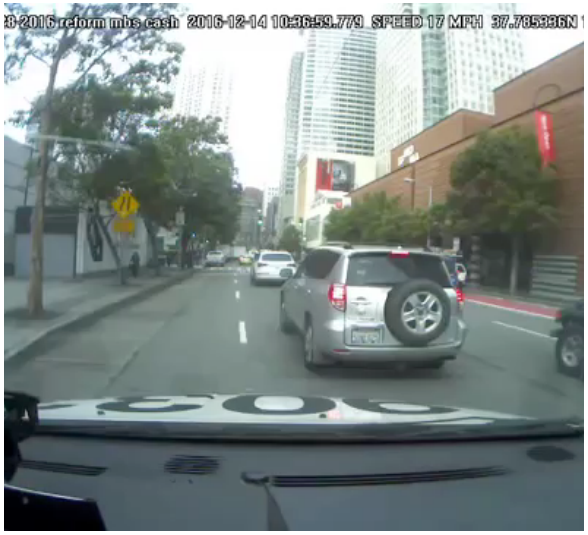
# The future is now -ish




Boston Dynamics

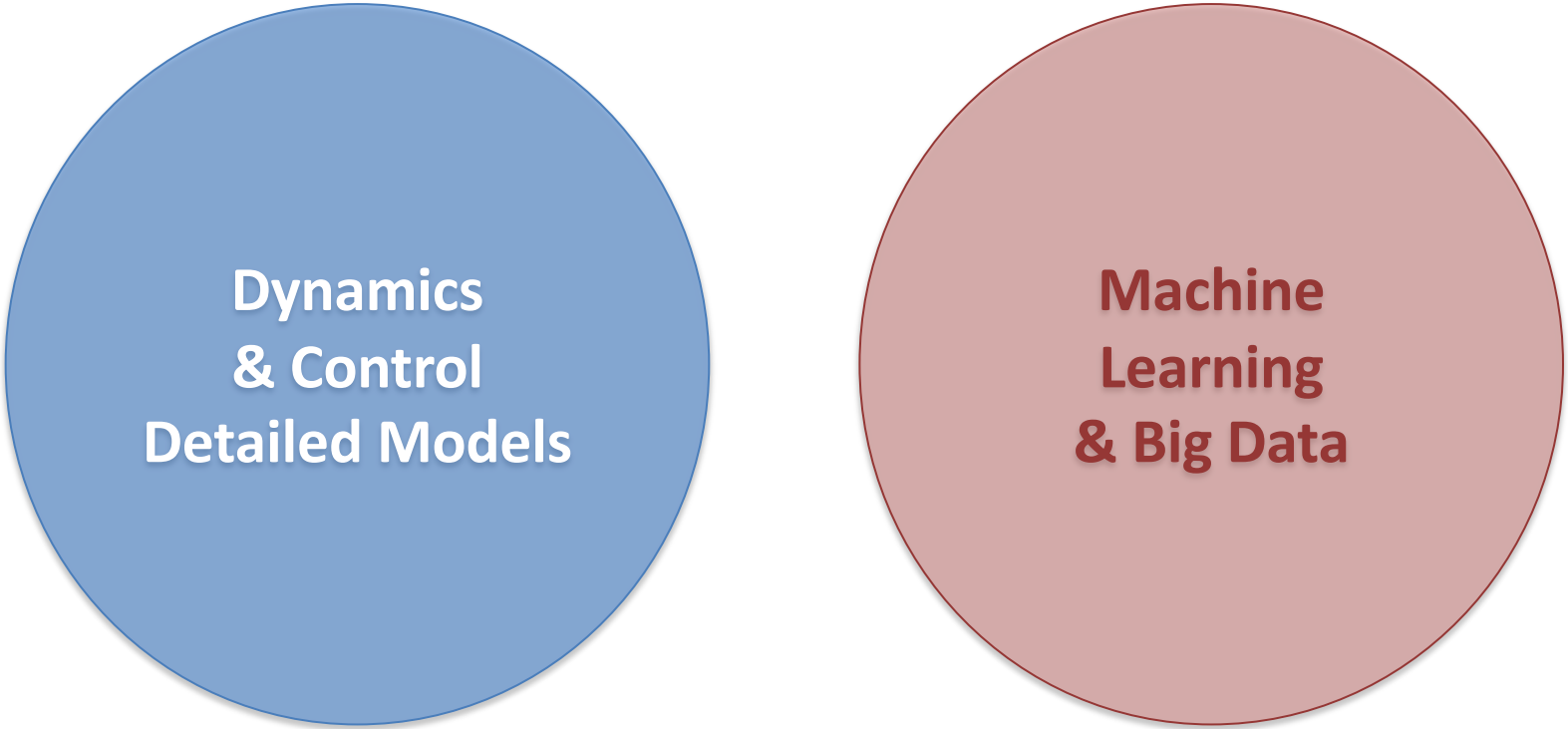


DeepMind

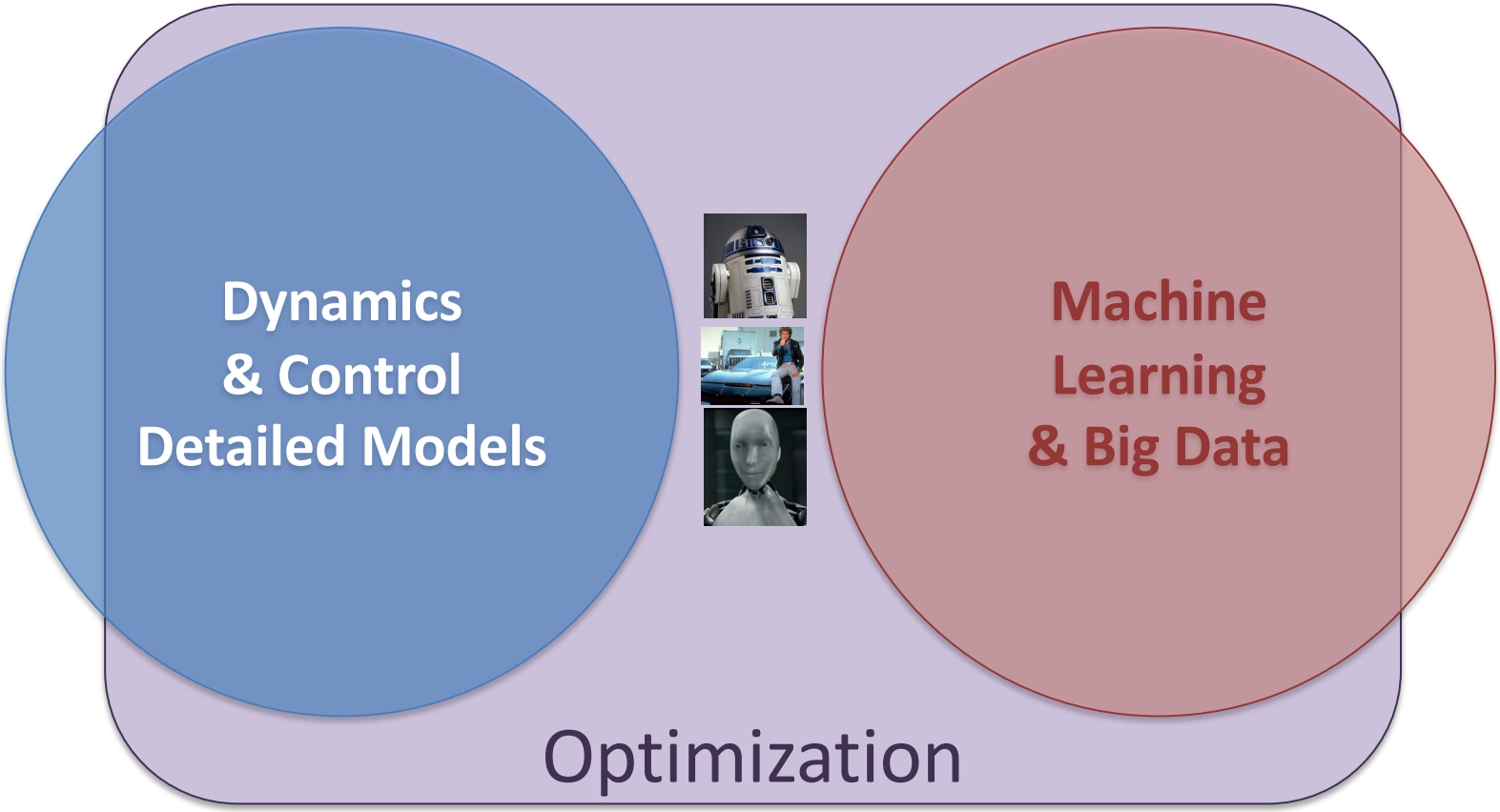


Uber running a red

# How do we fix things?



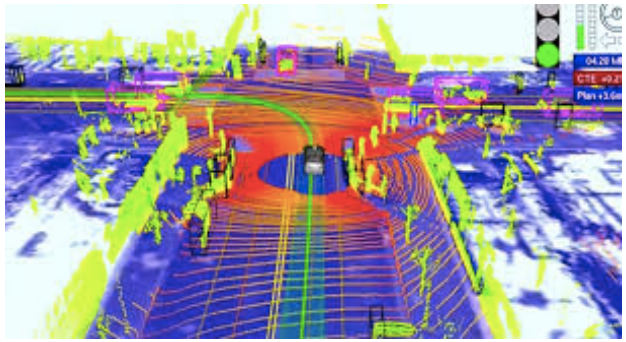
# How do we fix things?



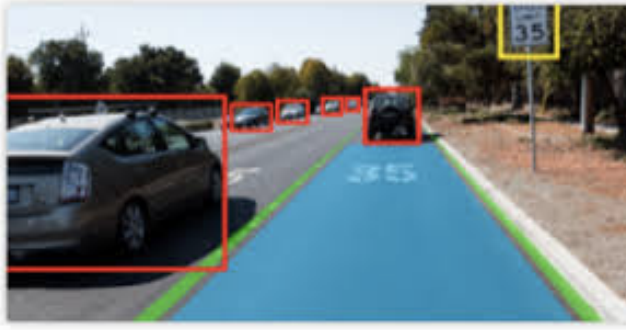
# What is ML? Why do we need it?

using past **data** to **learn** about and/or **act** upon the world

Environments
too complex



Sensing
too complex

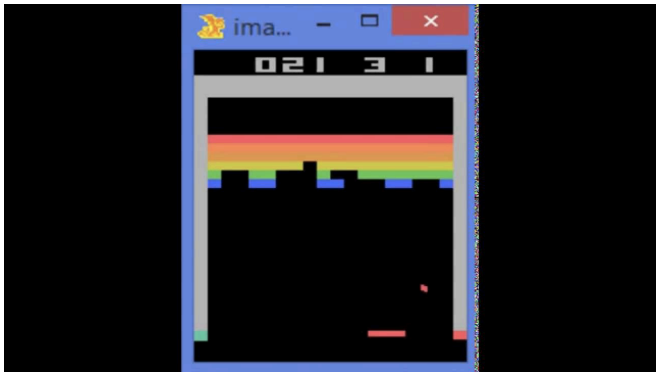


Models
too complex

**Guarantees** are (usually) **probabilistic**

# We don’t need models!


DeepMind AlphaGo




DeepMind DQN

DeepMind AlphaZero

# We don’t need models!


Goldberg lab, UC Berkeley

Google DeepMind

Open AI

**Controlled settings with no cost to failure**

*Uber's Self-Driving Cars Were Struggling Before Arizona Crash*

**Tesla Says Autopilot Was Engaged in Fatal Crash Under Investigation in California**

Vehicle's system shows driver had hands off the wheel for six seconds before striking highway divider

Las Vegas' self-driving bus crashes in first hour of service

**Google AI looks at rifles and sees helicopters**

**Street sign hack fools self-driving cars**

**ML deployed in the wild needs guarantees of stability, performance, robustness, safety**

# Robustness through feedback

$$m\left[\frac{du}{dt} + qw - rv\right] = F_x - mg\sin\Theta$$

$$m\left[\frac{dv}{dt} + ru - pw\right] = F_y + mg\cos\Theta\sin\Phi$$

$$m\left[\frac{dw}{dt} + pv - qu\right] = F_z + mg\cos\Theta\cos\Phi$$

$$M_x = I_{xx}\frac{dp}{dt} - I_{xz}\frac{dr}{dt} + [I_{zz} - I_{yy}]qr - I_{xz}pq$$

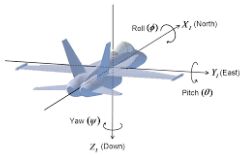
$$M_y = I_{yy}\frac{dq}{dt} + [I_{xx} - I_{zz}]rp + I_{xz}[p^2 - r^2]$$

$$M_z = -I_{xz}\frac{dp}{dt} + I_{zz}\frac{dr}{dt} + [I_{yy} - I_{xx}]pq + I_{xz}q$$

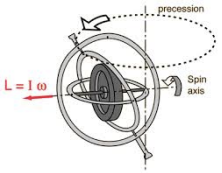
$$p = \frac{d\Phi}{dt} - \frac{d\Psi}{dt}\sin\Theta$$

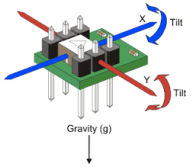
$$q = \frac{d\Theta}{dt}\cos\Phi + \frac{d\Psi}{dt}\sin\Phi\cos\Theta$$

$$r = -\frac{d\Theta}{dt}\sin\Phi + \frac{d\Psi}{dt}\cos\Phi\cos\Theta$$




**feedback to close the loop**




**Controller**

**measurements**

**control action**

**Actuators**

**Sensors**

**disturbances**

**reference**

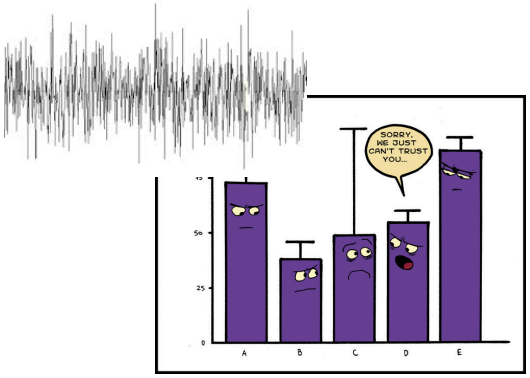
**system**

# What is Robust Control? Why do we need it?

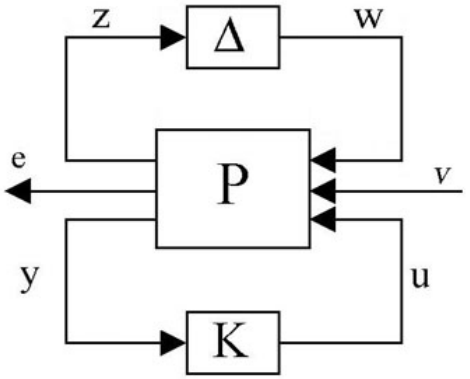
using **feedback** to **mitigate** the effects of **dynamic uncertainty**

Environments
are uncertain

Sensing/components
are uncertain




Models
are uncertain



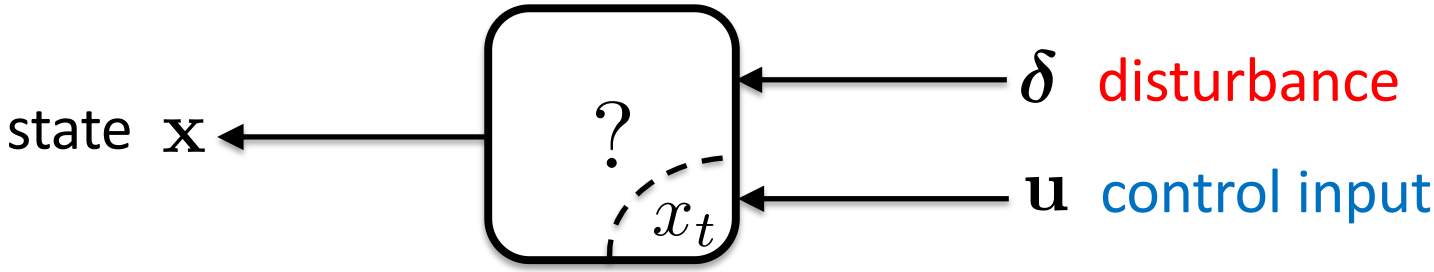

**Guarantees** are (usually) **worst-case and deterministic**

# Robustness and Learning?

| **Machine Learning** | **Robust Control** |
| --- | --- |
| uses data to **reduce uncertainty** | uses feedback to **mitigate uncertainty** |
| more data → better models/predictions | better models/predictions → better performance |
| **probabilistic guarantees** | **worst-case guarantees** |

Can ML and RC be combined so that we **safely achieve** more data → better performance?

# Optimal control of an unknown system




**Goal:**

$$\text{minimize}_{\pi_t} \quad \text{Cost}_\delta(\boldsymbol{x}, \boldsymbol{u})$$

**unknown**

$$\text{s.t} \quad x_{t+1} = f_t(x_t, u_t, \delta_t)$$

**state-feedback** $u_t = \pi_t(x_{0:t}, u_{0:t-1})$

# Model Based RL & Control

Uncertainty inherent to the output of a ML algorithm

How do we (safely) learn a model and quantify its uncertainty?

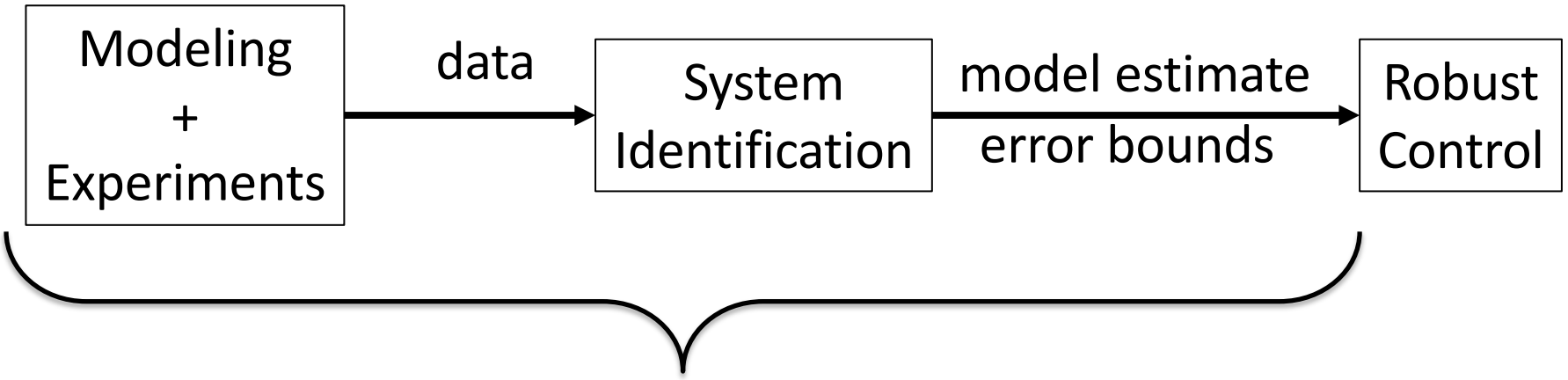
*System Identification, Concentration Bounds, Bootstrapping*

How do we explicitly account for this uncertainty when we design control policies?

*Robust and optimal control, end-to-end analysis*

# A learning and control pipeline

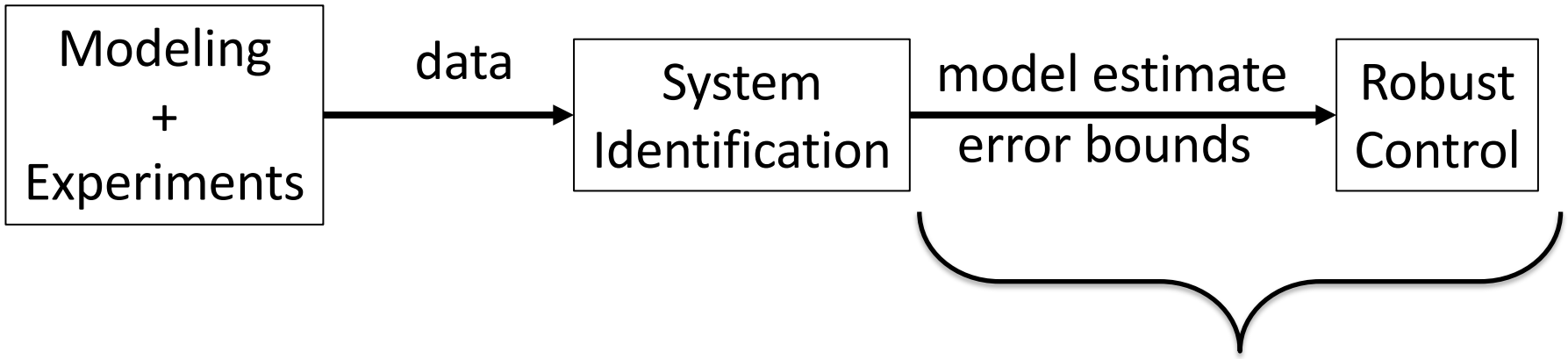
# A learning and control pipeline




**ML & Uncertainty Quantification**

show error bounds hold ***with high probability***

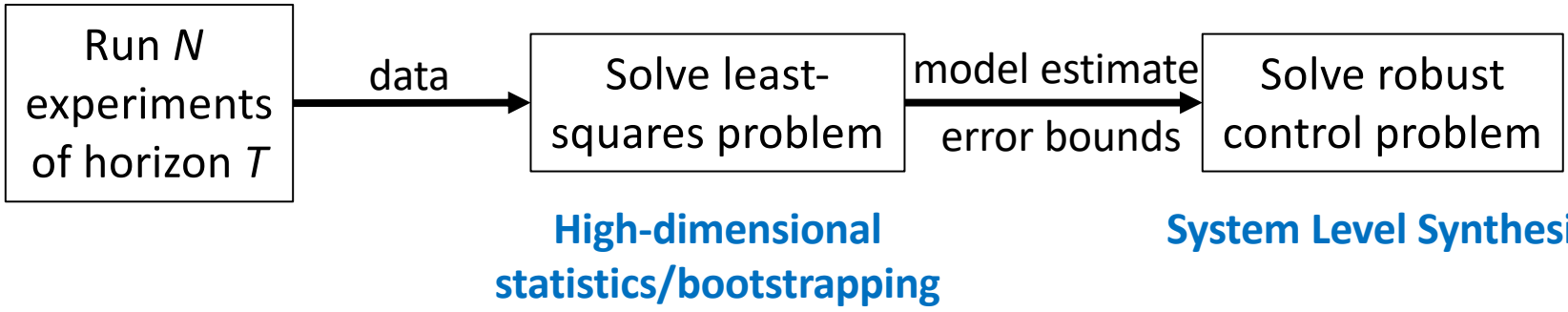
# A learning and control pipeline




**RC to Handle Uncertainty**

assume error bounds hold ***but are adversarial***

# End-to-end guarantees



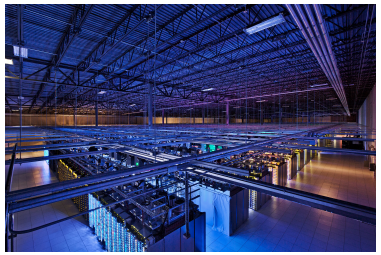

**Theorem**

With probability $1-\delta$, for $N$ sufficiently large, the synthesized controller is stabilizing and achieves the relative performance bound

$$\frac{\hat{J}-J_\star}{J_\star} \leq \mathcal{C}\,(\text{robustness}, \text{excitability})\sqrt{\frac{(d+p)\log(1/\delta)}{N}}$$

# of states # of inputs

# Example: Cooling a server farm



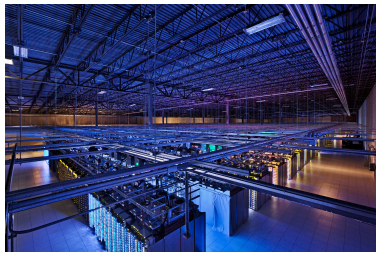
$$x_{t+1} = \begin{bmatrix} 1.01 & 0.01 & 0 \\ 0.01 & 1.01 & 0.01 \\ 0 & 0.01 & 1.01 \end{bmatrix} x_t + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} u_t + \delta_t$$

**temperature deviations**

**cooling**

**"noise"**

# Example: Cooling a server farm

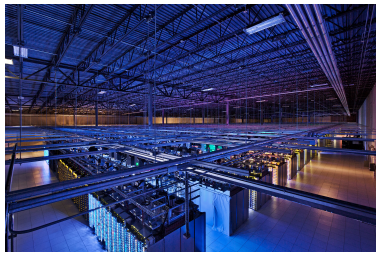


$$x_{t+1} = \begin{bmatrix} 1.01 & 0.01 & 0 \\ 0.01 & 1.01 & 0.01 \\ 0 & 0.01 & 1.01 \end{bmatrix} x_t + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} u_t + \delta_t$$

**Goal:** $$\min_{u_0, u_1, \ldots} \frac{1}{T} \sum_{t=0}^{T} \mathbb{E}\left[x_t^T Q x_t + u_t^T R u_t\right]$$

**don't burn servers** **don't burn $$$**

# Example: Cooling a server farm

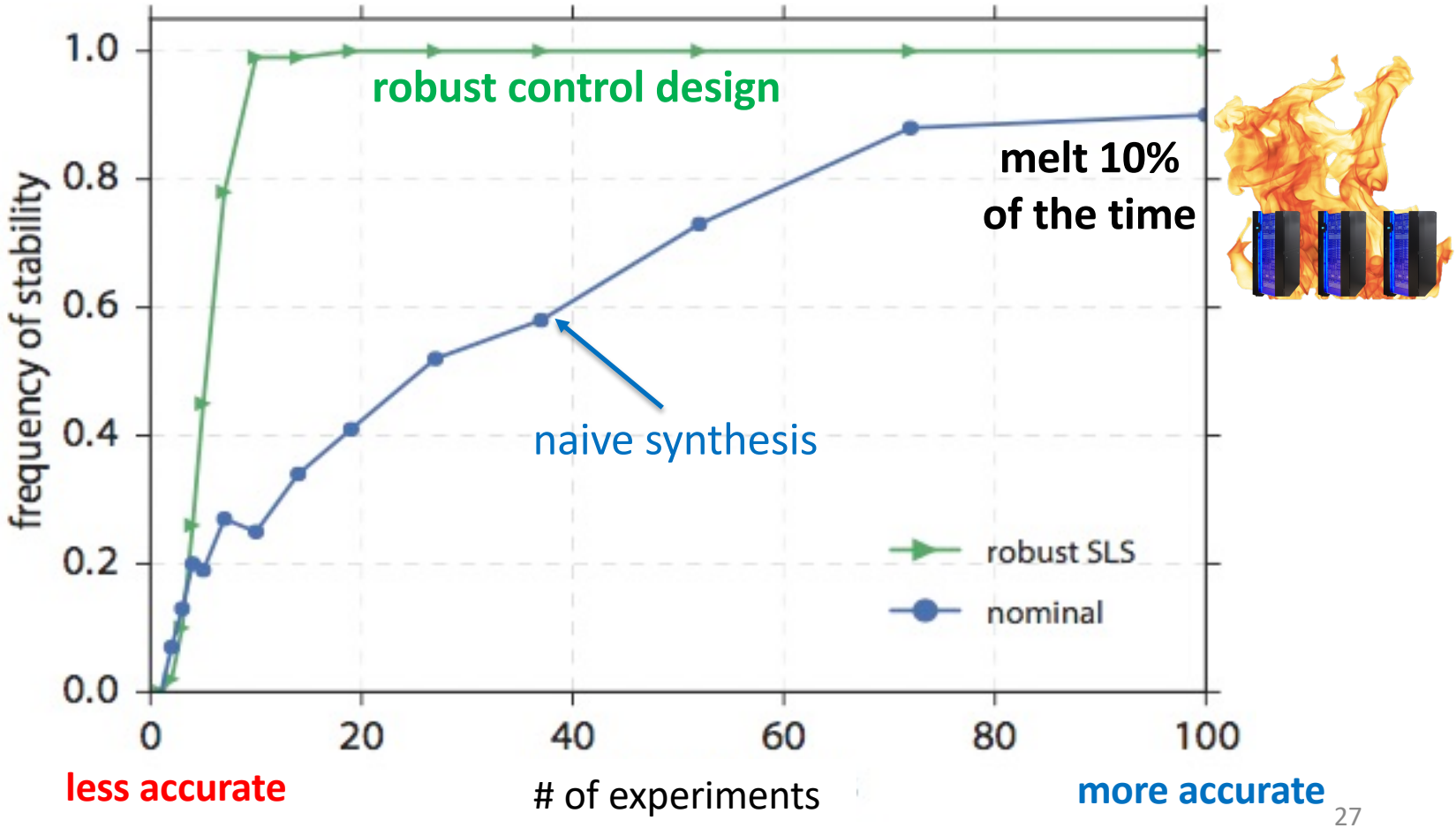


$$x_{t+1} = \begin{bmatrix} ? \end{bmatrix} x_t + \begin{bmatrix} ? \end{bmatrix} u_t + \delta_t$$

**Known dynamics:** $u_t = K_t x_t$ **feedback based policy**

**Unknown dynamics?**

# Frequency of servers NOT melting (stability)



# Example: Vision based control

Waymo

**How should we train vision systems for safety-critical control loops?**

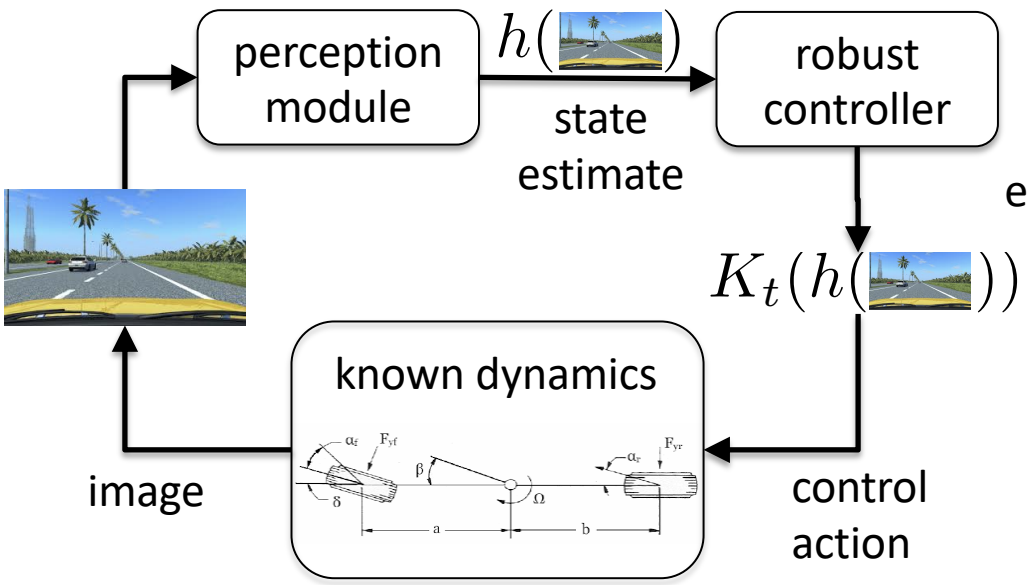
**Should we learn a map from pixels to:**
**control action (end-to-end)?**
**waypoints? position?**

**Challenges**
robustness to environmental shifts
robustness to spurious errors

# Vision based control

$$x_{t+1} = Ax_t + Bu_t + w_t$$

$$u_t = K_t(h(\text{image}))$$




**ML and Uncertainty Quantification:**
learn map $h$:image → position
and learn **estimation error**

**Robust control:**
Control robust to estimation error

**Challenges**
environmental shifts and spurious errors

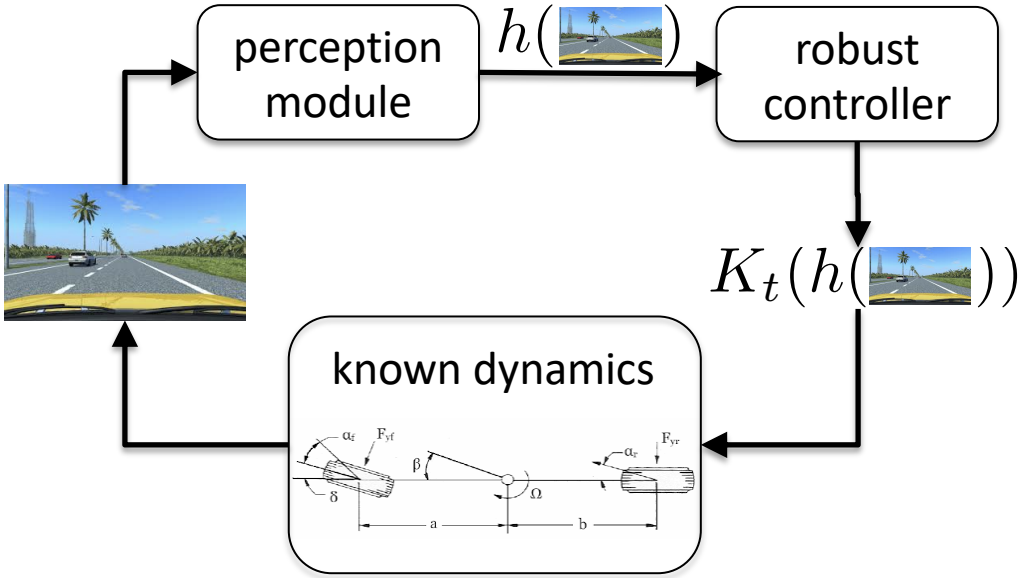
**Approach:**
take an adversarial view of the world

**Can we exploit that we know: dynamics? control objectives?**

# Vision based control

$$x_{t+1} = Ax_t + Bu_t + w_t$$

$$u_t = K_t(h(\text{  }))$$




**Theorem (informal)**
https://arxiv.org/abs/1907.03680

Synthesize a **robust controller** with respect to the **learned error model.**

Then* there is a **robust invariant set** around training data for which **generalization error** is bounded, and controller has provable performance.

*under smoothness assumptions

# Example: Vision based control

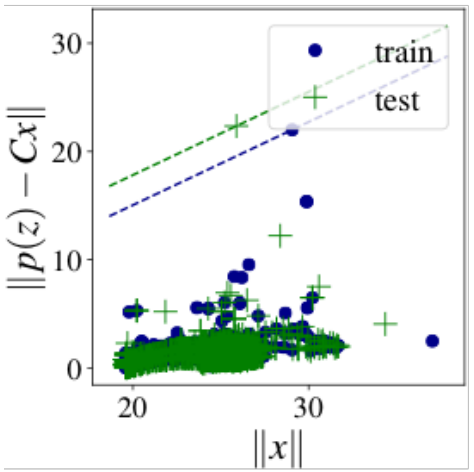
CARLA self-driving car simulator

Desired trajectory

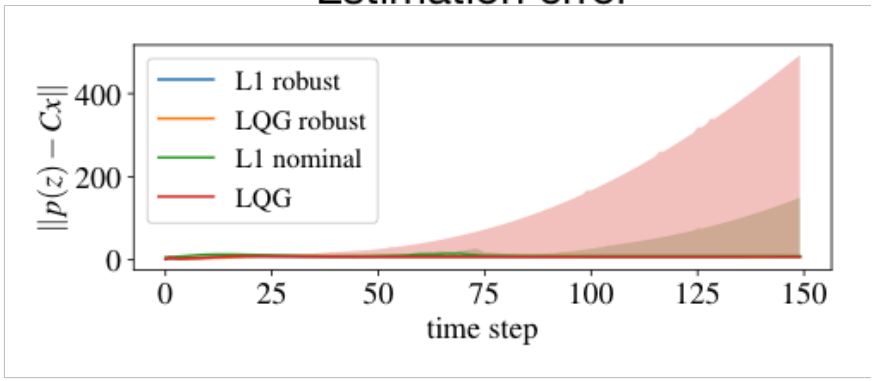
Visual input

# Self-driving car simulator

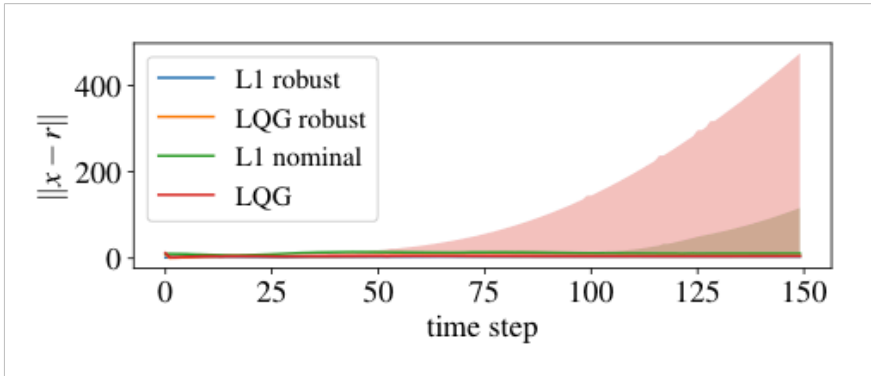
Learned error profile



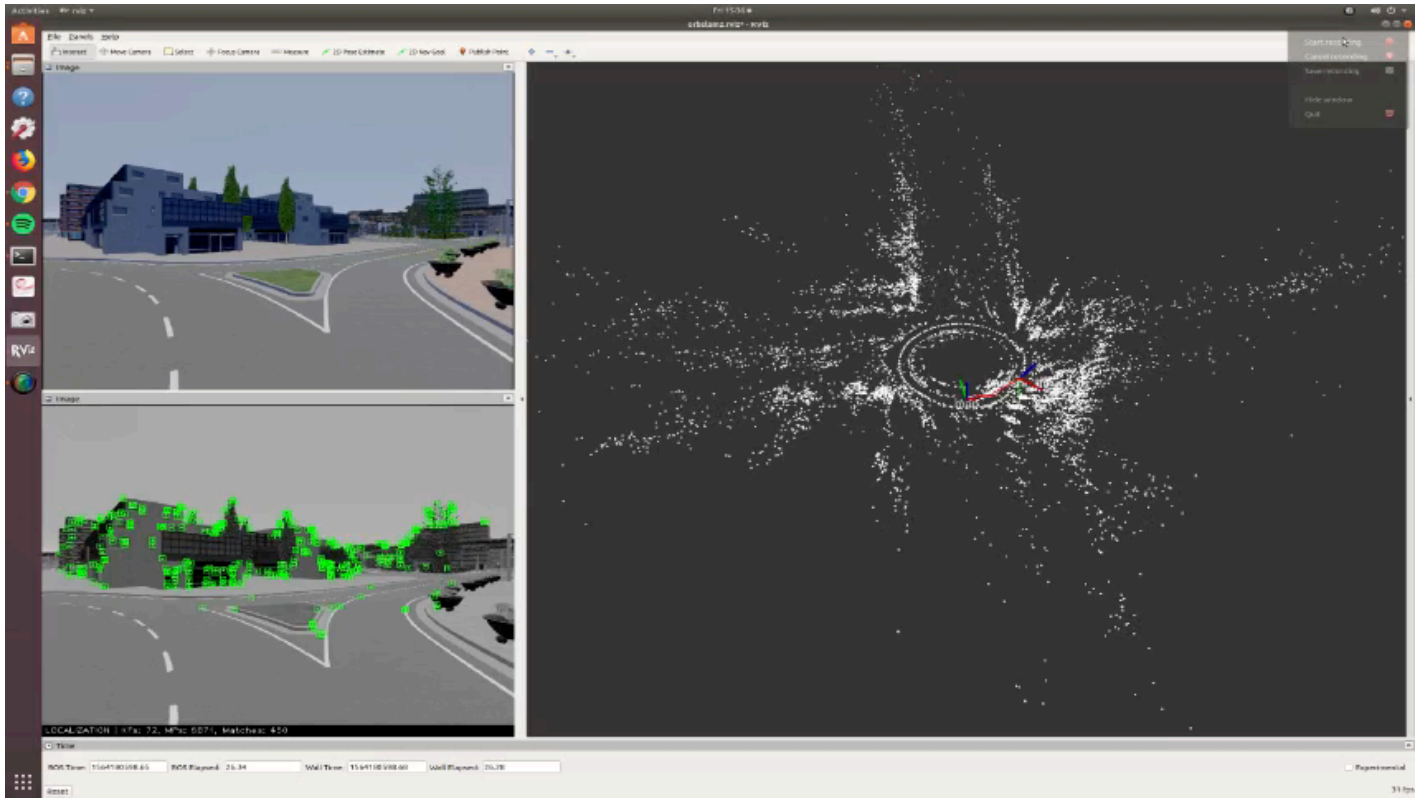

Estimation error



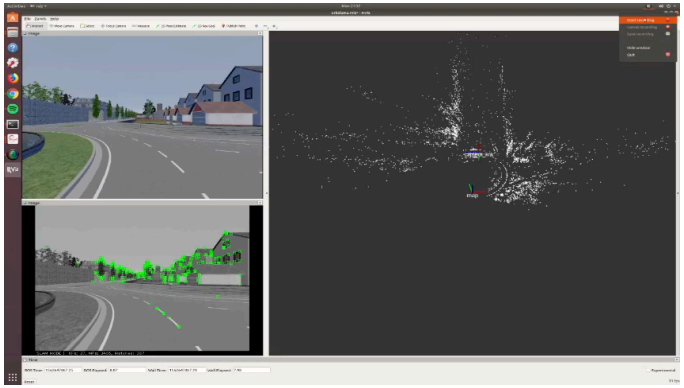

Tracking error

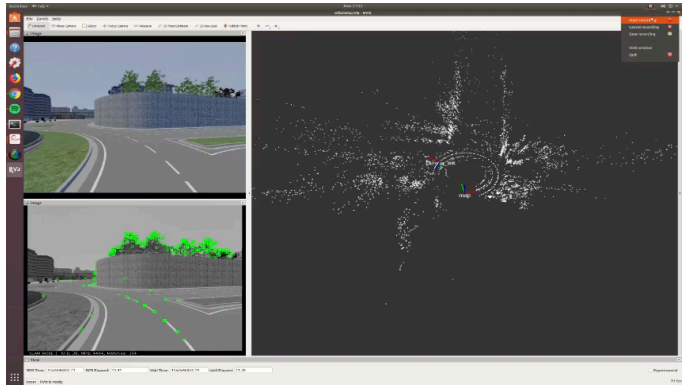


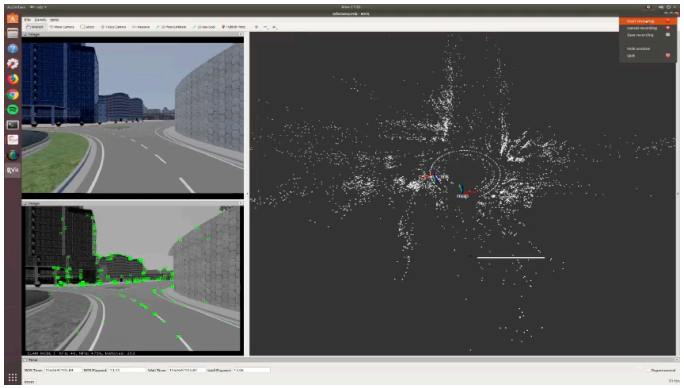
# Self-driving car simulator

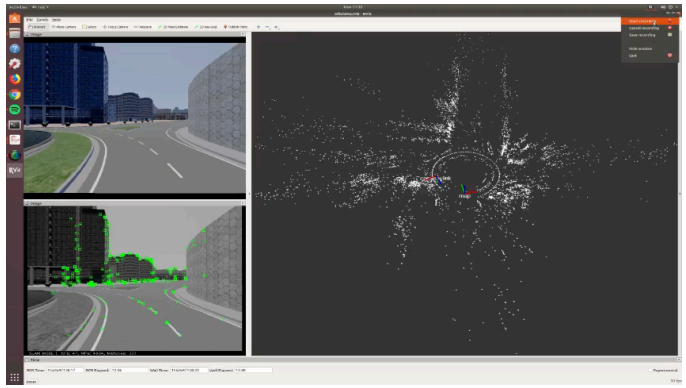


# Self-driving car simulator









# Model Based RL & Control

Uncertainty inherent to the output of a ML algorithm

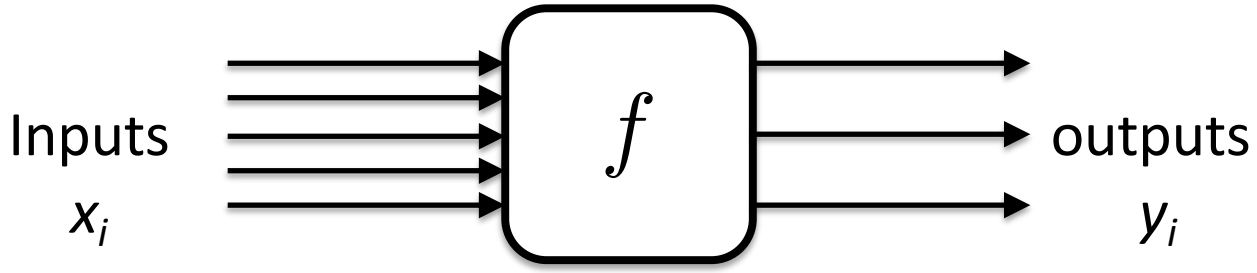
How do we (safely) learn a model and quantify its uncertainty?

*System identification, concentration bounds, bootstrapping*

How do we explicitly account for this uncertainty when we design control policies?

*Robust and optimal control, end-to-end analysis*

# Statistical Learning Theory




Given *n* training examples: $\{(x_1, y_1), \ldots, (x_n, y_n)\} \overset{\text{iid}}{\sim} p^n$

Learn function $\hat{f}$ such that

$$\hat{f}(x) \approx y$$

on future *(x,y)* pairs. We are learning to ***predict*** *y* from *x.*
Want to learn a map that ***generalizes*** to future instances.

# Example: Linear regression

$$y = \Theta^\top x + w,\ w \perp x$$

| **Input *x*** | **Output *y*** |
|---|---|
| High school GPA, SAT score | University GPA |
| Stock price for FB, NFLX, APPL | NASDAQ idx fund price |
| Drug dose | Disease population |
| Current state & input $(x_t, u_t)$ | Next state $x_{t+1}$ |

# Example: Linear regression

$$y = \Theta^\top x + w,\ w \sim \mathcal{N}(0, \sigma^2) \perp x$$

Given *iid n* training examples: $\{(x_1, y_1), \ldots, (x_n, y_n)\}$

Solve OLS: $\hat{\Theta} = \arg\min_{\Theta} \frac{1}{n} \sum_{i=1}^{n} (y_i - \Theta^\top x_i)^2$

(assume invertibility) $= \left( \sum_{i=1}^{n} x_i x_i^\top \right)^{-1} \sum_{i=1}^{n} x_i y_i$

(plug in $y_i = x_i^\top \Theta + w_i$) $= \Theta + \left( \sum_{i=1}^{n} x_i x_i^\top \right)^{-1} \sum_{i=1}^{n} x_i w_i$

**estimation error**

For ***iid*** training examples:

- Zero mean (unbiased)
- Converges to 0 (asymptotically consistent)
- Converges whp as $O(n^{-1/2})$ (finite-data guarantees)

# Example: Linear System Identification

$$x_{t+1} = Ax_t + Bu_t + w_t,\ w_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_w^2) \perp x_t$$

**Idea:** run 1 identification experiment over *n+1* timesteps and collect *n samples* $\{(x_t, u_t), x_{\{t+1\}}\}_{t=0}^{n}$ and solve:

$$(\hat{A}, \hat{B}) = \arg\min_{(A,B)} \sum_{t=0}^{n} \|x_{t+1} - Ax_t - Bu_t\|_2^2$$

Same problem as last slide, but now data is *not iid*!

This can cause problems: this estimator is not always consistent!

# A general framework: Risk Minimization

Want to solve: $\min_{f:\mathcal{X}\to\mathcal{Y}} \mathbb{E}_p\left[\ell(y, f(x))\right]$ (RM)

- $\ell: \mathcal{Y}\times\mathcal{Y} \to \mathbb{R}$ is the loss function (e.g. logistic, least-squares, etc.)
- *P* is a probability measure on $\mathcal{X}\times\mathcal{Y}$, and is *unknown*
- Only given $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \sim P^n$
  (n.b., this iid assumption will not always hold for us)
- What makes this hard to solve?
  - Only information we have about *P* is from samples *S*
  - ∞-dimensional optimization problem over function space

# Empirical Risk Minimization

Want to solve: $\min_{f:\mathcal{X}\to\mathcal{Y}} \mathbb{E}_p\left[\ell(y, f(x))\right]$ (RM)

Instead solve: $\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(x_i))$ (ERM)

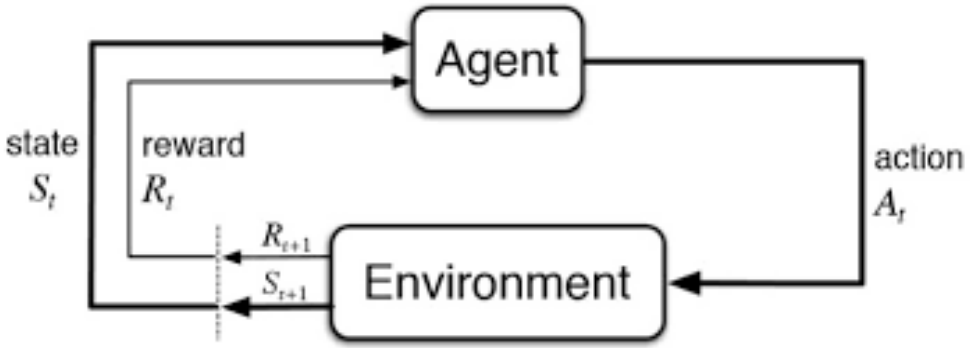
**Much of statistical learning theory:**
understanding when (ERM) is a good proxy for (RM),
i.e., when small empirical risk implies small population risk

# Model free RL & control

$$\text{maximize } \mathbb{E}_{\delta_t}\left[\sum_{t=0}^{N} R(x_t, u_t)\right]$$

$$\text{subject to } x_{t+1} = f(x_t, u_t, \delta_t)$$

$$u_t = \pi_t(u_{0:t-1}, x_{0:t})$$




**Dynamic programming:** if we know *f*, can solve* via DP

$$\mathcal{Q}(x,u) := \max\left\{\mathbb{E}_{\delta_t}\left[\sum_{t=0}^{N} R(x_t,u_t)\right] \,:\, x_{t+1} = f(x_t,u_t,\delta_t),\ (x_0,u_0) = (x,u)\right\}$$

Recursively solve Bellman equation:

$$\mathcal{Q}_N(x,u) = R(x,u),\ \mathcal{Q}_k(x,u) = R(x,u) + \mathbb{E}_{\delta}\left[\max_{u'} \mathcal{Q}_{k+1}(f(x,u,\delta),u')\right]$$

Optimal control action: $u_k = \arg\max_u \mathcal{Q}_k(x_k,u)$

# Reinforcement vs Machine Learning

**Machine learning**

***Draw*** *n* training examples: $\{(x_1, y_1), \ldots, (x_n, y_n)\} \overset{\text{iid}}{\sim} p^n$

Learn function *f* such that $f(x) = \hat{y} \approx y$

**Reinforcement learning**

***Generate*** sample trajectories: $\{(u^{(i)}_{0:t-1}, x^{(i)}_{0:t}, r^{(i)}_{0:t}\} \overset{\text{iid}}{\sim} p^n_{\delta, \pi_{train}}$

Learn policy $\pi$ that produces $u$ so that $r$ is large

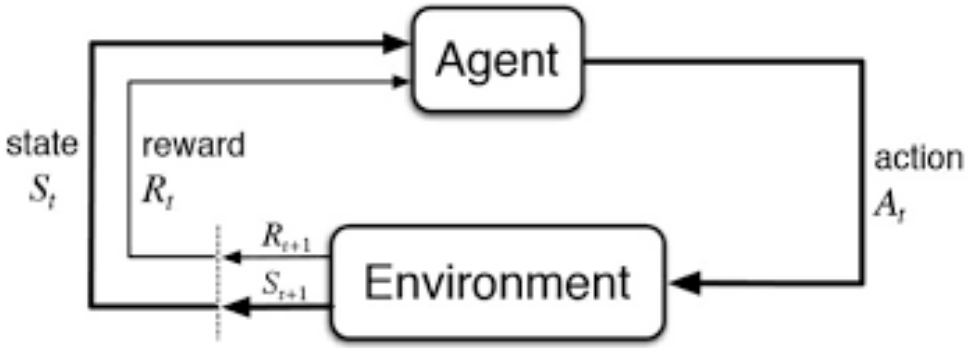
Policy can explore to learn about system, or use existing knowledge to control: this leads to *exploration vs. exploitation tradeoff*

# Model free RL & control

$$\text{maximize } \mathbb{E}_{\delta_t}\left[\sum_{t=0}^{N} R(x_t, u_t)\right]$$

$$\text{subject to } x_{t+1} = f(x_t, u_t, \delta_t)$$

$$u_t = \pi_t(u_{0:t-1}, x_{0:t})$$




**Approximate Dynamic Programming:** if we don't know *f*, if state/action space is too big, etc.
Draw a sample trajectory $\{x_k, u_k\}$ by playing a policy – if it is optimal, then we have

$$\mathcal{Q}(x, u) \approx R(x, u) + \max_{u'} \mathcal{Q}(x_{k+1}, u')$$

(approximately in expectation)

Suggests update rule:

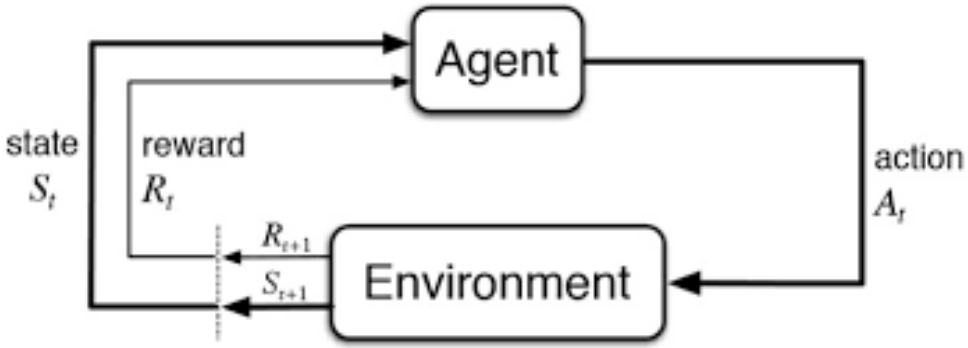
$$\mathcal{Q}_{new}(x_k, u_k) = (1-\eta)Q_{old}(x_k, u_k) + \eta\left(R(x_k, u_k) + \max_{u'} Q_{old}(x_{k+1}, u')\right)$$

# Model free RL & control

$$\text{maximize } \mathbb{E}_{\delta_t}\left[\sum_{t=0}^{N} R(x_t, u_t)\right]$$

$$\text{subject to } x_{t+1} = f(x_t, u_t, \delta_t)$$

$$u_t = \pi_t(u_{0:t-1}, x_{0:t})$$




**Direct Policy Search:** if we don't know *f*, if state/action space is too big, etc.,
We can instead optimize directly over policy $\pi$

Parametrize probabilistic policy as $\pi(x_t) = p(u|u_{0:t-1}, x_{0:t}, \theta)$ for $\theta$ our design parameter
Sample trajectories $(u_{0:t-1}, x_{0:t})$, and then run:

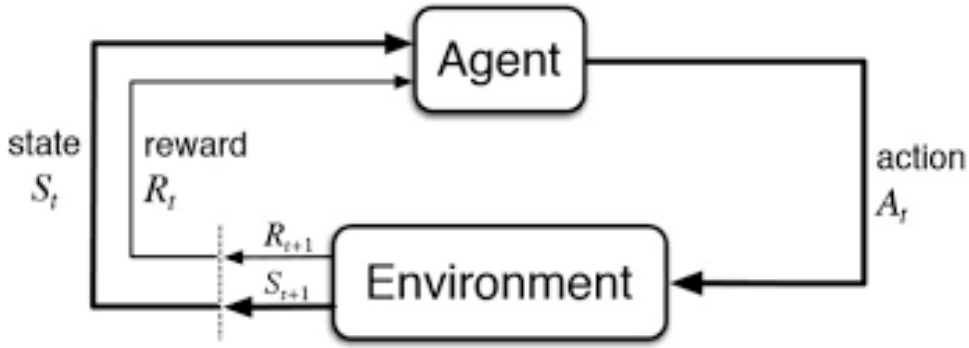
- policy gradient updates using stochastic gradient descent (SGD)
- random search via derivative-free-optimization (DFO)

# What about Deep Learning?

$$\text{maximize } \mathbb{E}_{\delta_t}\left[\sum_{t=0}^{N} R(x_t, u_t)\right]$$

$$\text{subject to } x_{t+1} = f(x_t, u_t, \delta_t)$$

$$u_t = \pi_t(u_{0:t-1}, x_{0:t})$$



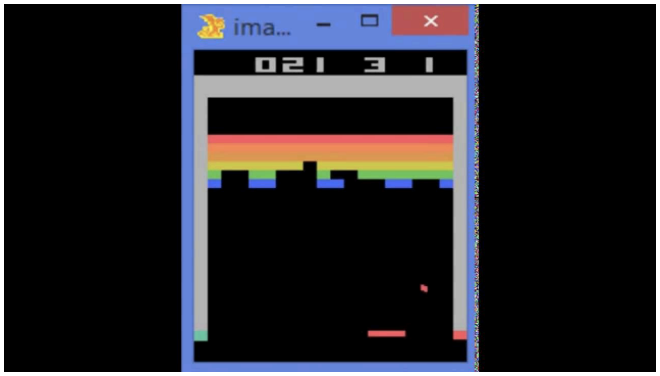

**Model based:** parameterize learned model by a neural net

**ADP:** parameterize Q-function by a neural net

**Direct policy search:** parameterize policy distribution by a neural net

# We don’t need models!


DeepMind AlphaGo




DeepMind DQN

DeepMind AlphaZero

# We don’t need models!


Goldberg lab, UC Berkeley

Google DeepMind

Open AI

**Controlled settings with no cost to failure**

# Model Free Control

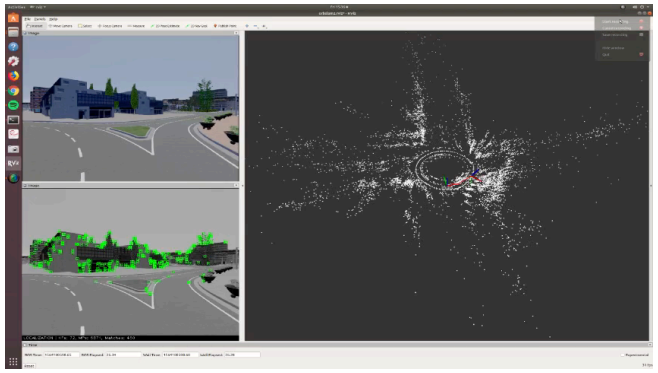
What is the right function space to work with?

Can we enforce stability in a model-free setting?
Safety? Robustness?

How much data do we need to learn good control policies?

When should we use model-free vs. model-based methods?

# Simulation environments



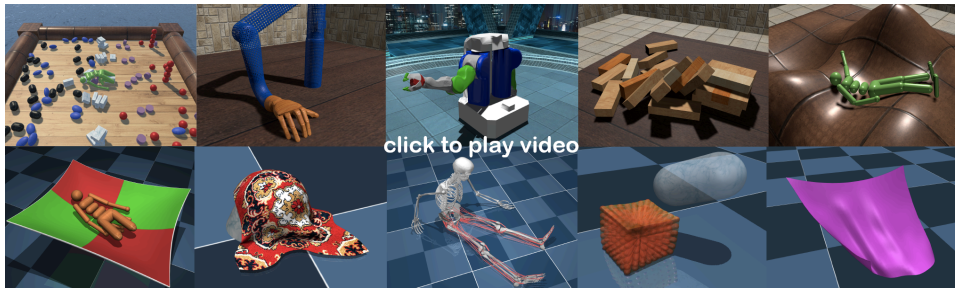
CARLA

FLIGHTGOGGLES

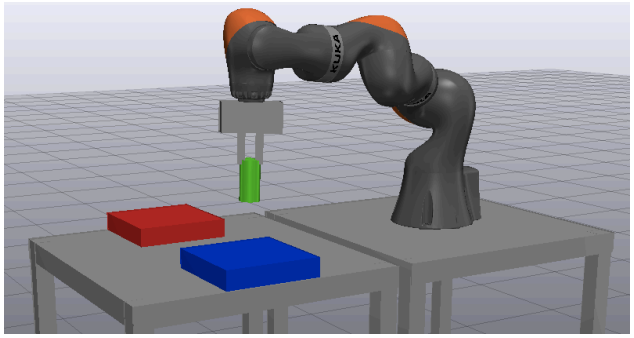
DUCKIETOWN


UDACITY/UNITY-3D

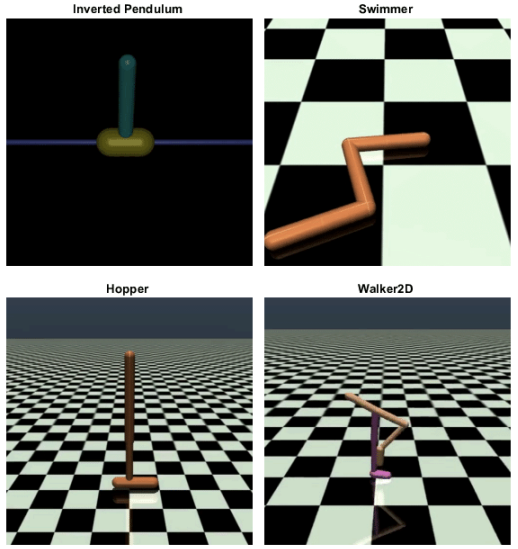
# Simulation environments




MuJoCo



Drake




OpenAI Gym

# Recap

- **Learning** to deal with complexity, **control** to deal with uncertainty

- **Learning + control** promises high-performing and adaptive systems, **but** stability/safety/robustness must be established!

- **Model based approaches**
  Learn model and quantify uncertainty, then apply robust control: can we get end-to-end performance bounds?

- **Model free approaches**
  RL/ADP/PG: broadly applicable, easy to use, impressive demos, but what about safety? robustness? sample efficiency?

# Please do the following!

- Please fill out background poll at https://www.surveymonkey.com/r/XFSR26N
- **If you are sitting in, e-mail me with subject line starting with ESE680, and please still fill this out.**
- Sign up sheet: https://docs.google.com/spreadsheets/d/1Kqa-ZnnqO-r7-c0rTKNSaCNiz_hTQARH9V_Ug8blp3Q/edit?usp=sharing
- Need someone to scribe next class, and please sign up for presentations on Sep 10 and 12 and e-mail me to set up a meeting time next week!