A Few Challenge Problems from Robotics

Russ Tedrake

Follow along at https://slides.com/russtedrake/rl-2020-bc/live
or view later at https://slides.com/russtedrake/rl-2020-bc/
My goals for today

- How well is control working in robotics today?

- A few core questions/challenges
  - What role can simulation play?
  - Why/when does gradient-based policy search work?
  - Parameterizations/algorithms from control.
  - RL in the rare-event regime.

- Challenge problem instances
  - Raibert's hopper, planar gripper, plates, onions, shoe laces.
How well is control working in robotics?

https://www.youtube.com/embed/fRj34o4hN4I?enablejsapi=1&mute=1
Feels like an opportunity for RL?

Is the task difficult because we don't have a model?
What role can simulation play?

for spreading peanut butter, buttoning my shirt, etc.
As more people started applying RL to robots, we saw a distinct shift from "model-free" RL to "model-based" RL. 

*(most striking at the 2018 Conference on Robot Learning)*
As more people started applying RL to robots, we saw a distinct shift from "model-free" RL to "model-based" RL. *(most striking at the 2018 Conference on Robot Learning)*

Recent IFRR global panel on "data-driven vs physics-based models"
As more people started applying RL to robots, we saw a distinct shift from "model-free" RL to "model-based" RL.  
(most striking at the 2018 Conference on Robot Learning)

Recent IFRR global panel on "data-driven vs physics-based models"

(caveat: I didn't choose the panel name)
Core technology: Deep learning perception module that learns "dense correspondences"
Learn a **deep** dynamic model of "keypoint" dynamics.
Online: use model-predictive control (MPC)
What is a (dynamic) model?

input \[ ... \, u_{-1}, u_0, u_1, ... \]

System

output \[ ... \, y_{-1}, y_0, y_1, ... \]
What is a (dynamic) model?

\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]

\[ y_n = g(n, x_n, u_n, w_n, \theta) \]
What is a (dynamic) model?

System

\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]
\[ y_n = g(n, x_n, u_n, w_n, \theta) \]

State-space
What is a (dynamic) model?

System

\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]
\[ y_n = g(n, x_n, u_n, w_n, \theta) \]
What is a (dynamic) model?

\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]
\[ y_n = g(n, x_n, u_n, w_n, \theta) \]
What is a (dynamic) model?

\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]
\[ y_n = g(n, x_n, u_n, w_n, \theta) \]

- **input**: \( u_{-1}, u_0, u_1, \ldots \)
- **state**: \( x_n \)
- **noise/disturbances**: \( w_n \)
- **parameters**: \( \theta \)
- **output**: \( y_{-1}, y_0, y_1, \ldots \)

**State-space**
What is a (dynamic) model?

**State-space**

\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]
\[ y_n = g(n, x_n, u_n, w_n, \theta) \]

**Auto-regressive (eg. ARMAX)**

\[ y_{n+1} = f(n, u_n, u_{n-1}, \ldots, w_n, w_{n-1}, \ldots, \theta) \]
What is a (dynamic) model?

\[ \ldots, u_{-1}, u_0, u_1, \ldots \rightarrow \text{System} \rightarrow \ldots, y_{-1}, y_0, y_1, \ldots \]
What is a (dynamic) model?

Lagrangian mechanics,
Recurrent neural networks (e.g. LSTM), ...

State-space

$x_{n+1} = f(n, x_n, u_n, w_n, \theta)$
$y_n = g(n, x_n, u_n, w_n, \theta)$
What is a (dynamic) model?

State-space

\[
x_{n+1} = f(n, x_n, u_n, w_n, \theta)
\]

\[
y_n = g(n, x_n, u_n, w_n, \theta)
\]

Lagrangian mechanics,

Recurrent neural networks (e.g. LSTM), ...

Feed-forward networks (e.g. \(y_n = \text{image}\))
What is a (dynamic) model?

**State-space**

\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]
\[ y_n = g(n, x_n, u_n, w_n, \theta) \]

**Auto-regressive (eg. ARMAX)**

\[ y_{n+1} = f(n, u_n, u_{n-1}, \ldots, y_n, y_{n-1}, \ldots, w_n, w_{n-1}, \ldots, \theta) \]

Lagrangian mechanics,
Recurrent neural networks (e.g. LSTM), ...

Feed-forward networks
(e.g. \( y_n = \text{image} \))
What is a (dynamic) model?

Lagrangian mechanics,
Recurrent neural networks (e.g. LSTM), ...

State-space
\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]
\[ y_n = g(n, x_n, u_n, w_n, \theta) \]

Auto-regressive (eg. ARMAX)
\[ y_{n+1} = f(n, u_n, u_{n-1}, \ldots, y_n, y_{n-1}, \ldots, w_n, w_{n-1}, \ldots, \theta) \]

Feed-forward networks
(e.g. \( y_n = \text{image} \))

Input
\[ \ldots, u_{-1}, u_0, u_1, \ldots \]

System

Output
\[ \ldots, y_{-1}, y_0, y_1, \ldots \]
Models come in many forms

$Q^\pi(n, x_n, u_n, \theta)$

$\ldots, u_{-1}, u_0, u_1, \ldots \rightarrow Q^\pi(n, x_n, u_n, \theta) \rightarrow \ldots, y_{-1}, y_0, y_1, \ldots$

$Q$-functions are models, too. They try to predict only one output (the cost-to-go).

As you know, people are using $Q$-functions in practice on non-Markovian state representations.
Model "class" vs model "instance"

\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]
\[ y_n = g(n, x_n, u_n, w_n, \theta) \]

- \( f \) and \( g \) describe the model class.
Model "class" vs model "instance"

\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]
\[ y_n = g(n, x_n, u_n, w_n, \theta) \]

- \( f \) and \( g \) describe the model class.
- with \( \theta \) describes a model instance.
Model "class" vs model "instance"

- $f$ and $g$ describe the model class.
- with $\theta$ describes a model instance.

"Deep models vs Physics-based models?" is about model **class**:
Should we prefer writing $f$ and $g$ using physics or deep networks?
Model "class" vs model "instance"

\[ x_{n+1} = f(n, x_n, u_n, w_n, \theta) \]
\[ y_n = g(n, x_n, u_n, w_n, \theta) \]

- \( f \) and \( g \) describe the model class.
- with \( \theta \) describes a model instance.

"Deep models vs Physics-based models?" is about model \textbf{class}:

Should we prefer writing \( f \) and \( g \) using physics or deep networks?

Maybe not so different from

- should we use ReLU or \texttt{tanh}?
- should we use LSTMs or Transformers?
Galileo, Kepler, Newton, Coulomb, Hooke... were *data scientists*. 

Galileo's notes on projectile motion
Galileo, Kepler, Newton, Coulomb, Hooke... were *data scientists.*

They fit very simple models to very noisy data.

Galileo's notes on projectile motion
Galileo, Kepler, Newton, Coulomb, Hooke... were \textit{data scientists}.

They fit very simple models to very noisy data.

Gave us a rich \textit{class} of parametric models that \textit{we} could fit to new data.
Galileo, Kepler, Newton, Coulomb, Hooke... were *data scientists.*

They fit very simple models to very noisy data.

Gave us a rich *class* of parametric models that *we* could fit to new data.

What if Newton had deep learning...?
"All models are wrong, but some are useful" -- George Box

What makes a model useful?
Use case: Simulation

e.g., for

- generating synthetic training data
- Monte-Carlo policy evaluation
- offline policy optimization

What makes a model useful?
Use case: Simulation

e.g., for

- generating synthetic training data
- Monte-Carlo policy evaluation
- **offline policy optimization**

What makes a model useful?

- Reasonable: $y_{sim}(u) \in \{y_{real}(u)\}$
Use case: Simulation

e.g., for

- generating synthetic training data
- Monte-Carlo policy evaluation
- offline policy optimization

What makes a model useful?

- Reasonable: $y_{sim}(u) \in \{y_{real}(u)\}$
- Coverage: $\forall y_{real}(u), \exists y_{sim}(u)$

Unreal 5 engine trailer
Use case: Simulation

e.g., for

- generating synthetic training data
- Monte-Carlo policy evaluation
- **offline policy optimization**

What makes a model useful?

- **Reasonable:** $y_{sim}(u) \in \{y_{real}(u)\}$
- **Coverage:** $\forall y_{real}(u), \exists y_{sim}(u)$
- **Accuracy:** $p(y_{real}|u) \approx p(y_{sim}|u)$
Use case: Simulation

e.g., for

- generating synthetic training data
- Monte-Carlo policy evaluation
- **offline policy optimization**

What makes a model useful?

- **Reasonable:** $y_{sim}(u) \in \{y_{real}(u)\}$
- **Coverage:** $\forall y_{real}(u), \exists y_{sim}(u)$
- **Accuracy:** $p(y_{real}|u) \approx p(y_{sim}|u)$
- **Corollary:** Reliable system identification (data $\Rightarrow \theta$)

Unreal 5 engine trailer
Use case: Simulation

e.g., for

- generating synthetic training data
- Monte-Carlo policy evaluation
- offline policy optimization

What makes a model useful?

- Reasonable: $y_{sim}(u) \in \{y_{real}(u)\}$
- Coverage: $\forall y_{real}(u), \exists y_{sim}(u)$
- Accuracy: $p(y_{real}|u) \approx p(y_{sim}|u)$
- Corollary: Reliable system identification (data $\Rightarrow \theta$)
- Generalizable, efficient, repeatable, interpretable/deployable, ...

Unreal 5 engine trailer
Use case: Online Decision Making (Planning/Control)

What makes a model useful?

- Reasonable, Accurate, Generalizable, ...
- *Efficient / compact*
- *Observable*
- *Task-relevant*
What makes a model useful?

- Reasonable, Accurate, Generalizable, ...
- **Efficient / compact**
- Observable
- Task-relevant

State-space models tend to be more efficient/compact, but require *state estimation.*

**State-space**

\[
\begin{align*}
x_{n+1} &= f(n, x_n, u_n, w_n, \theta) \\
y_n &= g(n, x_n, u_n, w_n, \theta)
\end{align*}
\]

**Auto-regressive**

\[
\begin{align*}
y_{n+1} &= f(n, u_n, u_{n-1}, \ldots, y_n, y_{n-1}, \ldots, w_n, w_{n-1}, \ldots, \theta)
\end{align*}
\]
What makes a model useful?

- Reasonable, Accurate, Generalizable, ...
- Efficient / compact
- Observable
- Task-relevant

State-space models tend to be more efficient/compact, but require state estimation.

- Ex: chopping onions.
  - Lagrangian state not observable.
  - Task relevant?

\[
x_{n+1} = f(n, x_n, u_n, w_n, \theta) \\
y_n = g(n, x_n, u_n, w_n, \theta)
\]

vs.

\[
y_{n+1} = f(n, u_n, u_{n-1}, \ldots, y_n, y_{n-1}, \ldots, w_n, w_{n-1}, \ldots, \theta)
\]
Use case: Online Decision Making (Planning/Control)

What makes a model useful?

- Reasonable, Accurate, Generalizable, ...
- **Efficient / compact**
- Observable
- Task-relevant

State-space models tend to be more efficient/compact, but require *state estimation*.

- Ex: chopping onions.
  - Lagrangian state not observable.
  - Task relevant?
- Doesn't imply "no mechanics"

State-space

\[
x_{n+1} = f(n, x_n, u_n, w_n, \theta) \\
y_n = g(n, x_n, u_n, w_n, \theta)
\]

**vs.**

Auto-regressive

\[
y_{n+1} = f(n, u_n, u_{n-1}, \ldots, w_n, w_{n-1}, \ldots, \theta) \\
y_n, y_{n-1}, \ldots, w_n, w_{n-1}, \ldots, \theta
\]
Occam's razor and predicting what *won't* happen

Perhaps the biggest philosophical difference between traditional physics models and "universal approximators".
Occam's razor and predicting what *won't* happen

Perhaps the biggest philosophical difference between traditional physics models and "universal approximators".

- $f, g$ are *not* arbitrary. Mechanics gives us *constraints*:
  - Conservation of mass
  - Conservation of energy
  - Maximum dissipation
  - ...
Occam's razor and predicting what \textit{won't} happen

Perhaps the biggest philosophical difference between traditional physics models and "universal approximators".

- $f, g$ are \textit{not} arbitrary. Mechanics gives us \textit{constraints}:
  - Conservation of mass
  - Conservation of energy
  - Maximum dissipation
  - ...

- Arguably these constraints give our models their \textit{structure}:
  - Control affine
  - Inertial matrix is positive definite
  - Inverse dynamics have "branch-induced sparsity"
Occam's razor and predicting what won't happen

Perhaps the biggest philosophical difference between traditional physics models and "universal approximators".

- $f, g$ are not arbitrary. Mechanics gives us constraints:
  - Conservation of mass
  - Conservation of energy
  - Maximum dissipation
  - ... 

- Arguably these constraints give our models their structure
  - Control affine
  - Inertial matrix is positive definite
  - Inverse dynamics have "branch-induced sparsity"

- *Without structure, maybe we can only ever do stochastic gradient descent...?*
The failings of our physics-based models are mostly due to the unreasonable burden of estimating the "Lagrangian state" and parameters.

For e.g. onions, laundry, peanut butter, ...
For e.g. onions, laundry, peanut butter, ...

The failings of our physics-based models are mostly due to the unreasonable burden of estimating the "Lagrangian state" and parameters.

The failings of our deep models are mostly due to our inability to do efficient/reliable planning, control design and analysis.
For e.g. onions, laundry, peanut butter, ...

The failings of our physics-based models are mostly due to the unreasonable burden of estimating the "Lagrangian state" and parameters.

The failings of our deep models are mostly due to our inability to due efficient/reliable planning, control design and analysis.

I want the next Newton to come around and to work on onions, laundry, peanut butter...
Why/when does gradient-based policy search work?
"PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance."

https://openai.com/blog/openai-baselines-ppo/
Global Convergence of Policy Gradient Methods for the Linear Quadratic Regulator

Maryam Fazel¹, Rong Ge², Sham M. Kakade¹, and Mehran Mesbahi¹

Learning the model-free linear quadratic regulator via random search

Hesameddin Mohammadi
Mahdi Soltanolkotabi
Mihailo R. Jovanović

Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089.
But there are also cases where it will not work...

A simple counter-example from static output feedback:

\[ \dot{x} = Ax + Bu, \quad y = Cx, \]

\[
A = \begin{bmatrix}
0 & 0 & 2 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}, \quad B = \begin{bmatrix}
1 \\
0 \\
0
\end{bmatrix}, \quad C = \begin{bmatrix}
1 & 1 & 3
\end{bmatrix},
\]

\[ u = -ky. \]

The set of stabilizing \( k \) is a disconnected set.

http://underactuated.mit.edu/policy_search.html
But there are also cases where it will not work...

A simple counter-example from static output feedback:

\[ \dot{x} = Ax + Bu, \quad y = Cx, \]

A = \[
\begin{bmatrix}
0 & 0 & 2 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{bmatrix}
\], \quad B = \[
\begin{bmatrix}
1 \\
0 \\
0 \\
\end{bmatrix}
\], \quad u = -ky.

The set of stabilizing \( k \) is a disconnected set.

<table>
<thead>
<tr>
<th>( k )</th>
<th>Maximum real closed-loop eigenvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>-0.035</td>
</tr>
<tr>
<td>1.5</td>
<td>0.032</td>
</tr>
<tr>
<td>2.1</td>
<td>-0.009</td>
</tr>
</tbody>
</table>

http://underactuated.mit.edu/policy_search.html
What characterizes the good cases?

• Maybe it is *over-parameterization* of deep policies?
  ▪ Is there a comparable story to interpolating solutions in high-dimensional policy space?

• Maybe it is the *distribution over tasks*?
  ▪ Control has traditionally studied algorithms that must work for all A,B,C. Maybe the world never gives us the hard ones?
  ▪ Optimizing simultaneously over diverse tasks might be easier than optimizing over one task.
Lessons from Control
(for instance: better controller parameterizations)
• Just because you *can* search over $u = -Kx$ directly, does not mean that you should!
  ▪ Slow convergence
  ▪ Set of stabilizing $K$ is nontrivial

• If model is known, searching $Q$ and $R$ is better.

Feedback Controller Parameterizations for Reinforcement Learning

John W. Roberts  
CSAIL, MIT  
Cambridge, MA 02139  
Email: jwr@mit.edu

Ian R. Manchester  
CSAIL, MIT  
Cambridge, MA 02139  
Email: irm@mit.edu

Russ Tedrake  
CSAIL, MIT  
Cambridge, MA 02139  
Email: russt@mit.edu

ADPRL, 2012
For LQG, $H_\infty$, etc., we know convex parameterizations.

Feedback Controller Parameterizations for Reinforcement Learning

John W. Roberts  
CSAIL, MIT  
Cambridge, MA 02139  
Email: jwr@mit.edu

Ian R. Manchester  
CSAIL, MIT  
Cambridge, MA 02139  
Email: irm@mit.edu

Russ Tedrake  
CSAIL, MIT  
Cambridge, MA 02139  
Email: russt@mit.edu

ADPRL, 2012
For LQG, $H_\infty$, etc., we know convex parameterizations.

- Youla parameterization (disturbance-based feedback)

Feedback Controller Parameterizations for Reinforcement Learning

John W. Roberts  
CSAIL, MIT  
Cambridge, MA 02139  
Email: jwr@mit.edu

Ian R. Manchester  
CSAIL, MIT  
Cambridge, MA 02139  
Email: irm@mit.edu

Russ Tedrake  
CSAIL, MIT  
Cambridge, MA 02139  
Email: russt@mit.edu

ADPRL, 2012
For LQG, $H_\infty$, etc., we know convex parameterizations.

- Youla parameterization (disturbance-based feedback)
- LMI formulations

Feedback Controller Parameterizations for Reinforcement Learning

John W. Roberts  
CSAIL, MIT  
Cambridge, MA 02139  
Email: jwr@mit.edu

Ian R. Manchester  
CSAIL, MIT  
Cambridge, MA 02139  
Email: irm@mit.edu

Russ Tedrake  
CSAIL, MIT  
Cambridge, MA 02139  
Email: russt@mit.edu

ADPRL, 2012
For LQG, $H_\infty$, etc., we know convex parameterizations.

- Youla parameterization (disturbance-based feedback)
- LMI formulations
- Result in convex formulations only for linear systems, but the benefits are likely more general.

Feedback Controller Parameterizations for Reinforcement Learning

John W. Roberts  
CSAIL, MIT  
Cambridge, MA 02139  
Email: jwr@mit.edu

Ian R. Manchester  
CSAIL, MIT  
Cambridge, MA 02139  
Email: irm@mit.edu

Russ Tedrake  
CSAIL, MIT  
Cambridge, MA 02139  
Email: russt@mit.edu

ADPRL, 2012
Youla parameters (time-domain, no-noise)

\[
\begin{align*}
\min & \sum_{n=0}^{N-1} x^T[n] Q x[n] + u^T[n] R u[n], \quad Q = Q^T \succeq 0, R = R^T > 0 \\
\text{subject to} & \quad x[n + 1] = A x[n] + B u[n], \\
& \quad x[0] = x_0
\end{align*}
\]

\[
\begin{align*}
u[n] &= K_n x[n], \\
x[1] &= A x_0 + B K_0 x_0, \\
x[2] &= A(A + B K_0) x_0 + B K_1 (A + B K_0) x_0 \\
x[n] &= \left( \prod_{i=0}^{n-1} (A + B K_i) \right) x_0
\end{align*}
\]

\[
\begin{align*}
u[n] &= \bar{K}_n x_0, \\
x[1] &= A x_0 + B \bar{K}_0 x_0, \\
x[2] &= A(A + B \bar{K}_0) x_0 + B \bar{K}_1 x_0 \\
x[n] &= \left( A^n + \sum_{i=0}^{n-1} A^{n-i-1} B \bar{K}_i \right) x_0
\end{align*}
\]

http://underactuated.csail.mit.edu/lqr.html
Control parameterizations from the "robot whisperers"
The MIT Leg Lab Hopping Robots

The MIT Leg Lab Hopping Robots

RL in the rare-event regime (for robotics)
Motivation

Consider the task of loading a dishwasher...

(One project I've been working on at TRI Robotics)
Finding subtle bugs

Start mug load

Added (calibrated) noise for rack perception

Rack Position (m)

Time (sec)

OK to place mug

Stop! Rack appears closed
Estimating failure probability for black-box, very high-dimensional, complex simulators

- adaptive multi-level splitting
- Hamiltonian MC
- parametric warping distributions via normalizing flows

Scalable End-to-End Autonomous Vehicle Testing via Rare-event Simulation

Matthew O’Kelly*1 Aman Sinha*2 Hongseok Namkoong*2
John Duchi2 Russ Tedrake3

1University of Pennsylvania
2Stanford University
3Massachusetts Institute of Technology

mokelly@seas.upenn.edu {amans, hnamk, jduchi}@stanford.edu russt@mit.edu

paper link

Neural Bridge Sampling for Evaluating Safety-Critical Autonomous Systems

Aman Sinha*1 Matthew O’Kelly*2 John Duchi1 Russ Tedrake3

1Stanford University
2University of Pennsylvania
3Massachusetts Institute of Technology

amans@stanford.edu, mokelly@seas.upenn.edu, jduchi@stanford.edu, russt@mit.edu

paper link
Failure probability vs "falsification"

Falsification algorithms are not designed for *coverage*.

\[
\text{find } x < 20 \quad \text{vs} \quad \text{estimate } p(x < 20)
\]
The risk-based framework

Don’t just find one spot

Prioritize these failures

Not these
The risk-based framework

Don’t just find one spot

Prioritize these failures

Not these

Original Distribution

$P_0$

Learned sampler

$P_\theta$

Failure Region
The risk-based framework

\[ P_0 = \mathcal{N}(0, I) \]
\[ p(\max(x_i) > 6) \]

Region of interest

a smooth ladder of samplers
Finds surprisingly rare and diverse failures of the full comma.ai openpilot in the Carla simulator.

The primary authors have now created a startup: trustworthy.ai
Invites super interesting questions for RL / control.

- Convinced me that empirical risk estimation *might* actually scale to high dimensional data.
- Plausible formulation of "safety" in difficult domains.

- Do the parameterized warping distributions enable more efficient importance-sampling RL?
Some specific robotics problem instances
Distributional Robustness

We have parameterized simulations to study distributional robust / distribution shift.

- Parameterized environments (lighting conditions, etc)
- Carefully parameterized procedural mugs!
- ...

![Diagram showing procedural mugs from different angles and with different designs.](image-url)
FINGER PIVOTING  SLIDING  FINGER GAITING
Releasing all of these as a part of my fall manipulation class at MIT (which is open and online).

Summary

- How well is control working in robotics today?

- A few core questions/challenges
  - What role can simulation play?
  - Why/when does gradient-based policy search work?
  - Parameterizations/algorithms from control.
  - RL in the rare-event regime.

- Challenge problem instances
  - Plates, planar gripper, Raibert’s hopper, onions, shoe laces...
You will need to rerun this cell if you restart the kernel, but it should be fast because the machine will already have drake installed.

```python
import importlib
import sys
from urllib.request import urlretrieve

# Install drake.
if 'google.colab' in sys.modules and importlib.util.find_spec('pydrake') is None:
    # version='20200901'
```