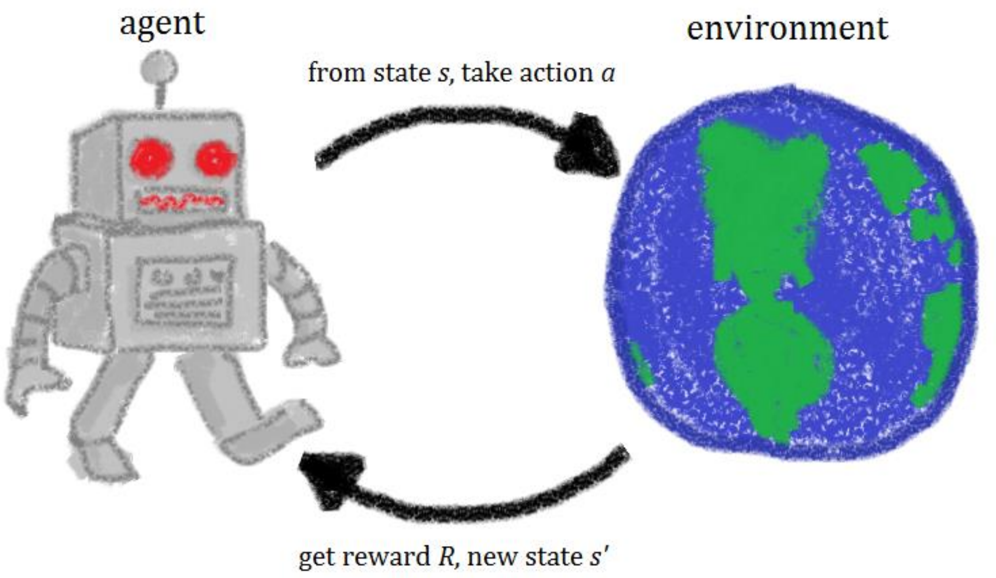
Advanced Robot Control and Learning

Prof. Sami Haddadin

Part 2

Reinforcement Learning in Robotics

Learning through interaction


agent
environment
from state s, take action a
get reward R, new state s'

Reinforcement Learning vs. Optimal Control

Reinforcement learning (RL) is concerned with how agents (dynamical systems) ought to act in an environment in order to maximize certain objective function (reward).

RL often makes intricate, model-free predictions from data alone.

Optimal Control deals with finding a control (actions) for a dynamical system over a period of time such that a certain objective function is optimized.

Optimal Control is the theory of designing complex actions from well-specified models.

Yet both RL and Control aim to design systems that use richly structured perception, perform planning and control that adequately adapt to environmental changes, and exploit safeguards when surprised by a new scenario.

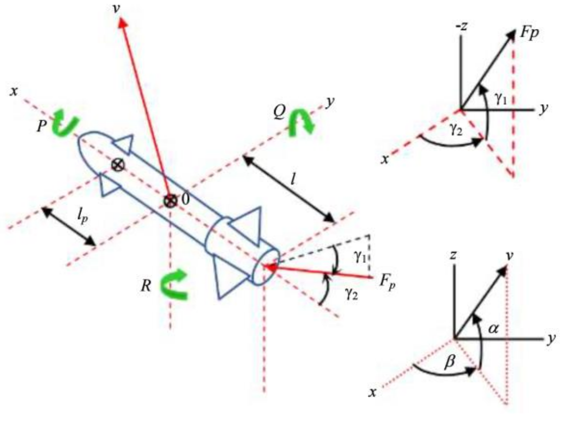
Reinforcement Learning vs. Optimal Control

ANYmal is an autonomous four-legged robot capable of solving various inspection tasks in challenging industrial environments with a wide range of sensors.

ANYmal - Robotic Systems Lab, ETH Zurich

Modelling of a Rocket Optimal Control presents the derivation of the mathematical model for a rocket's autopilot in state-space.

State-Space Modelling of a Rocket for Optimal Control System Design - Kisabo et al



Optimal Control

In optimal control for **discrete time**, the system is described by the state transition function

$$x_{t+1} = f_t(x_t, u_t, e_t)$$

- x_t is the state of the system,
- u_t is the control action, and
- e_t is a random disturbance.

At every time step a cost $c(x_t, u_t)$ is received.
(The negative of cost is what is in RL called reward $c(x_t, u_t) = -R(x_t, u_t)$).
The collection of all state-action pairs is called a trajectory $\tau_t = (u_1, \cdots, u_{t-1}; x_0, \cdots, x_t)$.

The goal is to minimize (maximize) the cost (reward), i.e. minimize $\mathbb{E}_{e_t} \sum_{t=0}^{T} c(x_t, u_t)$ subject to the state transition function.

Optimal Control

Controller is allowed to observe the state before deciding upon the next action. This allows for mitigation of uncertainty through feedback.

The function

$$u_t = \pi_t(\tau_t)$$

determining the next action is called a policy.

In Optimal Control one assumes the knowledge of the state transition function

$$x_{t+1} = f_t(x_t, u_t, e_t)$$

This is precisely where RL comes in. We must learn something about the dynamical system and subsequently choose the best policy based on our knowledge.

Reinforcement Learning

The strategy in RL is to decide on

- a policy π_t, and
- the length of horizon T.

Subsequently, these two are fed to a simulation or a real physical system which in turn returns

- a trajectory τ_T, and
- a sequence of rewards $\{R(x_t, u_t)\}$.

The goal is to find a policy maximizing the received reward with fewest total number of samples (efficiency). This is called episodic RL (see [1]).

[1] Sutton, Barto - Introduction to reinforcement learning. Cambridge: MIT press, 1998.

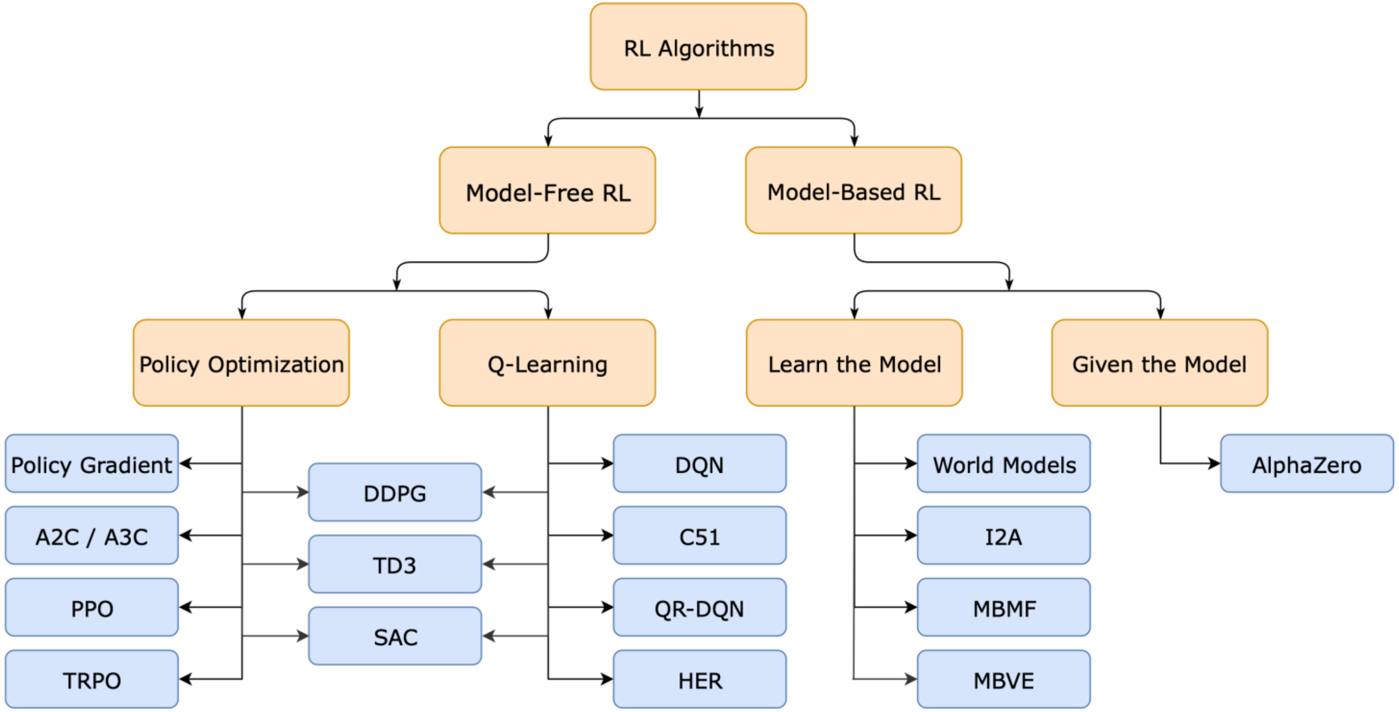
Reinforcement Learning

How to solve this problem? There are two generic strategies in RL, namely

- Model-Based RL – fits a model to previously observed data and then uses this model in some fashion to approximate the solution the problem

- Model-Free RL – deliberately avoids the need for a system’s model, and directly seeks a map from observations to actions.

Reinforcement Learning Taxonomy



Model or No Model?

- Model-Based RL – is hotly debated with strong argument that it is often easier to find a policy for a task than it is to fit a general purpose model of the system dynamics.

- Model-Free RL – are mainly divided into two approaches:
  1. Approximate Dynamic Programming - uses Bellman's principle of optimality to approximate the problem using previously observed data.
  2. Policy Search - directly search for policies by using data from previous episodes in order to improve the reward.

Reinforcement Learning

All of the approaches surveyed reduce to some sort of function fitting from noisy observations of the dynamical system, though performance can be drastically different depending on how you fit this function.

- Model-Based RL – one fits a model of the state transitions to best match observed trajectories.
- Approximate Dynamic Programming (ADP) – one estimates a function that best characterizes the “cost to go” for experimentally observed states.
- Policy Search – one attempts to find a policy, $\pi_t(\tau_t) = u_t$, that directly maximizes the optimal control problem using only input-output data.

The main question is which of these approaches makes the best use of samples and how quickly do the derived policies converge to optimality.

Model-Based Reinforcement Learning (MB RL)

A simple strategy is to choose the actions at random and collect the trajectories. Then one can build a model $\varphi(x_t, u_t)$ such that $x_{t+1} \approx \varphi(x_t, u_t)$.

This model φ might arise from a physical model or might be a approximation by a neural network.
Learning of φ is achieved by minimising

$$\sum_{t=0}^{T-1} \| x_{t+1} - \varphi(x_t, u_t) \|^2$$

This approach works well only if the learned φ and the state transition function f are close.
Similarly to system identification, MB RL aims to learn the model of the system dynamics.
Simultaneously, it learns a policy that is subject to the learned model.

Compare with lecture no. 5

Approximate Dynamic Programming (1st Model-Free)

Defines a function that returns the maximal future expected reward, called Q-function. This function is defined as

$$Q(x,u) = \max\{\mathbb{E}_e \sum_{t=0}^{T} R(x_t, u_t)\}$$

subject to

- the state transition function $x_{t+1} = f(x_t, u_t, e_t)$, and
- the initial condition (x_0, u_0)
- where $\mathbb{E}_e$ is the expectation over the random disturbance e_t

The Q-function determines the value of the optimal control problem that is attained when the first action is set to be u and the initial state is x, i.e. it determines quality of actions.

Approximate Dynamic Programming (1st Model-Free)

If one has the Q-function then the optimal policy is just

$$\pi(x_0) = \arg\max_u Q(x_0, u)$$

We can use dynamic programming to compute the Q-function associated with every action.

That is, we define the terminal Q-function to be

$$Q_T(x,u)=R(x_T,u_T)$$

For every other state the Q-function will be defined recursively as

$$Q_i(x,u)=R(x_i,u_i)+\mathbb{E}_e[\max_{u'} Q_{i+1}(f(x_i,u_i,e_i),u')]$$

The above equation is called the **Bellman’s equation**.

Approximate Dynamic Programming (1st Model-Free)

Approximate Dynamic Programming methods typically try to compute the Q-function from data.

They do so by

- assuming that the Q-function is stationary – the transition function does not change over time (i.e. $Q_i(x,u) = Q(x,u)$, for all i); and
- slightly modifying the Q-function by discounting future rewards to be
$$Q(x,u) = R(x_i,u_i) + \gamma \mathbb{E}_e[\max_{u'} Q(f(x_i,u_i,e_i),u')]$$
where γ is called the discount factor.

This assures that even for infinite horizon tasks the Q-function stays finite.

The equation is the **discounted Bellman's equation** for Q-function.

Approximate Dynamic Programming (1st Model-Free)

The Bellman's equation for Q-function can't be solved directly for the Q-function (because it's implicit). However, one can find the Q-function by solving the following problem

$$\max(1-\gamma)\mathbb{E}_e \sum_{t=0} \gamma^t R(x_t, u_t)$$

subjected to

- $x_{t+1} = f(x_t, u_t, e_t)$,
- $u_{t+1} = \pi_t(\tau_t) = \arg\max_u Q(x_t, u)$.

From here one can use the stochastic approximation method (next slide).

Approximate Dynamic Programming (1st Model-Free)

The stochastic approximation method uses the received reward and the current Q-function in order to improve the Q-function

$$Q^{(\text{new})}(x_t, u_t) = (1-\gamma)Q^{(\text{old})}(x_t, u_t) + \eta\left(R(x_t, u_t) + \gamma \max_{u'} Q^{(\text{old})}(x_{t+1}, u')\right)$$

where η is the learning rate.

Though we have switched away from models, there's no free lunch. We are still estimating functions, and we need to assume that they have some reasonable structure else we can't learn them.

Moreover, for continuous control problems, ADP methods make an inefficient use of samples. Suppose the internal state of the system is of dimension d. Modelling the state-transition function (Model-Based RL) provides d equations per time step. By contrast, ADP only uses 1 equation per time step.

Policy Search (2nd Model-Free)

- Policy search offers many features relevant to robotics (natural integration of expert knowledge).
- Optimal policies often have fewer parameters than optimal Q-functions.
- Most policy search methods optimize locally around existing policies π, parametrized by a set of policy parameters θ_i. By computing changes in the policy parameters $\Delta\theta_i$ that will increase the expected return and results in iterative updates of the form

$$\theta_{i+1} = \theta_i + \Delta\theta_i$$

Gradient based methods estimate $\Delta\theta_i$ using gradient of the expected return

$$\mathrm{J} = \lim_{T\to\infty} \mathbb{E}_e \frac{1}{T}\sum_{t=0}^{T} R(x_t, u_t)$$

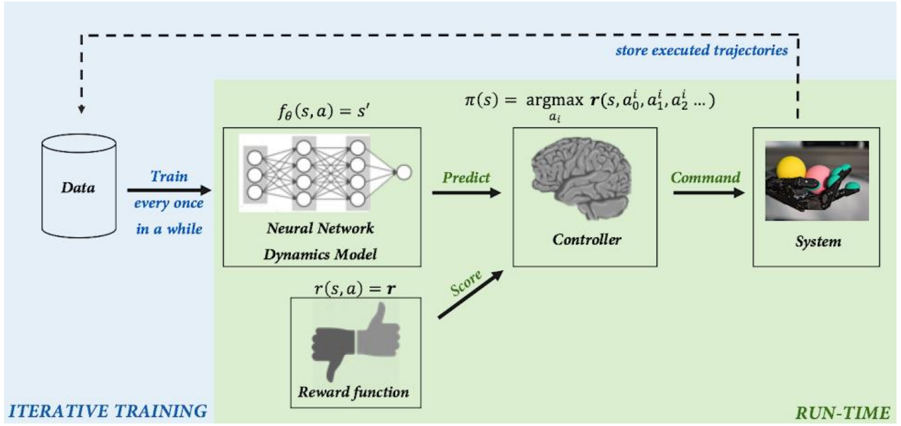
i.e. $\Delta\theta_i = \alpha\nabla J$ with α being the learning rate.

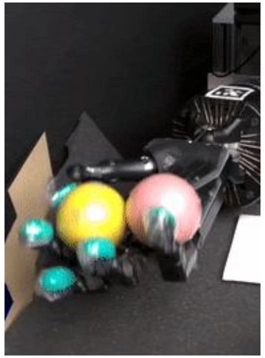
RL in Robotics

Deep Dynamics Models for Dexterous Manipulation

(Anusha Nagabandi, Kurt Konoglie, Sergey Levine, Vikash Kumar)

- Model-based RL – learns $\varphi(x_t, u_t) \approx x_{t+1}$
    1. data is iteratively collected by attempting the task using the latest model;
    2. updating the model using this experience


store executed trajectories
Data
Train every once in a while
$f_\theta(s,a) = s'$
Neural Network Dynamics Model
Predict
$\pi(s) = \underset{a_i}{\text{argmax}}\, r(s, a_0^i, a_1^i, a_2^i \ldots)$
Controller
Command
System
$r(s,a) = r$
Score
Reward function
ITERATIVE TRAINING
RUN-TIME



Additional RL examples

AI Learns to Park – Deep Reinforcement Learning

OpenAI Five – Dota Gameplay

Deep Mind - Emergence of Locomotion Behaviur in Rich Environments

Tractability Through Representation

- Most successes achieved by RL in Robotics have been achieved by leveraging a few key principles – **effective** representations, **approximate** models, and **prior knowledge** or information.
- Reducing the dimensionality of states or actions by smart state-action discretization is a representational simplification that may enhance Q-function based methods.

- Smart State-Action Discretization:

1. Hand Crafted Discretization – choices made by engineers
2. Learned from Data – build adaptively during the learning process
3. Meta-Actions – group of movement primitives that are assembled to perform an action
4. Relational Representations – the states, actions, and transitions are not represented individually, but are grouped and their relationships are considered

Tractability Through Prior Knowledge

- Prior knowledge can dramatically help guide the learning process. It includes:
  1. initial policies,
  2. demonstrations,
  3. initial models,
  4. a predefined task structure, or
  5. constraints on the policy (e.g. torque limits)
- These approaches significantly reduce the search space and, thus, speed up the learning process.

1. Prior Knowledge Through Demonstration – Imitation Learning
2. Prior Knowledge Through Task Structuring – task decomposed hierarchically into basic components or into a sequence of increasingly difficult tasks

Tractability Through Models

- Many robot reinforcement learning problems can be made tractable by learning forward models, i.e., approximations of the transition function based on data.

1. Iterative Learning Control – use of crude, approximate models to determine gradients
2. Locally Linear Quadratic Regulators – covered in the tutorial
3. Policy Search with Learned Models – uses both model-free and model-based RL

RL in Robotics Complications

- True state is not completely observable and noise-free; moreover even vastly different states might look very similar. Thus, robotics RL are often modelled as partially observed.

- RL in robotics demands safe exploration. This is a key issue of the learning process, a problem often neglected in the general RL community (due to simulated environments).

- The dynamics of a robot can change due to many external factors (e.g. temperature, wear). This makes comparing algorithms particularly hard.

RL in Robotics Complications

- Small modelling errors can accumulate to big ones, at least for highly dynamic tasks. Hence, algorithms need to be robust with respect to these errors (under-modelling, model uncertainty).

- Another problem is the generation of appropriate reward functions (reward shaping). Specifying good reward functions in robotics requires a fair amount of domain knowledge and may often be hard in practice.