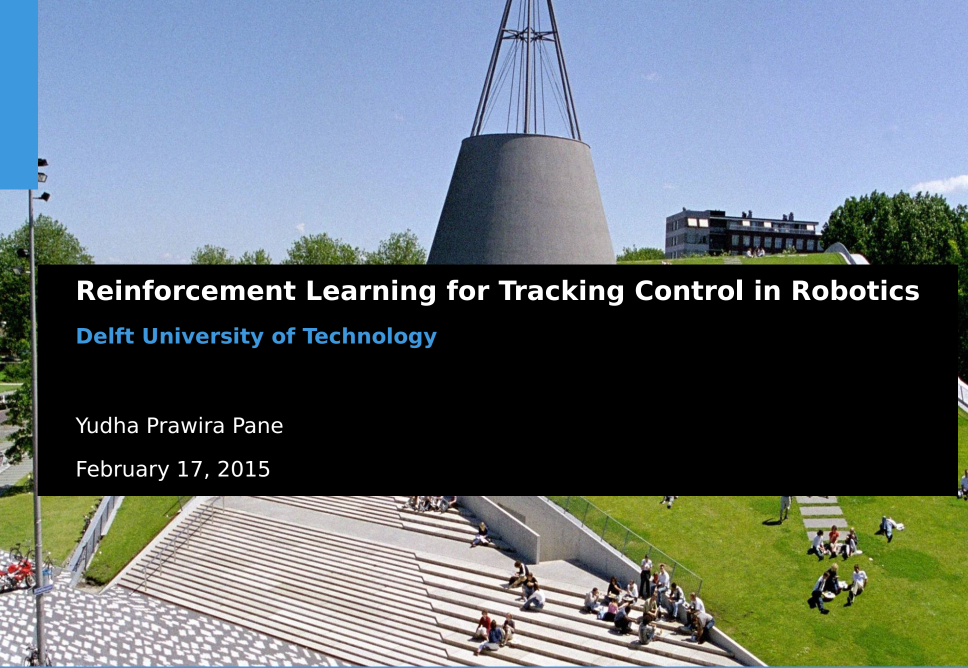


Reinforcement Learning for Tracking Control in Robotics

Delft University of Technology

Yudha Prawira Pane

February 17, 2015

Outline

1. Basics of Reinforcement Learning (RL) & Tracking Control
2. Experimental Setup: 3D Printing Robot System
3. RL for Tracking: A Survey
4. Research Plan & Conclusion

Motivation of RL

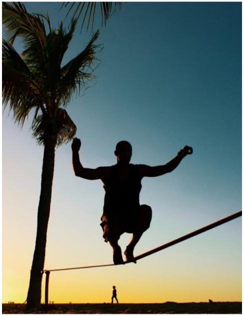
- Inspired by how living organisms learn

Motivation of RL

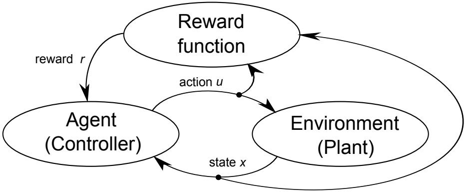
- Inspired by how living organisms learn
- Learning through interaction with environment

Motivation of RL

- Inspired by how living organisms learn
- Learning through interaction with environment



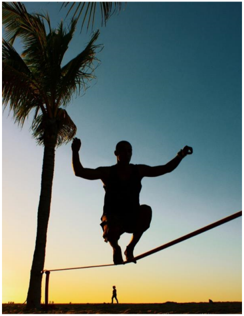
source: google



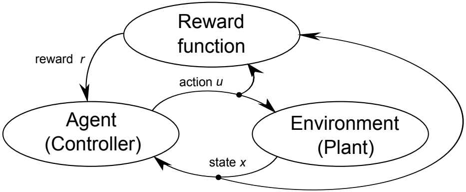
source: SC4081 slide

Motivation of RL

- Inspired by how living organisms learn
- Learning through interaction with environment



source: google



source: SC4081 slide

- Introduce the reward / reinforcement signal

Mathematical Formalization of RL

Definition

A Markov Decision Process (MDP) is defined as a tuple $\langle X, U, \bar{f}, \rho \rangle$ where

- X = state space
- U = action space
- $\bar{f} : X \times U \rightarrow X$ = system dynamics
- $\rho : X \times U \rightarrow \mathbb{R}$ = reward (cost) function

Mathematical Formalization of RL

Definition

A Markov Decision Process (MDP) is defined as a tuple $\langle X, U, \bar{f}, \rho \rangle$ where

- X = state space
- U = action space
- $\bar{f} : X \times U \to X$ = system dynamics
- $\rho : X \times U \to \mathbb{R}$ = reward (cost) function

Definition

System and policy dynamics are defined as:

$$x_{k+1} = \bar{f}(x_k, u_k) = f(x_k) + g(x_k)u_k \tag{1}$$

$$u_{k+1} = \pi(x_k, u_k) \tag{2}$$

$$r_{k+1} = \rho(x_k, u_k) \tag{3}$$

Mathematical Formalization of RL (2)

Definition

Formulize goal as return R

$$R_k = r_k + r_{k+1} + r_{k+1} + \cdots + r_T \tag{4}$$

Discounted return:

$$R_k = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \cdots = \sum_{j=0}^{\infty} \gamma^j r_{k+j+1} \tag{5}$$

Mathematical Formalization of RL (2)

Definition

Formulize goal as return R

$$R_k = r_k + r_{k+1} + r_{k+1} + \cdots + r_T \tag{4}$$

Discounted return:

$$R_k = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \cdots = \sum_{j=0}^{\infty} \gamma^j r_{k+j+1} \tag{5}$$

Definition

Value function V measures how a good is it to be at a certain state x

$$V(x_k) = \rho(x_k, u_k) + \gamma\rho(x_{k+1}, u_{k+1}) + \gamma^2\rho(x_{k+2}, u_{k+2}) + \dots \tag{6}$$

$$V(x_k) = \rho(x_k, u_k) + \gamma V(x_{k+1}) \tag{7}$$

Mathematical Formalization of RL (3)

How to obtain an optimal policy π^*?
First, an exact value function V $\forall x \in \mathcal{X}$ must be found

Definition

Bellman optimality principle:

$$V^*(x_k) = \min_{u_k} \left[\rho(x_k, u_k) + \gamma V^*(x_{k+1}) \right] \tag{8}$$

$$\pi^*(x_{k-1}) = u_k^* = \operatorname*{argmin}_{u_k} \left[\rho(x_k, u_k) + \gamma V^*(x_{k+1}) \right] \tag{9}$$

Solutions to RL problem

Dynamic Programming (DP):

- needs system model
- example: policy iteration (PI)

Solutions to RL problem

Dynamic Programming (DP):

- needs system model
- example: policy iteration (PI)

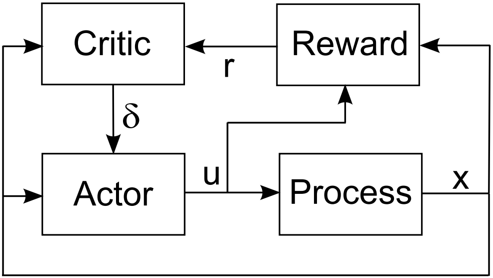
Algorithm 2 Policy Iteration

Initialization:
  Start from an admissible policy π, assign $V^{\pi}(x) \leftarrow 0$
repeat
  Policy Evaluation:
  repeat
    $\Delta \leftarrow 0$
    For each $x \in \mathcal{X}$:
      $v \leftarrow V^{\pi}(x)$
      $V^{\pi}(x) \leftarrow \rho(x, \pi(x)) + \gamma V^{\pi}(x')$
      $\Delta = \max(\Delta, |v - V^{\pi}(x)|)$
  until $\Delta < \varepsilon$ (a small positive number)
  Policy Improvement:
  For each $x \in \mathcal{X}$:
    $\pi(x) = \arg\min_{u} \rho(x, u) + \gamma V^{\pi}(x')$
until π converges

Solutions to RL problem (2)

Temporal Difference (TD)

- does not need system model
- examples: Q-learning, actor-critic



Actor-critic structure (source: SC4081 slide)

Solutions to RL problem (3)

Actor critic method

- suitable for continous state and action space e.g. robotics
- parameterize actor and critic using function approximators

Algorithm 3 Actor-critic algorithm

For every trial:

Initialize x_0 and $u_0 = \tilde{u}_0$

repeat

  apply u_k, measure x_{k+1}, receive r_{k+1}

  choose next action $u_{k+1} = \hat{\pi}(x_{k+1}, \psi_k) + \tilde{u}_{k+1}$

  $\delta_k = r_{k+1} + \hat{V}(x_{k+1}, \theta_k)\hat{V}(x_k, \theta_k)$

  $\theta_{k+1} = \theta_k + \alpha_c \delta_k \frac{\partial \hat{V}(x,\theta)}{\partial \theta}\Big|_{x=x_k, \theta\theta_k}$

  $\psi_{k+1} = \psi_k + \alpha_c \delta_k \frac{\partial \hat{V}(x,\psi)}{\partial \psi}\Big|_{\psi=\psi_k}$

until terminal state

Tracking Control

Typical tracking controllers:

- Open-loop control
- State/Output Feedback control (e.g. PID controller)
- Feedback + feedforward control (e.g. LQT optimal controller)

Drawback: even the best of the 3 only performs as good as the model!

Research Question

The following research question is raised:

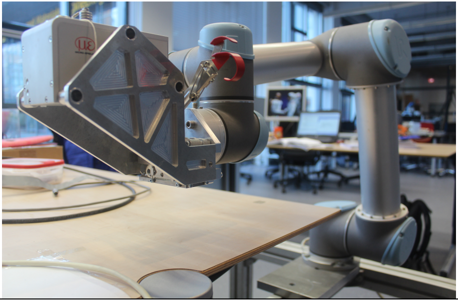
"*Is it possible to improve the tracking performance of a nominal controller using Reinforcement Learning?*"

3D Printing Robot

- The UR5 robot with unknown internal controller
- 4 types of command:
  - Tool Position (x, y, z) in meter
  - Tool Velocity $(\dot{x}, \dot{y}, \dot{z})$ in meter/s
  - Joint Position $(q_1, q_2, q_3, q_4, q_5, q_6)$ in rad
  - Joint velocity $(\dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5, \dot{q}_6)$ in rad/s
- The laser scanner
- The 3D Print head

3D Printing Robot

- The UR5 robot with unknown internal controller
- 4 types of command:
  - Tool Position (x, y, z) in meter
  - Tool Velocity $(\dot{x}, \dot{y}, \dot{z})$ in meter/s
  - Joint Position $(q_1, q_2, q_3, q_4, q_5, q_6)$ in rad
  - Joint velocity $(\dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5, \dot{q}_6)$ in rad/s
- The laser scanner
- The 3D Print head



3D Printing Robot: system identification

- Subspace identification

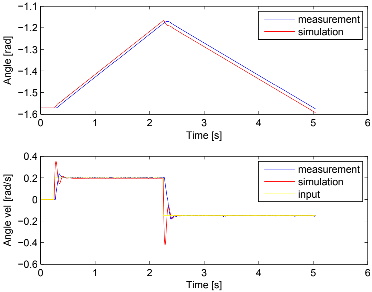
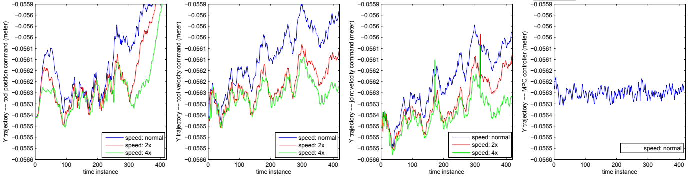


Table 1. VAF scores of the simulated outputs for all joints

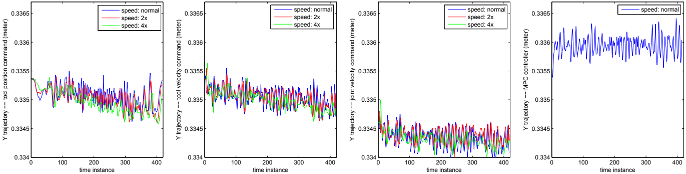
Joint	Position	Velocity
1	98.64	87.33
2	98.05	88.33
3	98.55	88.47
4	98.97	89.50
5	99.46	90.32
6	98.87	85.13

3D Printing Robot: MPC controller

- test a simple straight trajectory along X-axis
- compares MPC with previously mentioned controllers



Y trajectories of the robot with different controllers



Z trajectories of the robot with different controllers

Hypotheses

- *"Current controller (MPC) relies heavily on the identified model which is not perfect. Therefore, the model-mismatch induced by the unknown dynamics is responsible for the non-optimal performance of the MPC controller"*
- *"The best nominal controller yet is not PID, LQR, or else. It is the internal controller of the robot itself"*

RL for Optimal Tracking Control

Assume a SISO LQT problem:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{10}$$

with reference signal r_k.

Definition

The cost function:

$$J = V(x_k, r_k) := \frac{1}{2}\sum_{i=k}^{\infty}\Big(Cx_i - r_i\Big)^T Q\Big(Cx_i - r_i\Big) + u_i{}^T R u_i \tag{11}$$

RL for Optimal Tracking Control (2)

The solution to LQT is a combination of feedback and feedforward term:

$$u_k = -Kx_k + K_v v_{k+1} \tag{12}$$

where

$$v_k = (A - BK)^T v_{k+1} + C^T Q r_k \tag{13}$$

The control gains are:

$$\begin{aligned} K &= (B^T S B + R)^{-1} B^T S A \\ K_v &= (B^T S B + R)^{-1} B^T \end{aligned} \tag{14}$$

with S is the solution of ARE:

$$S = A^T S A - A^T S B (B^T S B + R)^{-1} B^T S A + C^T Q C \tag{15}$$

Drawback: Have to solve a non-causal difference equation

RL for Optimal Tracking Control (3)

Important assumption:

$$r_{k+1} = F r_k \tag{16}$$

Construct an augmented state:

$$\begin{aligned} \begin{bmatrix} x_{k+1} \\ r_{k+1} \end{bmatrix} &= \begin{bmatrix} A & 0 \\ 0 & F \end{bmatrix} \begin{bmatrix} x_k \\ r_k \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k \\ X_{k+1} &= T X_k + B_1 u_k \end{aligned} \tag{17}$$

Definition

Define a lyapunov function:

$$V(x_k, r_k) = V(X_k) = \frac{1}{2} {X_k}^T P X_k \tag{18}$$

Combining the infinite cost with lyapunov function yields a Bellman equation for augmented LQT:

$${X_k}^T P X_k = {X_k}^T Q_1 X_k + {u_k}^T R u_k + {X_{k+1}}^T P X_{k+1} \tag{19}$$

RL for Optimal Tracking Control (4)

Taking the time derivative of LQT Bellman, we obtain the LQT ARE

$$Q_1 - P + T^T PT - T^T PB_1 (R + B_1{}^T PB_1)^{-1} B_1{}^T PT = 0 \tag{20}$$

The optimal policy is given by:

$$u_k = -K_1 X_k \tag{21}$$

with

$$K_1 = (R + B_1{}^T PB_1)^{-1} B_1{}^T PT \tag{22}$$

and

$$Q_1 = \begin{bmatrix} C^T QC & -C^T Q \\ -QC & Q \end{bmatrix} \tag{23}$$

TU Delft

RL for Optimal Tracking Control (5)

From the LQT, obtain lyapunov equation

$$P = Q_1 + {K_1}^T R K_1 + (T - B_1 K_1)^T P (T - B_1 K_1) \tag{24}$$

Solve for P which satisfies (24)

Algorithm 4 Offline Policy Iteration

1: **Initialization:** Select an admissible (stable) gain ${K_1}^0$
2: **repeat**
3: **Policy evaluation:**
4: $P^{j+1} = Q_1 + ({K_1}^j)^T R K_1^j + (T - B_1 {K_1}^j)^T P^{j+1} (T - B_1 {K_1}^j)$
5:
6: **Policy improvement:**
7: ${K_1}^{j+1} = (R + {B_1}^T P^{j+1} B_1)^{-1} {B_1}^T P^{j+1} T$
8: **until** P converges

RL for OTC with unknown system dynamics

- Use a temporal difference (TD) RL technique

Definition

Q-function:

$$Q(X(k), u(k)) = \frac{1}{2} X(k)^T P X(k) \tag{25}$$

Combining (25) with Bellman yields:

$$\begin{aligned} Q(X(k), u(k)) &= \frac{1}{2} X(k)^T Q_1 X(k) + \frac{1}{2} u(k)^T R u(k) + \frac{1}{2} \gamma X^T(k+1) P X(k+1) \\ &= \frac{1}{2} X(k)^T Q_1 X(k) + \frac{1}{2} u(k)^T R u(k) + \frac{1}{2} \gamma (TX(k) + B_1 u(k))^T P (TX(k) + B_1 u(k) \\ &= \frac{1}{2} \begin{bmatrix} X(k) \\ u(k) \end{bmatrix}^T \begin{bmatrix} Q_1 + \gamma T^T P T & \gamma T^T P B_1 \\ \gamma B_1^T P T & R + \gamma B_1^T P B_1 \end{bmatrix} \begin{bmatrix} X(k) \\ u(k) \end{bmatrix} \end{aligned} \tag{26}$$

TU Delft

RL for OTC with unknown system dynamics (2)

By defining:

$$H = \begin{bmatrix} Q_1 + \gamma T^T PT & \gamma T^T PB_1 \\ \gamma B_1^T PT & R + \gamma B_1^T PB_1 \end{bmatrix} = \begin{bmatrix} H_{XX} & H_{Xu} \\ H_{uX} & H_{uu} \end{bmatrix} \tag{27}$$

The optimal input is reached when $\frac{\partial Q(X(k),u(k))}{\partial u(k)} = 0$ which yields:

$$u(k) = -H_{uu}^{-1} H_{uX} X(k) \tag{28}$$

Fortunately, one can apply PI to solve for H:

Algorithm 5 Model-free Policy Iteration

1: **Initialization:** Select an initial admissible (stable) control input $u = -K_1^0 X_0$
2: **repeat**
3: **Policy evaluation:**
4: $Z(k)^T H^{j+1} Z(k) = X(k)^T Q_1 X(k) + (u(k)^j)^T R u(k)^j + Z(k+1)^T H Z(k+1)$
5: **Policy improvement:**
6: $u^{j+1}(k) = -(H_{uu}^{-1})^{j+1} H_{uX}^{j+1} X(k)$
7: **until** H converges

TU Delft

RL for OTC: Summary

Advantages:

- Mathematically rigorous
- Proven to converge

Disadvantages:

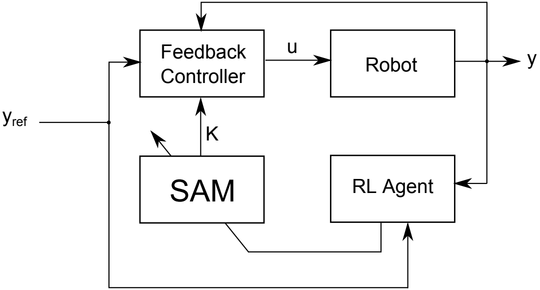
- non-linear RL-based OTC does not exist
- Needs a persistently exciting input, could be dangerous for the robot

Dynamic Tuning via RL

- A feedback control e.g. PID performs well at a certain region
- For different regions, the controller needs to be retuned
- Solution: gain scheduling
- The general diagram:

Dynamic Tuning via RL

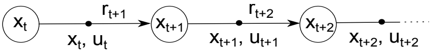
- A feedback control e.g. PID performs well at a certain region
- For different regions, the controller needs to be retuned
- Solution: gain scheduling
- The general diagram:



- Various types of RL algorithm can be used: actor-critic, SARSA, etc

Dynamic Tuning via RL (2): SARSA

- Stands for state-action-reward-state-action
- Policy π as gain modifier, not controller

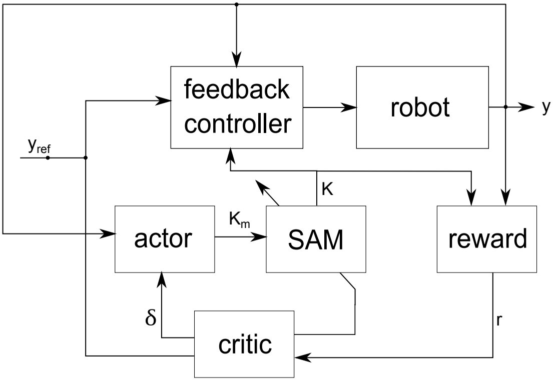


Algorithm 6 SARSA algorithm

Initialization: Initialize $Q(x, u)$ arbitrarily
repeat
    Initialize x
    update π based on Q
    Choose K at x using π
    repeat
        Modify feedback controller using gain K
        Take action u from the controller, observe r, x'
        update π based on Q
        Choose K' from x' using π
        $Q(x, K) \leftarrow Q(x, K) + \alpha[r + \gamma Q(x', K') - Q(x, K)]$
        $x \leftarrow x'$
        $K \leftarrow K'$
    until x is terminal
until episodes run out

Dynamic Tuning via RL (3): Actor-critic

- An alternative to SARSA
- Actor and critic can be parameterized with a basis function e.g. RBF neural network



Dynamic Tuning via RL (4): Summary

Advantages:

- Intuitive and easier to implement

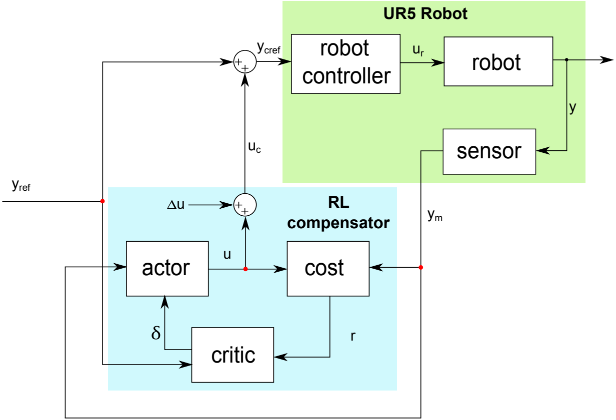
Disadvantages:

- so far only applies to feedback controller
- feedback controller "waits" until error occurs, hence it is always late
- will not perform better due to this reason

TU Delft

Nonlinear Compensator using RL

- A relatively new approach
- Acts as an additive input
- An actor critic RL is proposed



Nonlinear Compensator using RL (2)

- cost function is defined to be similar to LQT

$$r_{k+1} = \rho(y_k, y_k^d, u_k) = (y_k^d - y_k)^T Q (y_k^d - y_k) + u_k^T R u_k \tag{29}$$

- Critic $V(x_k, \theta_k)$ and actor $\pi(x_k, \vartheta_{k-1})$ are parameterized by function approximators e.g. LLR, neural network, etc and respectively.

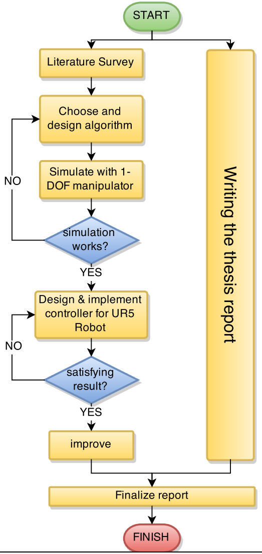
Advantages:

- a relatively new approach, hence interesting for research
- the compensator does not depend directly on the nominal control, hence adding a degree of freedom in control design

Disadvantage:

- Not mathematically as rigorous as RL-based optimal control

Research Plan



Conclusion

- The RL-based additive compensator is chosen as the most promising solution
- Simulation on 1-DoF arm will be performed before starting implementation on UR5 robot
- Possible comparison with ILC

End of presentation

Thank you for the attention

Question?