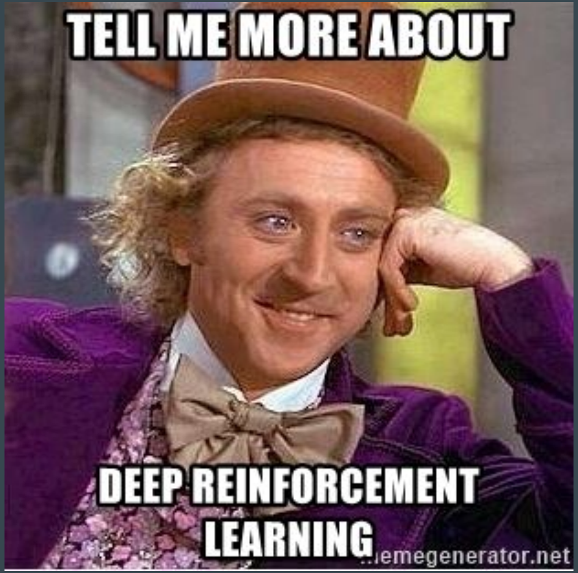
Automating Keyboard Typing

Practical Reinforcement Learning

Aidan Keaveny

Overview

1. Problem Formulation
2. What is Reinforcement Learning?
3. Practical RL for URC Competition
4. Future Work

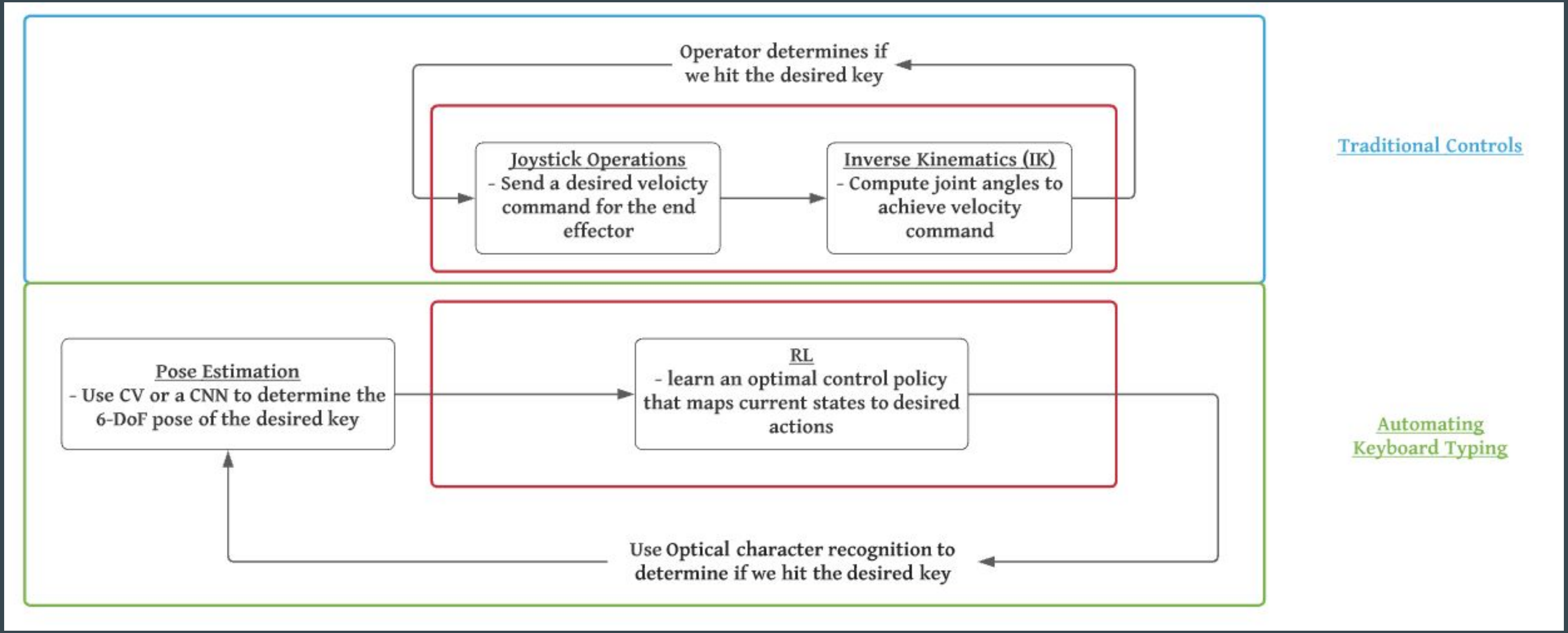


Problem Formulation

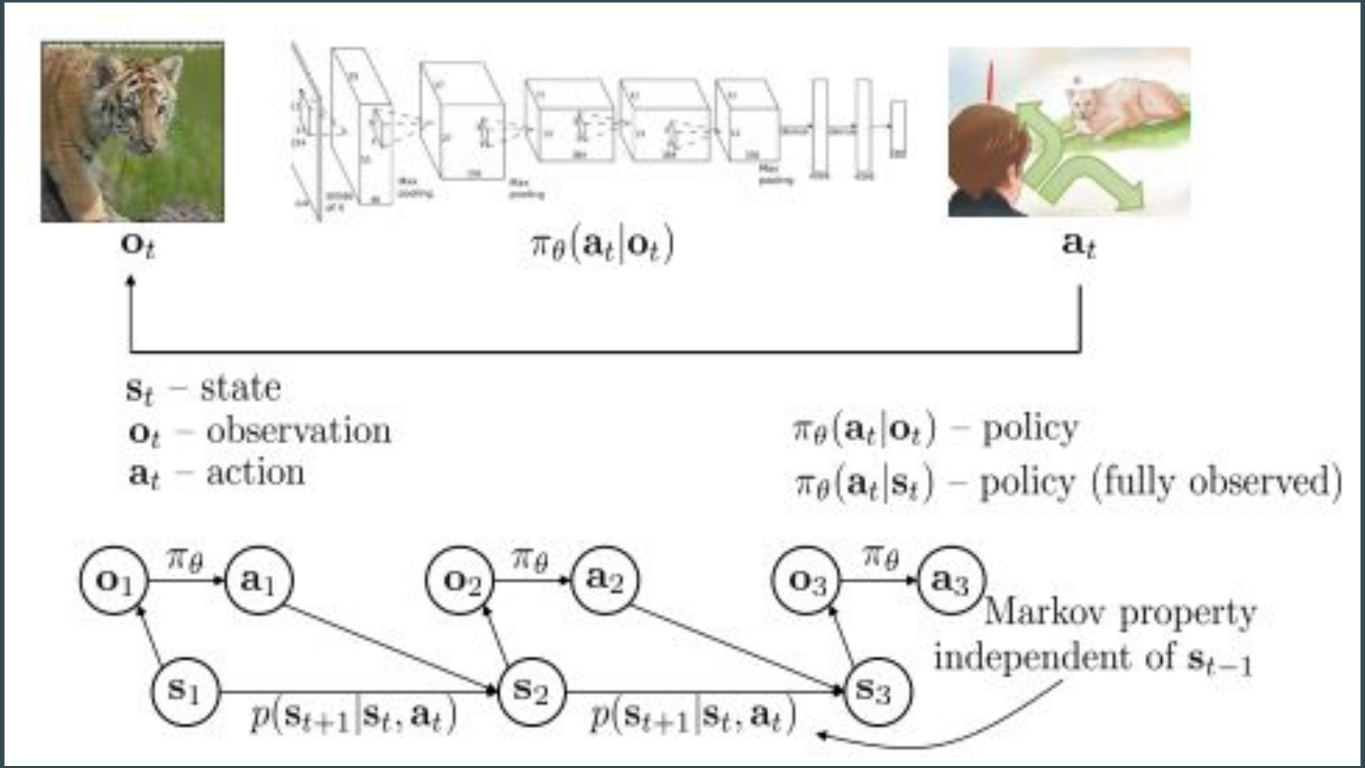
- Keyboard typing is a *high precision task* at URC competition

- Main Goal: Attempt to replace traditional path planning techniques or joystick operations with an optimal control policy that is learnt via Reinforcement Learning.

Problem Formulation

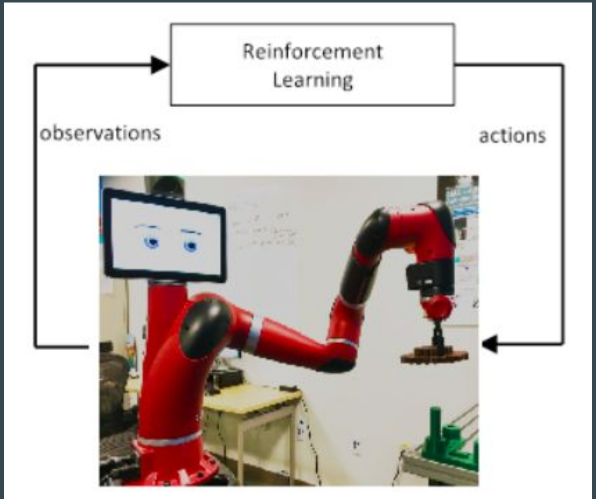


What is RL?


$\mathbf{o}_t$
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$
$\mathbf{a}_t$
$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)
$\mathbf{o}_1$ $\xrightarrow{\pi_\theta}$ $\mathbf{a}_1$
$\mathbf{o}_2$ $\xrightarrow{\pi_\theta}$ $\mathbf{a}_2$
$\mathbf{o}_3$ $\xrightarrow{\pi_\theta}$ $\mathbf{a}_3$
$\mathbf{s}_1$ $\mathbf{s}_2$ $\mathbf{s}_3$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
Markov property
independent of $\mathbf{s}_{t-1}$

What is RL?

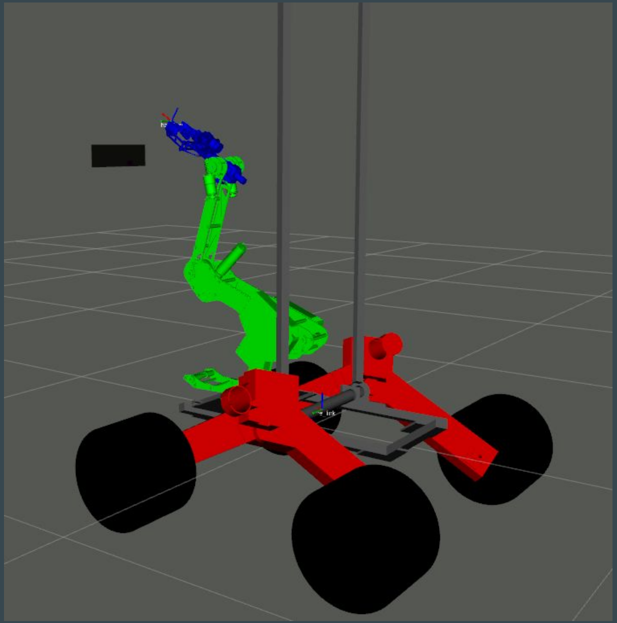
- Env: The agent is always acting in an environment.
- State-Actions Pairs: The agent in one of many states (s) of the environment can choose to take one of many actions (a).
- Model: How the environment reacts to certain actions is defined by a model which we may or may not know.
- Reward: Once an action is taken, the environment delivers a reward (r) as feedback.



What is RL?

- Goal: We want to hit a desired key so we need the 6-DoF Pose ...
- States s_t [1x39]: goal, initial pose, *current* joint positions, velocities, efforts, position and velocity limit switches
- Actions a_t [1x5]: *desired* joint positions
  (position control)
- Reward Function R:

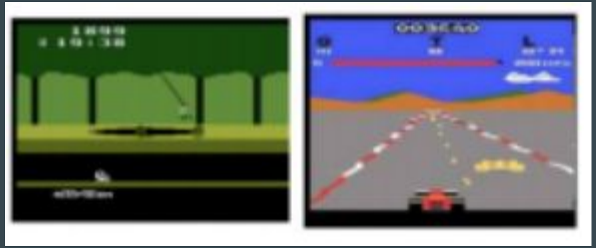
$$\% \text{ distance remaining} = \frac{|\text{distance to target}|}{|\text{initial distance}|}$$

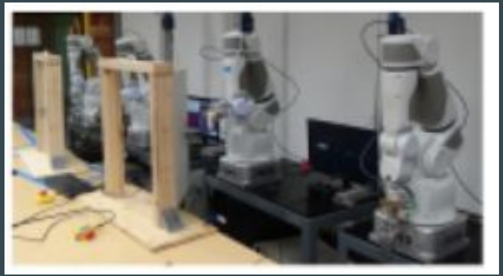


RL Overview

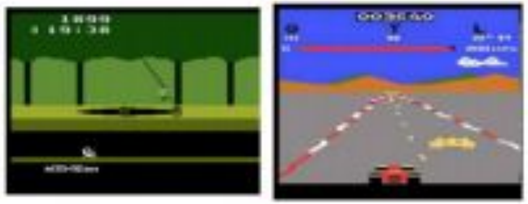
- Game playing: AlphaGo involves both model-free methods (CNN), and also model-based methods (Monte Carlo Tree Search)

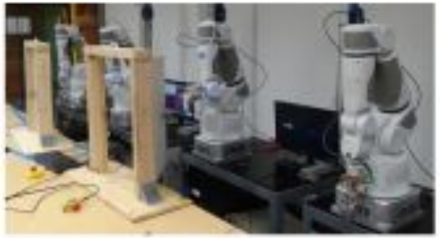
- Control problems: much less structure..
  - Effective representations of the state space s_t
  - Discrete or continuous control actions a_t
  - Engineered reward functions R





RL Overview





Atari games:

Q-learning:

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, et al. "Playing Atari with Deep Reinforcement Learning". (2013).

Policy gradients:

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". (2015).

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, et al. "Asynchronous methods for deep reinforcement learning". (2016).

Real-world robots:

Guided policy search:

S. Levine*, C. Finn*, T. Darrell, P. Abbeel. "End-to-end training of deep visuomotor policies". (2015).

Q-learning:

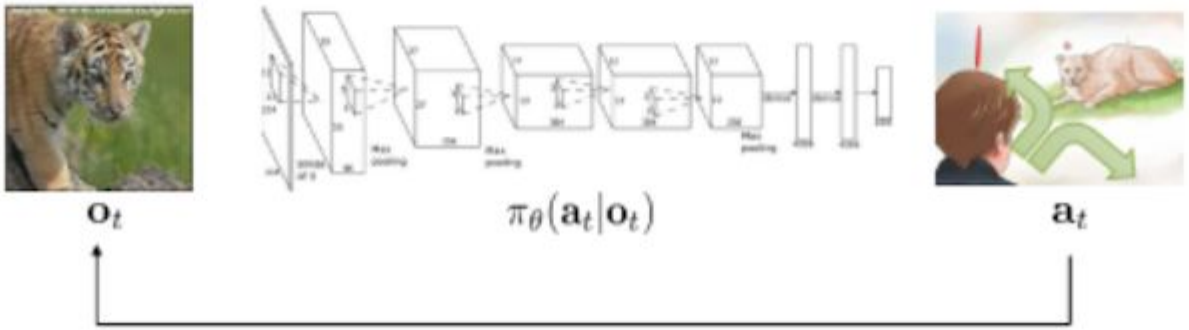
D. Kalashnikov et al. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". (2018).

Beating Go champions:

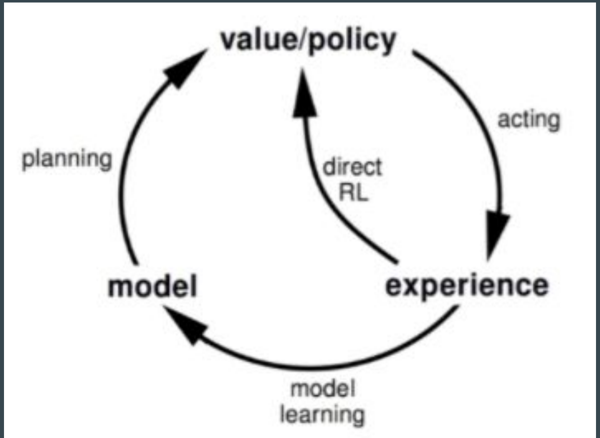
Supervised learning + policy gradients + value functions + Monte Carlo tree search:

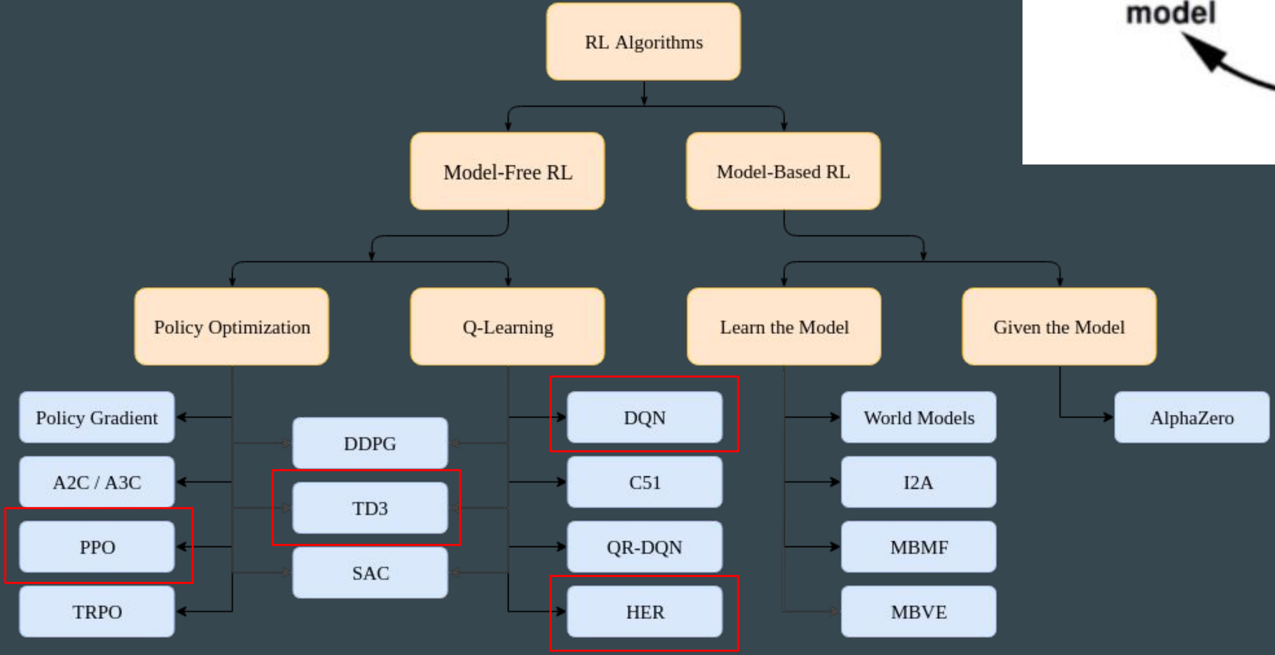
D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, et al. "Mastering the game of Go with deep neural networks and tree search". Nature (2016).

What is RL?

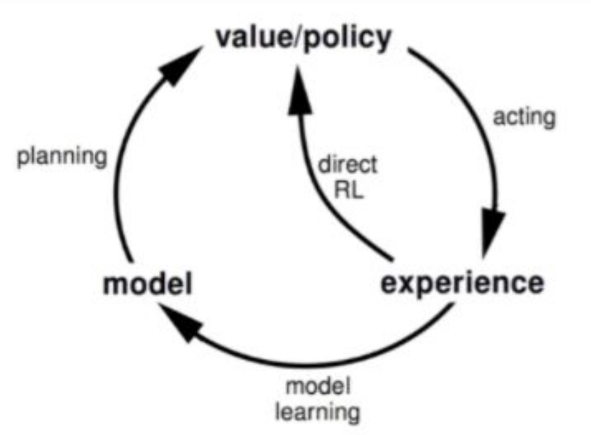


Different Types of RL





Different Types of RL

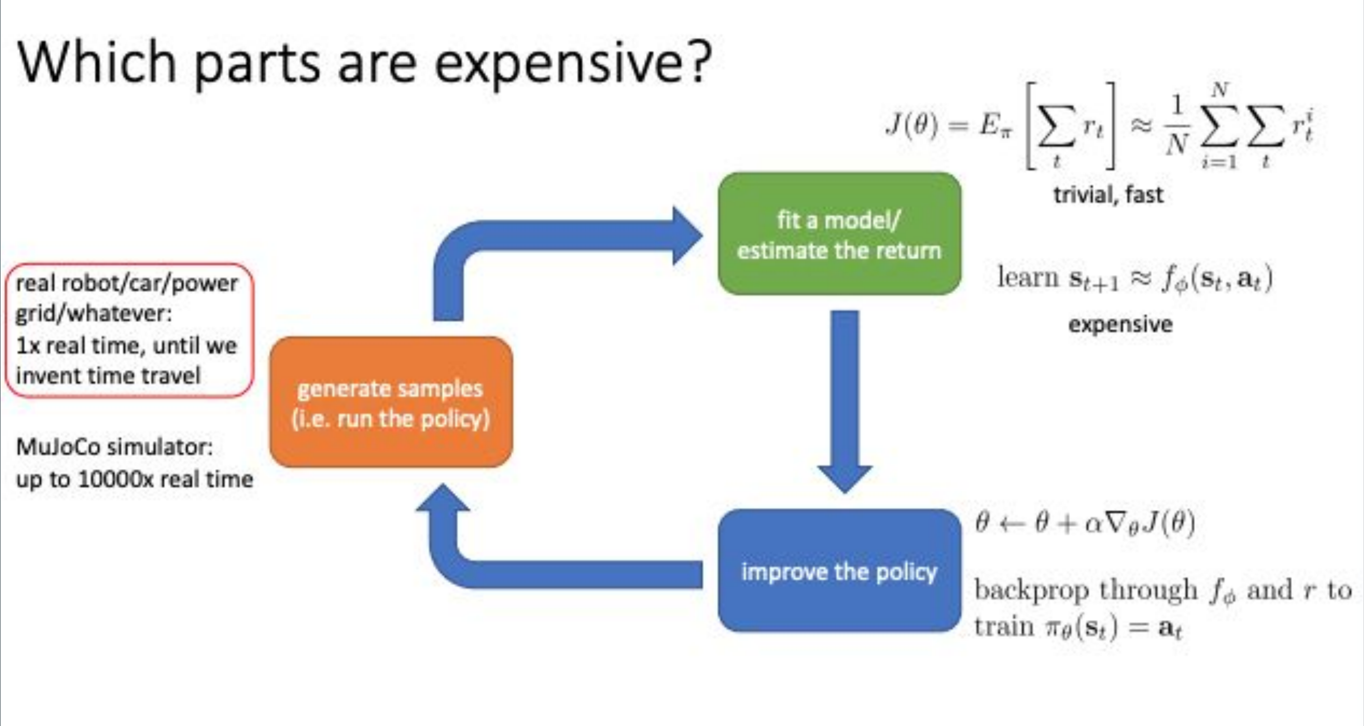


Types of RL algorithms

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

- Policy gradients: directly differentiate the above objective
- Value-based: estimate value function or Q-function of the optimal policy (no explicit policy)
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy
- Model-based RL: estimate the transition model, and then…
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
  - Something else

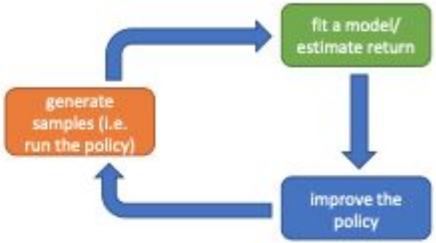
RL Overview


Which parts are expensive?
$J(\theta) = E_\pi\left[\sum_t r_t\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r_t^i$
trivial, fast
fit a model/ estimate the return
learn $\mathbf{s}_{t+1} \approx f_\phi(\mathbf{s}_t, \mathbf{a}_t)$
expensive
real robot/car/power grid/whatever: 1x real time, until we invent time travel
generate samples (i.e. run the policy)
MuJoCo simulator: up to 10000x real time
$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
improve the policy
backprop through f_ϕ and r to train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

RL Overview

Why so many RL algorithms?

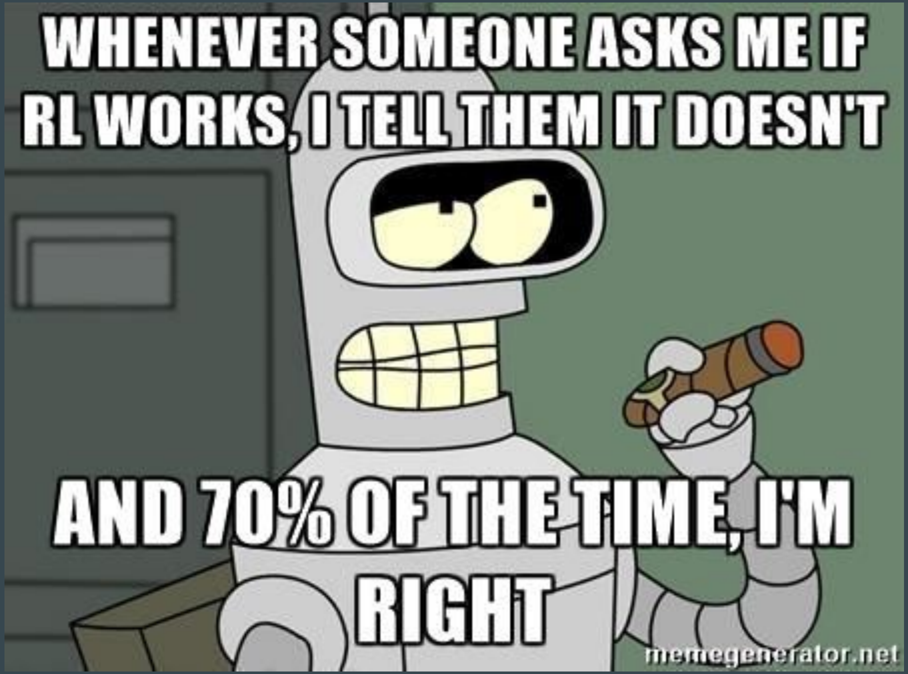
- Different tradeoffs
  - Sample efficiency
  - Stability & ease of use
- Different assumptions
  - Stochastic or deterministic?
  - Continuous or discrete?
  - Episodic or infinite horizon?
- Different things are easy or hard in different settings
  - Easier to represent the policy?
  - Easier to represent the model?



Different Types of RL

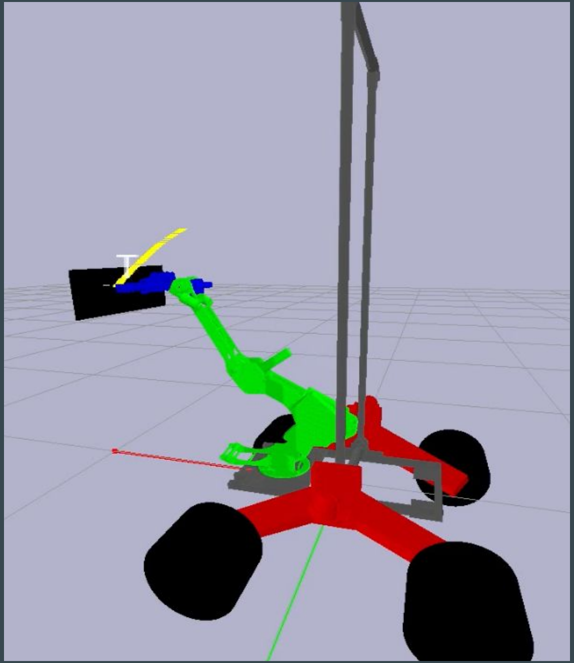
- Learning from demonstrations
  - Directly copying observed behavior
  - Inferring rewards from observed behavior (inverse reinforcement learning)
- Learning from observing the world
  - Learning to predict *"yeah, keyboard typing isn't hard enough .."*
  - Unsupervised learning
- Learning from other tasks
  - Transfer learning
  - Meta-learning: learning to learn

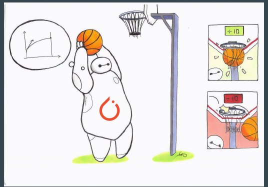
Small break before Practical RL ..



OpenAI + Stable-Baselines3

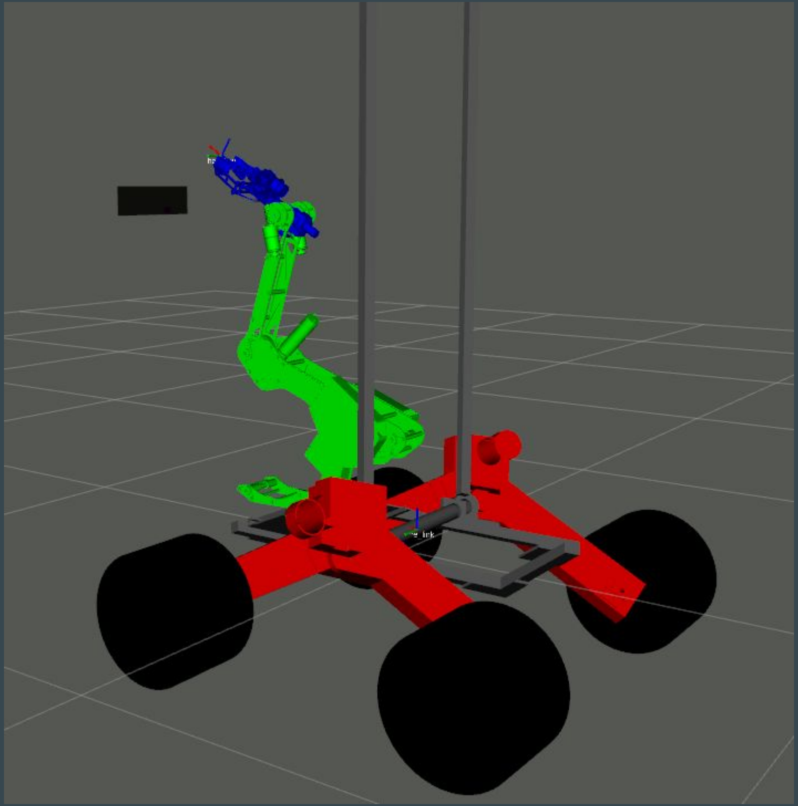
```python
class TestClass:
    NUM_EPISODES = 1
    MAX_STEPS = 5000
    KEY_POSITION = np.array([0.6, 0.6, 0.6])
    KEY_ORIENTATION = np.array([0, 0, 0, 1])

    def __run_test(self, env):
        pp = pprint.PrettyPrinter()  # TODO: update to python 3.8 to use sort_dicts = False

        for episode in range(self.NUM_EPISODES):
            initial_observation = env.reset()
            print('Initial Observation:')
            pp.pprint(initial_observation)

            for sim_step in range(self.MAX_STEPS):
                action = env.action_space.sample()
                observation, reward, done, info = env.step(action)

                print()
                print('Action:')
                pp.pprint(action)
                print('Observation:')
                pp.pprint(observation)
                print('Info:')
                pp.pprint(info)
                print('Reward:')
                pp.pprint(reward)

                if done:
                    print()
                    print(f'Episode #{episode} finished after {info["sim"]["steps_executed"]} steps!')
                    print(f'Episode #{episode} exit condition was {info["sim"]["end_condition"]}')
                    print()
                    break
```

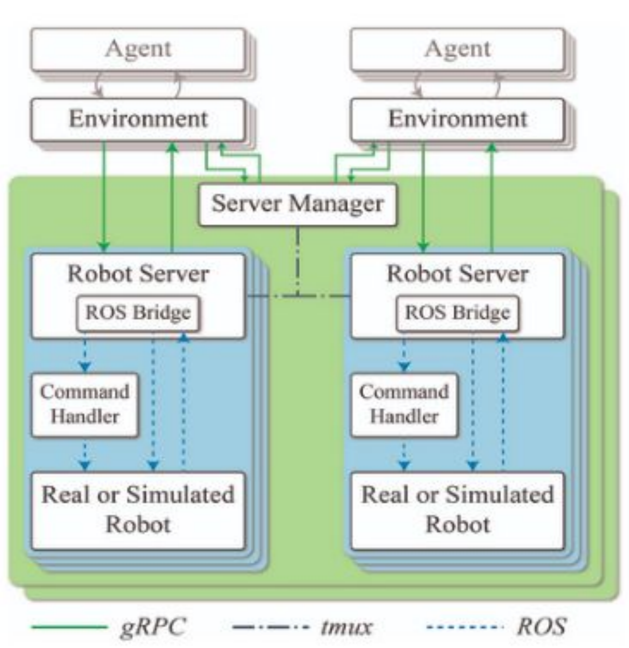




Progress Update: SAR

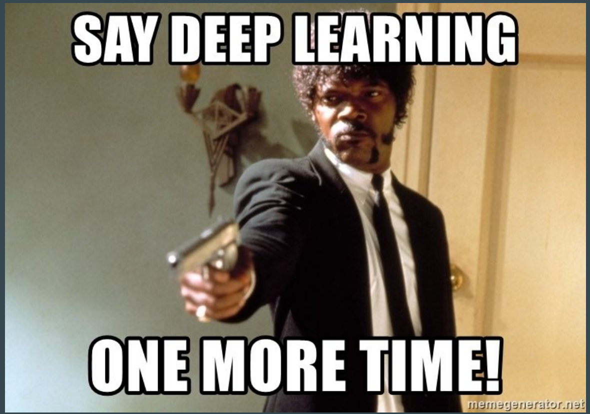
Robo-Gym: *Sim 2 Real ..*

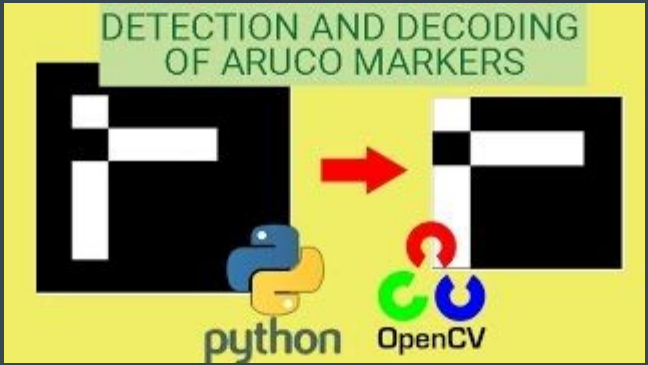


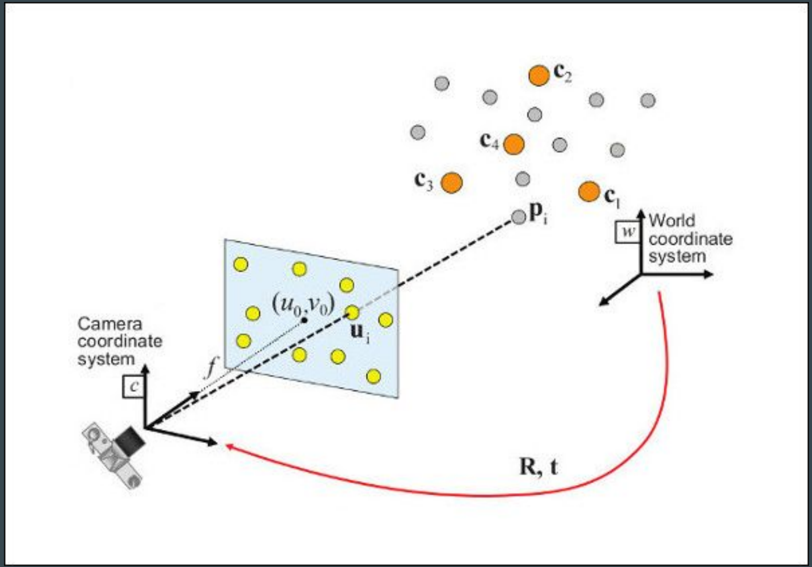


GAZEBO

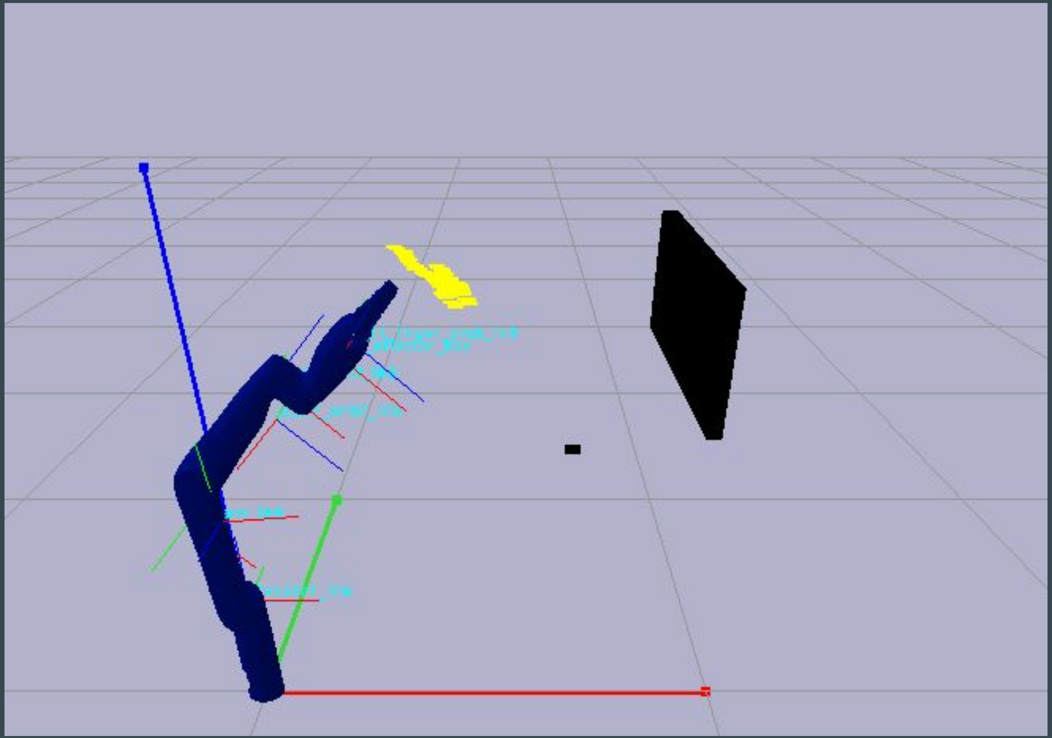
Pose Estimation: *my thesis work..*





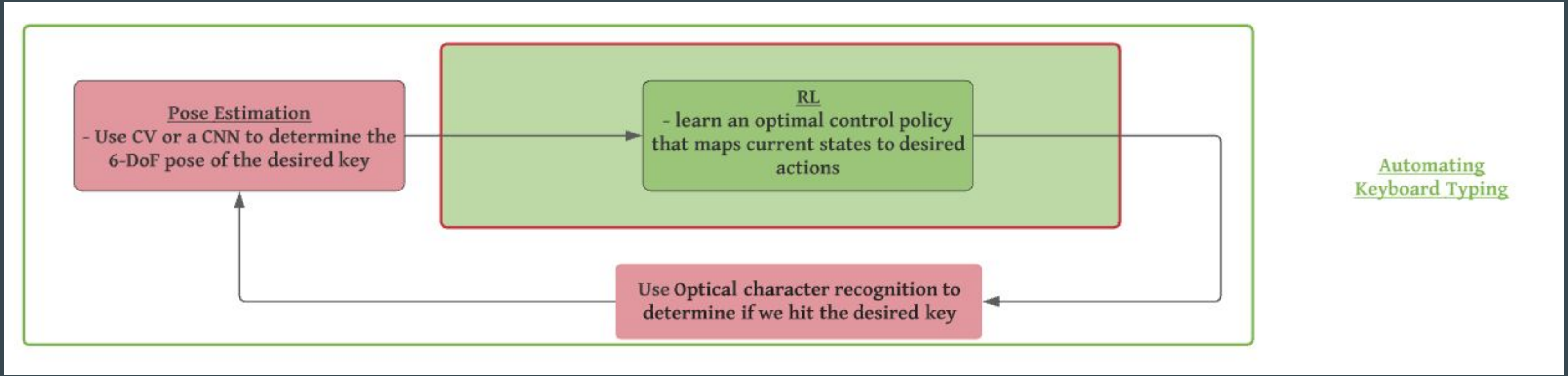


Robohub's Kinova Gen3



Future Work

- Arm: Test arm's IK with closed loop feedback
- RL: Test RL Algorithm with real hardware
- Pose Estimation: Experiments with fixed keyboard position ...
- Robohub: Can always use the Gen3 arm for testing



References

- AlphaGo Documentary
- Sutton & Barto Textbook (U of A !!!)
- CS285: Berkeley's Deep Reinforcement Learning
- OpenAI Gym Tutorial
- Stable -Baselines3
- Robo-Gym