Working in OpenAI Environments & Designing Your Own

Mike Rudd

CS 885 Guest Lecture

May 18, 2018

OpenAI*

- Not-for-profit, funded by private and corporate donations
- Employ small team of high-caliber researchers/advisors
- Promote research towards safe AGI

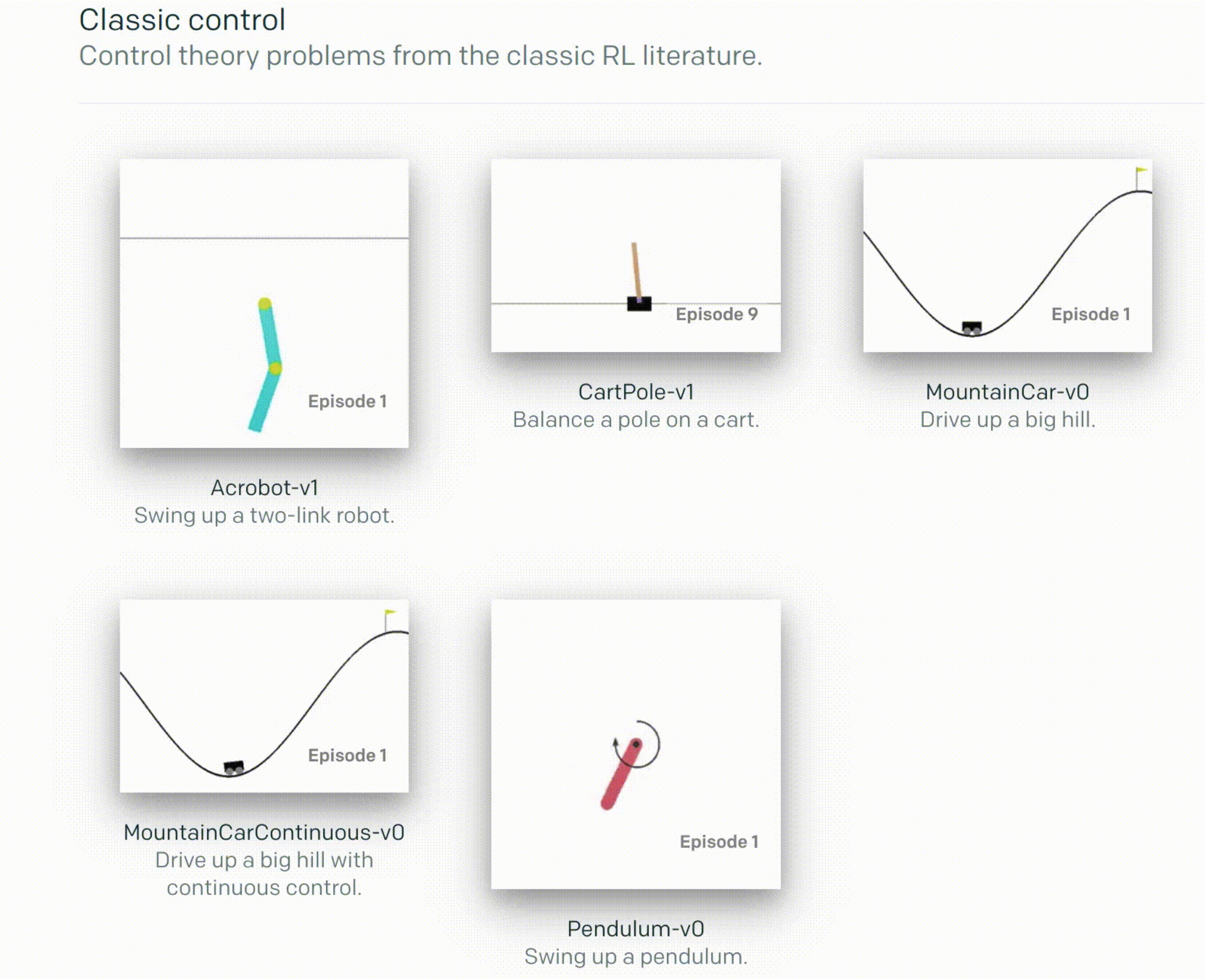
OpenAI

Discovering and enacting the path to safe artificial general intelligence.

*https://openai.com/

OpenAI Gym

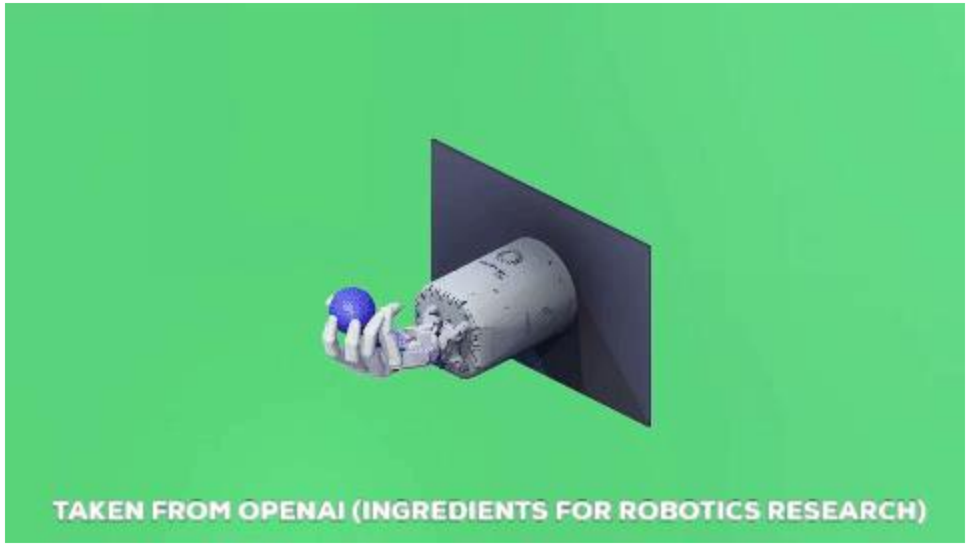
- Standard set of environments for evaluating RL agents
- Provide benchmark for most new algorithms
- Extended to more complex problems as solutions improve



*https://openai.com/

Recent Extensions

- Robotics
  - MuJoCo continuous control tasks now “easily solvable”
  - Harder set of continuous control tasks
- Retro contest
  - Agents can overfit to their environment
  - Train agent that can transfer skills to new environments



Interacting with the Environment

Standardized Code Applicable Across Tasks

Sample Code

```python
import gym

def run(NUM_EPISODES,MAX_STEPS):

    for i in range(NUM_EPISODES):

        # Reset the environment, get the initial state of the episode
        cur_state = env.reset()

        # Episodes are sometimes only allowed to run for a max number
        # of steps, so the training process doesn't get stuck in a loop
        for t in range(MAX_STEPS):

            # Predict an action based on the current state
            action = agent.get_action(cur_state)

            # Take the action in the environment, and observe the
            # resulting next state, reward, and whether the episode
            # finishes. info variable is for debugging purposes
            next_state, reward, done, info = env.step(action)

            # the "next" state is now the current state, we have moved
            # one step forward in time.
            cur_state = next_state

            # done variable signals the episode is over. E.g. the
            # goal was reached or the agent crashed
            if done: break


env = gym.make('CartPole-v0')
agent = dqn.DQNAgent()

run(1e5,200)
```

Building Your Own Environment

Practically more important than beating Gym benchmarks

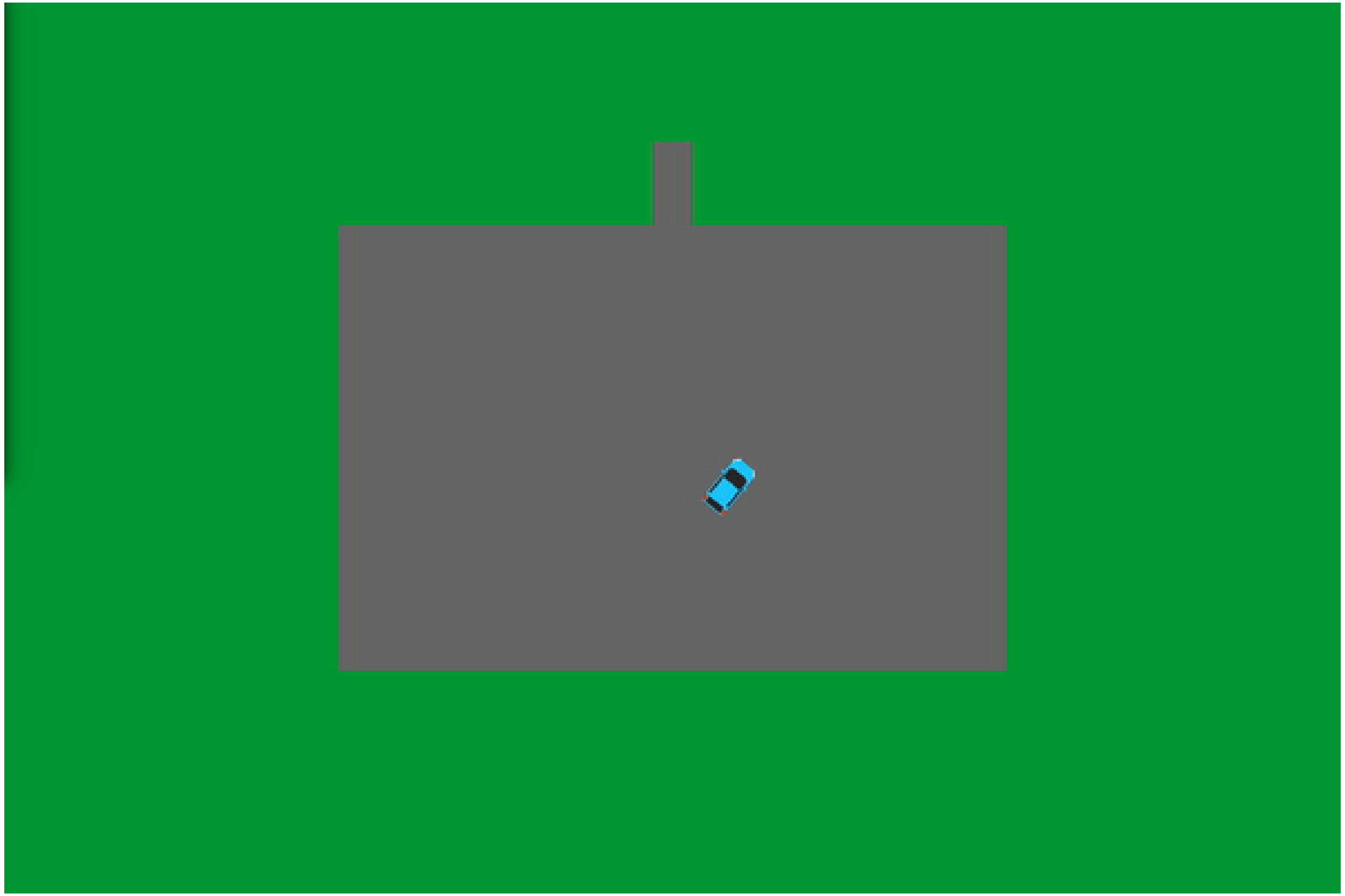
Building Your Own Environment

- Not very difficult

- Just define a Python class with methods for:
  - Initialization
  - Step
  - Reset
  - Render

- Existing packages (physics engines) do most of the heavy lifting
  - Box2D
  - MuJoCo

Example: Teaching a Car to Self-Park



Challenge of Reward Definition

- Major difficulty is in creating reward function
- Algorithms can learn to exploit gaps in our logic, resulting in undesirable behaviours
- See e.g. Ng et al. (1999) for examples and theoretical analysis

Ng, A. Y., Harada, D., & Russell, S. (1999, June). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML* (Vol. 99, pp. 278-287).

Reward Shaping

- Theoretically correct reward is 1 for success and 0 otherwise
- This is sparse though, and in practice is very difficult to learn
- Reward shaping seeks to modify the reward function to speed up learning (with dense signal) but to leave the theoretically optimal policy unchanged
- Ng et al. (1999) show that only shaping function F satisfying the following equation would guarantee that the optimal policy is preserved:

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) \quad \forall s \in S \backslash \{s_0\}$$

```python
parked_reward = 1

if self.out_of_bounds:

    # Don't penalize for hitting wall if in the spot
    if not self.corner_in_spot:
        self.reward -= 0.2

    # Reward for going slow at time of collision
    if self.in_spot:
        self.reward += parked_reward * ((self.max_speed - self.car.speed) / self.max_speed) ** 2

# This is the only one I kept
# The reward is 1 for successfully parking and 0 otherwise
if self.parked: self.reward += parked_reward

# Reward agent if car gets at least partly into spot
if self.corner_in_spot: self.reward += 0.2

# This scales the stopping reward
dist_multiplier = 5

# This reward is for when the car stops moving. If it is close enough to the spot it will get a reward.
if self.car.speed==0:
    self.reward += parked_reward * np.exp(-dist_multiplier * new_dist ** 2 / self.longest_dist)

# Didn't finish writing this reward. Idea was to only reward car if it was pointing at the spot
angle_diff = np.abs(self.compute_angle_to_spot()-self.car.hull.angle) / math.pi
if (angle_diff < 0.1 or angle_diff > 0.9):
    # Do something
```

```python
# Asymmetrically penalize the agent for getting closer or further from the spot.
# This was meant to discourage circling behaviour observed.
if (old_dist - new_dist) >= 0:
    self.reward += (old_dist - new_dist) / 10
else:
    self.reward += (old_dist - new_dist) / 5


# Reward (penalty) for keeping wheels straight (turning)
self.reward -= np.abs(self.car.steering) + 0.05

smooth_reward = 0.05
# Reward for smooth actions
for i in range(len(self.last_action)):
    self.reward += smooth_reward * np.exp(-20 * np.abs((action[i]-self.last_action[i])))

# Time penalty to encourage algorithm to finish quickly
self.reward -= 0.5
```