Effective Reinforcement Learning for Mobile Robots

By William D. Smart and
Leslie Pack Kaelbling

Presented by Gal Peleg
CSCI2950-Z, Brown University
March 1, 2010

Proceedings of IEEE International Conference on Robotics and Automation (ICRA 2002), volume 4, pages 3404-3410, 2002.

Presentation Outline

- Forecast
- Background
  - Motivation &Problem Statement
  - The World of Reinforcement Learning (RL)
  - The Q-Learning Algorithm
  - RL Applied to Mobile Robots
  - The Learning Framework: Inclusion of Prior Knowledge
- Experimental Results
  - Corridor Following
  - Obstacle Avoidance
- Conclusions
- Future Work

Forecast

- It's easier and more intuitive for the programmer to specify *what* the robot should be doing
- Having a robot *learn* how to accomplish a task, rather than being told explicitly is an appealing idea
- The Authors introduce a framework for reinforcement learning (RL) on mobile robots and describe experiments that validate its performance

Motivation &Problem Statement

- Challenges
  - Programming robots can be very time-consuming
    - Many iterations to fine-tune low-level mapping from sensors to actuators
  - Robots' sensors and actuators are different from those of humans
  - Difficult to translate knowledge about a task into terms useful for the robot
- Instead…
  - Provide some high-level specification of the task and use machine learning to "fill in the details"

The World of Reinforcement Learning

- Can be described by
  - A set of states S, and a state of actions A
- At each (discrete) time step
  - Agent observes state s_t of the world
  - Chooses an action a_t to take
  - Is then given a reward r_{t+1}
    - Reflects how good the action was in a short-term sense
  - Observes new state of the world s_{t+1}
- Goal
  - Use tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ to learn a mapping from the state-action pair to an optimal value function

The Q-Learning Algorithm

- Q-Function
  - Is typically stored in a table, indexed by state and action
  - Usually starts with arbitrary values
- We iteratively approximate the optimal Q-Function based on our observation of the world

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{old\ value} + \underbrace{\alpha_t(s_t,a_t)}_{learning\ rate} \times \left[\overbrace{\underbrace{r_{t+1}}_{reward} + \underbrace{\gamma}_{discount\ factor} \underbrace{\max_a Q(s_{t+1},a)}_{max\ future\ value}}^{expected\ discounted\ reward} - \overbrace{Q(s_t,a_t)}^{old\ value} \right]$$

- Considering all possible actions given a state, we select the one with the largest Q-value

$$\pi^*(s) = \arg\max_a Q(s,a).$$

Blackjack Q-Learning Example

```java
    public static final int numSleep = 100;

    /** The number of cards left in the deck before cutting off and re
    public static int CUT_OFF_SIZE = 10 * numPlayers;

    /** The minimum bet allowed in this simulation. */
    public static double MIN_BET = 5.0;

    public static final double ALPHA = 0.1; //learning rate
    public static final double GAMMA = 0.9; //discount factor

    public static final int COUNT_STATES = 3;
}
```

Problems | @ Javadoc | Declaration | Console | Error Log

<terminated> BlackjackSimulator [Java Application] /System/Library/Frameworks/JavaVM.framework/

Reinforcement Learning Applied to Mobile Robots

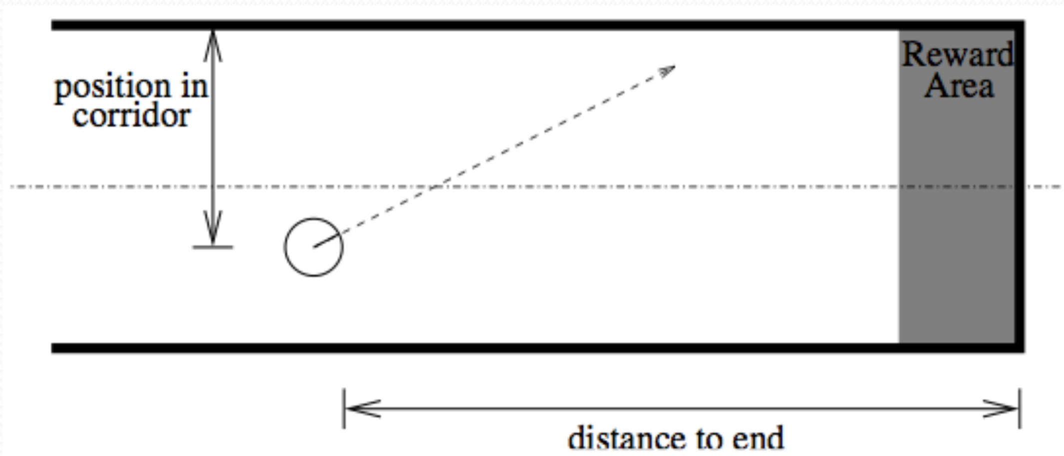
- Makes sense because
  - We can design a much higher-level task description in the form of the reward function, $R(s,a)$
- Shortcomings
  - Q-learning requires discrete states and actions
    - Authors combat this by using a suitable value-function approximation technique (i.e. the HEDGER algorithm)
  - Sparse reward functions
    - Combated through "Inclusion of Prior Knowledge," the meat and potatoes of the authors' learning framework

The Learning Framework: Inclusion of Prior Knowledge

- First phase
  - Value-Function approximation is not complete enough to control the robot
  - Robot is therefore supplied control policy
    - Can be through actual control code or teleoperation
    - Exposes the RL system to “interesting” parts of the state space
  - RL system passively watches states, action, and rewards
    - We use these to bootstrap the value-function approximation
- Second phase
  - Full control is handed back to the standard RL system
    - Robot is now capable of finding reward-giving states

Corridor Following: The Setup

- State Space Contains 3 Dimensions
  - Distance to end of corridor, Distance from left hand wall, Angle to target point
- Rewards
  - +10 for reaching end of corridor, 0 for anything else
- Phase 1 tested using
  - Coded control policy, direct control examples, and simulation



Corridor Following: Results

- Coded Control Policy
  - Statistically indistinguishable from “optimal”
- Direct Control Examples
  - Also statistically indistinguishable from “optimal”
  - Experienced more varied, so framework is able to generalize more effectively
- Simulation
  - Fastest simulation time > 2 hours
  - Both phase 1 learning attempts above were done in 2 hours

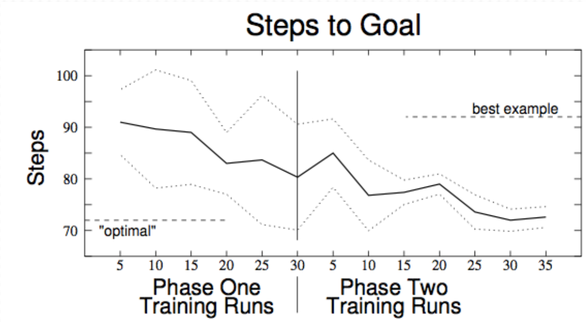


Fig. 4. Corridor following performance with simple policy examples.

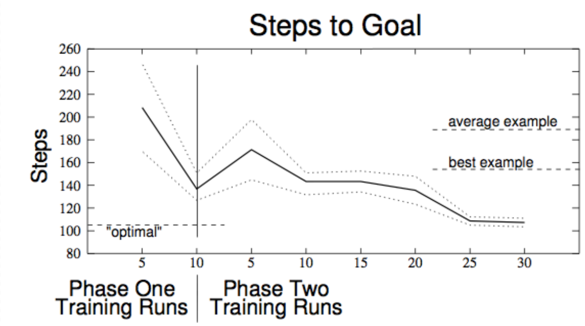


Fig. 5. Corridor following performance with direct control examples.

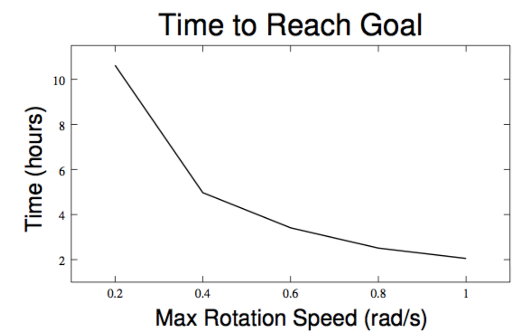
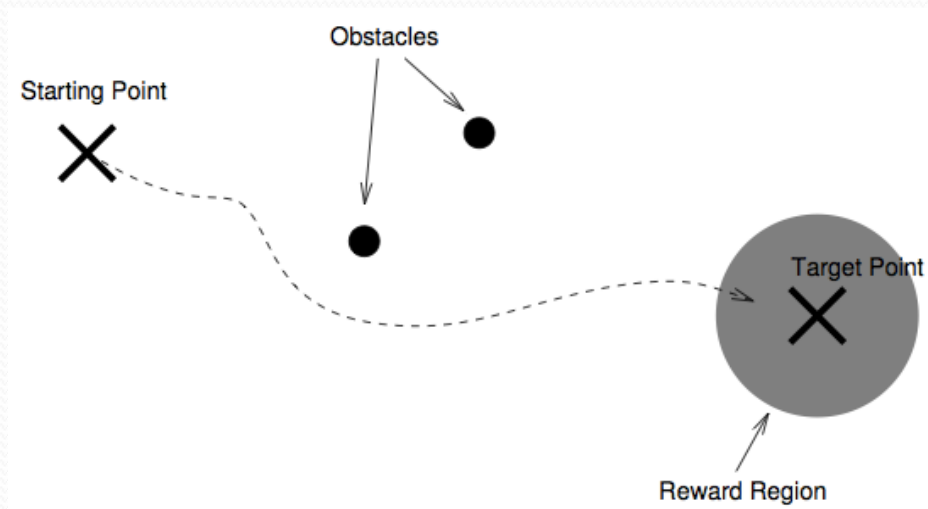


Fig. 6. Performance on the simulated corridor following task.

Obstacle Avoidance: The Setup

- State Space Contains 2 Dimensions
  - Distance to goal, Direction to goal
- Rewards
  - +1 for reaching target, -1 for collision with obstacle, otherwise 0
- Phase 1 tested using
  - Only direct control examples, and simulation
- Much harder task



Obstacle Avoidance: Results

- Direct Control Examples
  - Statistically indistinguishable from "optimal"
- Simulation
  - Took more than 6 hours to complete the task, and reached the goal only 25% of the time

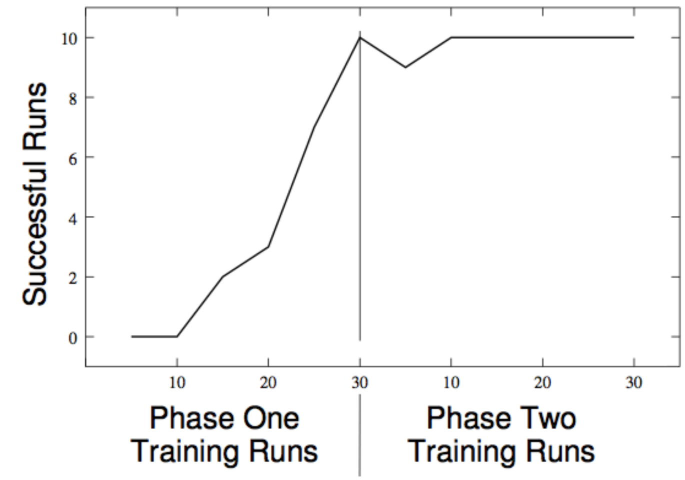


Fig. 9. Successful runs (out of 10) for the obstacle avoidance task.

	Starting distance		
	1m	2m	3m
Successful	46.2%	25.0%	18.7%
Time (hours)	2.03	6.24	6.54

TABLE I

PERFORMANCE ON THE SIMULATED OBSTACLE AVOIDANCE TASK.

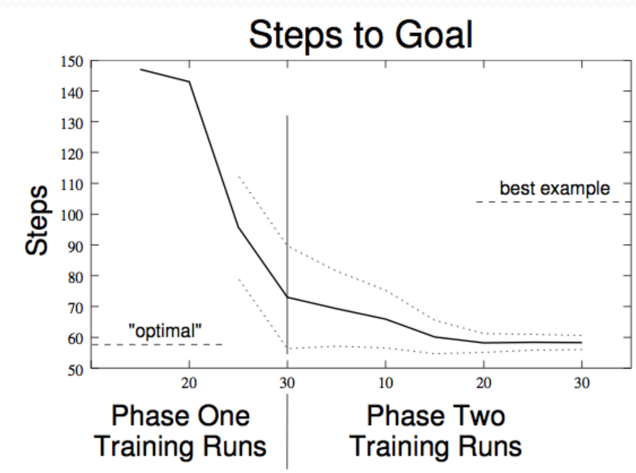


Fig. 10. Performance on the obstacle avoidance task.

Conclusions

- 1. Final performance for both tasks is significantly better than any of the examples used in phase 1 training

- 2. Using example trajectories allows us to incorporate *human knowledge* about how to perform a task in the learning system

- 3. The framework is capable of learning good control policies more quickly than moderately experienced programmers can hand-code them

Future Work

- How complex a task can be learned with sparse reward functions?
- How does the balance of “good” and “bad” phase one trajectories affect the speed of learning?
- Can we automatically determine when to change learning phases?