Reinforcement Learning and Motion Planning

Mrinal Kalakrishnan
University of Southern California

August 25, 2010
Reinforcement Learning

- Holy grail of learning for robotics
- Curse of dimensionality...

- Trajectory-based RL
- High dimensions
- Continuous states and actions
- State-of-the-art: Policy Improvement with Path Integrals - Theodorou et al., 2010
Reinforcement Learning

- Holy grail of learning for robotics
- Curse of dimensionality...

- Trajectory-based RL
- High dimensions
- Continuous states and actions
- State-of-the-art: Policy Improvement with Path Integrals - Theodorou et al., 2010
Motion Planning

- Sampling-based planners
  - Solve very difficult problems
  - Jerky paths, require smoothing
  - Feasible paths, not optimal

- Optimization-based planners
  - CHOMP (Ratliff et al., 2009)
  - Covariant gradient descent
  - Smooth trajectories
  - Solves “easy” problems
  - Local minima
Motion Planning

- Sampling-based planners
  - Solve very difficult problems
  - Jerky paths, require smoothing
  - Feasible paths, not optimal

- Optimization-based planners
  - CHOMP (Ratliff et al., 2009)
  - Covariant gradient descent
  - Smooth trajectories
  - Solves “easy” problems
  - Local minima
Apply PI² to motion planning

- Create a new policy: \( \dot{x} = K(u - x) \)
- Control command \( u(t) = x(t+1) \)
- Quadratic control cost: \( u^T Ru \)
- \( R = A^T A \):
  - \( A \) is an acceleration differentiation matrix
  - \( R \) measures squared accelerations
- Cost = control cost + state costs
- State costs can include:
  - Collision cost
  - Energy cost
  - Constraint violation cost
- Need not be differentiable!
Apply PI$^2$ to motion planning

- Create a new policy: $\dot{x} = K(u - x)$
- Control command $u(t) = \text{state } x(t + 1)$
- Quadratic control cost: $u^T Ru$
- $R = A^T A$
  - $A$ is an acceleration differentiation matrix
  - $R$ measures squared accelerations
- Cost = control cost + state costs
- State costs can include:
  - Collision cost
  - Energy cost
  - Constraint violation cost
- Need not be differentiable!
Apply PI² to motion planning

- Create a new policy: \( \dot{x} = K(u - x) \)
- Control command \( u(t) = \text{state } x(t+1) \)
- Quadratic control cost: \( u^T R u \)
- \( R = A^T A \):
  - \( A \) is an acceleration differentiation matrix
  - \( R \) measures squared accelerations
- Cost = control cost + state costs
- State costs can include:
  - Collision cost
  - Energy cost
  - Constraint violation cost
- Need not be differentiable!
Apply PI² to motion planning

- Create a new policy: \( \dot{x} = K(u - x) \)
- Control command \( u(t) = \text{state } x(t + 1) \)
- Quadratic control cost: \( u^T R u \)
- \( R = A^T A \):
  - \( A \) is an acceleration differentiation matrix
  - \( R \) measures squared accelerations
- Cost = control cost + state costs
- State costs can include:
  - Collision cost
  - Energy cost
  - Constraint violation cost
  - Need not be differentiable!
Collision cost

- Distance field / distance transform
- Answers clearance and penetration depth queries
- Voxelize robot body and add up costs for each voxel
Collision cost

- Distance field / distance transform
- Answers clearance and penetration depth queries
- Voxelize robot body and add up costs for each voxel
The algorithm

- Generate initial straight-line trajectory
- Repeat until convergence:
  - Create noisy rollouts around the trajectory
    Noise does not modify start or goal due to $\Sigma = R^{-1}$!
  - Compute costs for each rollout
  - Apply PI$^2$ update: reward-weighted average
The algorithm

Initial trajectory
The algorithm

Noisy rollout

Mrinal Kalakrishnan

Reinforcement Learning and Motion Planning
The algorithm

Noisy rollout

Mrinal Kalakrishnan
Reinforcement Learning and Motion Planning
The algorithm

Noisy rollout
The algorithm

Noisy rollout

Mrinal Kalakrishnan
The algorithm

Noisy rollout
The algorithm

Noisy rollout

Mrinal Kalakrishnan
Reinforcement Learning and Motion Planning
The algorithm

Updated trajectory

Mrinal Kalakrishnan

Reinforcement Learning and Motion Planning
Video: Pole

Updated trajectory
<table>
<thead>
<tr>
<th>Condition</th>
<th>Success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconstrained</td>
<td>39 / 42</td>
</tr>
<tr>
<td>Constrained</td>
<td>38 / 42</td>
</tr>
</tbody>
</table>
Video: Real-world Reinforcement Learning and Motion Planning
Conclusion

- Optimization-based motion planner that does not require gradients
- Generates collision-free, smooth trajectories
- Optimizes arbitrary secondary criteria (constraints, torques)
- May handle local minima better than CHOMP (needs further testing)
- ICRA 2011 submission pending
- Code is in the `optimization_motion_planning` package, coming soon to a sandbox near you...
Future Work

- Torque optimality
- Trajectory libraries, cached plans

Thanks:
- Sachin Chitta
- Peter Pastor
- Willow Garage
Future Work

▶ Torque optimality
▶ Trajectory libraries, cached plans

Thanks:
▶ Sachin Chitta
▶ Peter Pastor
▶ Willow Garage