Trajectory Prediction: Learning to Map Situations to Robot Trajectories

Nikolay Jetchev and Marc Toussaint
TU Berlin
Berlin Machine Learning and Robotics Group

June 17th, 2009
ICML
Talk content

- Movement generation and motivation
- Define trajectory prediction
- Describe the prediction algorithm
- Experiments, results and conclusions
Biological Inspiration

- Humans and animals execute complex movements 'instantly'
- Don't plan a trajectory, but quickly choose a good movement

Reactive trajectory policy

- How do people manage to do it?
  - Experience in repeating movements
  - Capabilities of our bodies (proprioception)
  - Familiar obstacles and spaces
Apply to robot movement planning

- Robots are slower to generate movement, optimize from scratch without prior knowledge

- A novel approach – **trajectory prediction**

- The essence: a mapping from a situation to a whole trajectory
  - Observe patterns in situation-movement pairs
  - Choose quickly an approximately good trajectory

- Gain for robotics: speed up movement generation by learning an approximate situation-movement mapping
Robot movement basics

- $q_t$ is the robot posture, angles of all joints at time $t$

- The obstacle positions and $q_t$ fully define the current situation.

- Trajectory $q = (q_0, \ldots, q_T)$ for some time horizon $T$
Planning as cost function optimization

- A cost function characterizes good movements for the specified tasks

- Energy efficient trajectories with no collisions have low cost

\[ C(x, q) = \sum_{t=1}^{T} g(q_t) + h(q_t, q_{t-1}) \]

- A trajectory optimization algorithm finds the best movement for a given situation

\[ x \mapsto q^* = \arg\min_q C(x, q) \]
Trajectory prediction for faster optimization

- Approximate the mapping \( x \mapsto q^* \)

- Gather data \( D = \{(x_i, q_i)^d_{i=1}\} \) of optimized movements
  - Use any classical method for movement optimization

- Use this data to find quickly good initial trajectories
  - Methods like iterated Linear Quadratic Gaussian (iLQG) and gradient descent very sensitive to initialization
  - Good starting values can improve them drastically
Prediction algorithm overview

- Train: gather data $D$ and learn to predict

- Test input: situation $x$

  - Predict trajectory $q_i$ from $D$

  - Transfer $q_i$ to the current situation $x$

- Test output: transferred trajectory $q^*$
Situation descriptor

The world situation is specified by robot posture and object positions.

Model situation representation $x$ as a high-dimensional feature vector $x = (q_0, p, o) \in \mathbb{R}^{791}$:

- $q_0$ is initial posture vector
- $p$ is vector of pairwise distances btw centers of 20 objects
- $o$ is vector of z axis cosine between 20 objects in scene
Feature selection

Why these features?

- Much information about world situation
- Highly redundant, a lot of coordinate systems and measurements

Feature selection can refine the descriptor

- sparse 50 features

\[ P(f(x) = T_{x_i}q_i) = \frac{1}{Z} \exp\left\{ -\frac{1}{2}(x - x_i)^TW(x - x_i) \right\} \]

\[ E \{ C(x, f(x)) \} = \sum_{i=1}^{d} P(f(x) = T_{x_i}q_i)C(x, T_{x_i}q_i) \]

\[ \min_{w} \sum_{x \in D} E \{ C(x, f(x)) \} + \lambda |w|_1 \]
NN for prediction

- Nearest neighbor over database situations – NN

\[ k(x^*, x_i) = \exp\left\{-\frac{1}{2}(x^* - x_i)^T W (x^* - x_i)\right\} \]

as a diagonal Gaussian similarity metric

- \[ \hat{i} = \arg\max_i k(x_i, x) \]

The predicted movement is \( q_{\hat{i}} \), the trajectory of the most similar previous situation
Classification for trajectory prediction

- Classify situations according to movement type – **Cluster**
  - Select a smaller representative movement subset by clustering $\mathcal{D}$
  - Take cluster average movements as new trajectory set
  - Gather dataset with lowest cost prototypes for each movement
  - Train a SVM on this data
Situation transfer

- Repeating a movement in joint space is not likely to be good
  - Different object positions between situations

- We need to transfer to the new situation $x$
  - Task space -coordinates of finger relative to obstacle
  - Project from joint space to task space and then back project via inverse kinematics (IK)
    - Prioritized IK avoid collisions
  - Call this the transfer operator $T_{xi}$
Experiments

- Use in simulation a humanoid torso with 31 joints
- Reach red point target with finger without colliding with the table
- Generate world scenarios by randomly sampling robot posture, target and obstacle positions
Compared methods

- iLQG with different initializations
  - NN prediction
  - Cluster prediction
  - Linear interpolated path
  - Rapidly Exploring Random Tree (RRT) path

- Good initial trajectory will speed-up iLQG convergence
  - It corresponds to a reasonable table avoidance path

- A single iLQG iteration: 0.065 s, prediction + transfer – 0.1s, RRT – more than 1.5s for 2000 nodes
Results: time until convergence

- **Cluster** converges in 98.4% of scenarios to correct results (linear 85%)

- **NN** converges much faster
  - 0.32s for first feasible solution (linear 1.3s)
  - 0.9s for convergence (linear 1.96s)

- Sparse feature selection improves results
Results: iLQG iteration analysis

- Another view: cost convergence per iLQG iteration

- The prediction methods achieve very low costs in few iterations

- RRT not competitive: much slower than linear
Conclusions and future work

- Trajectory prediction can speed-up computation drastically
- Data representation should be carefully designed to transfer knowledge
  - Information in situation descriptor and transfer task space
- Future directions: more challenging movement problems
  - Dynamic worlds
  - Cluttered scenes
  - More complex manipulations - grasping
Thank you for your time.

More info at http://user.cs.tu-berlin.de/~jetchev/