CS545—Contents XIII

- Trajectory Planning
  - Control Policies
  - Desired Trajectories
  - Optimization Methods
  - Dynamical Systems

- Reading Assignment for Next Class
  - See http://www-clmc.usc.edu/~cs545
Learning Policies is the Goal of Learning Control

- Policy:
\[ u(t) = p(x(t), t, \alpha) \]

Internal & External State: \( x(t) \)  
Action: \( u(t) \)
Dynamic Programming & Reinforcement Learning

- **Dynamic Programming**
  - requires a model of the movement system

- **Reinforcement Learning**
  - can work without models of the movement system

- **Essentials**
  - both techniques require to learn a high-dimensional “value function” that assesses the quality of an action $u$ in a state $x$
  - learning the value function is a complex nonstationary, nonlinear learning process
  - both methods die the curse of dimensionality
Desired Trajectories

- **Essentials**
  - prescribe a desired trajectory
    \[
    (\theta, \dot{\theta})_{\text{desired}} = f(\xi_{\text{initial}}, \xi_{\text{target}}, t)
    \]
  - convert desired trajectory into a (time-dependent) control policy, e.g., by PD-controller
    \[
    u = p(x, t, \alpha) = k_\theta (\theta(t)_{\text{desired}} - \theta) + k_{\dot{\theta}} (\dot{\theta}(t)_{\text{desired}} - \dot{\theta})
    \]
- **Problems**
  - Where do desired trajectories come from
  - How to accomplish reactive control
  - How to generalize to new tasks or new situations
Desired Trajectories (cont’d)

- There is a difference between PATH and TRAJECTORY planning
  - A trajectory involves geometry AND time
  - A path involves only geometry
- Planning can happen either in joint or operational space
  \[ x_d = g(t, \alpha) \]
  \[ \text{or} \]
  \[ \theta_d = f(t, \alpha) \]
- There is usually an infinity of possible desired trajectories
- How is the desired trajectory represented?
  - Every point in time?
  - Only start & final point?
  - Via points?
- Movement Primitives
Joint Space Planning

- What could one plan?
  - Arbitrary trajectories from start to end
  - Trapezoidal (or any another kind of) velocity profiles
  - Polynomials:
    - 1.order: straight lines
    - 2.order: parabolas
    - 3.order: cubic splines
    - 5 order: quintic splines
  - Interesting:
    - Analyze the shape of the trajectories in position, velocity, acceleration, and jerk space.
    - How many constraints are needed to specify a trajectory
Example: Cubic Polynomial

- **Cubic Polynomial:**
  
  \[ q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \]
  
  \[ \dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 \]
  
  \[ \ddot{q}(t) = 2a_2 + 6a_3 t \]

- **Given: Start & Endpoint**
  
  \[ q_s, q_f \]

- **Plan a cubic polynomial through the start and endpoint**
  
  - Two additional constraints are needed, for instance:
    
    \[ \dot{q}_s, \dot{q}_f \text{ or } \dot{q}_s, \ddot{q}_s \text{ or } \dot{q}_f, \ddot{q}_f \]
  
  - Determine the coefficients by using 4 boundary conditions, e.g.,
    
    \[ q_s = a_0 \]
    
    \[ \dot{q}_s = a_1 \]
    
    \[ q_f = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \]
    
    \[ \dot{q}_f = a_1 + 2a_2 t + 3a_3 t^2 \]
Planning Complex Paths

- Prescribe a set of via-points
- Plan simple trajectories between via-points
- Ensure smooth transitions between trajectory segments
  - E.g., the tangent of two adjacent trajectory segments should match
Optimization Approaches to Desired Trajectories

- **Given:**
  - “hard constraints”, e.g., \( q_s, q_f, t \)
  - “soft constraints”, i.e., an optimization criterion

- **Goal:**
  - Find the trajectory that fulfills the hard constraints while minimizing (or maximizing) the soft constraint

- **Solution Methods:**
  - Calculus of Variation
  - Dynamic Programming
Optimization Approaches

Examples

- Minimum kinetic energy
  \[ J = \int_0^\tau \dot{q}^2 dt \]
  - Results in a quadratic polynomial as solution

- Minimum Jerk
  \[ J = \int_0^\tau \ddot{q}^2 dt \]
  - Results in a quintic polynomial as solution

- Minimum Torque Change
  \[ J = \int_0^\tau \dot{u}^2 dt \]
  - Results in something that does not have an analytical description
Operational Space Planning

- All joint space planning methods can also be used in operational space.
- Inverse kinematics is needed to convert operational space trajectories into joint space.
- The resulting joint space motion is usually quite complex.
- Geometric problems can arise:
  - Intermediate points are unreachable.
  - High joint space motion near singular postures.
  - Start and goal reachable in different solutions.
Examples of Geometric Problems
Pattern Generators for Desired Trajectories

- Use Pattern Generators to Create Kinematic Trajectory Plans
  - Use open parameters in pattern generator to generate different movement durations and target settings
Pattern Generators for Trajectory Planning

- What is a pattern generator?
  - A dynamical system (differential equation) with a particular behavior
    - E.g.: Reaching movement can be interpreted as a point attractive behavior:
      \[
      \dot{q}_d = \alpha (q_f - q_d)
      \]

- What is the advantage of a pattern generator?
  - Independent of initial conditions
  - Online planning
  - Online modification through additional “coupling” terms. i.e., planning can react to sensory input
    \[
    \dot{q}_d = \alpha (q_f - q_d) + \beta (q_d - q)
    \]
Pattern Generators for Trajectory Planning

- Disadvantages of Pattern Generators
  - Analysis of behavior is non trivial
  - Need to integrate the equation of motion of the pattern generator at sufficiently high frequency
  - Exact shape of desired trajectories that are generated by the pattern generator are not easy to predict if external coupling is added
  - Modeling of with pattern generators usually requires the manipulation of nonlinear dynamical equations, which is non trivial again
Shaping Attractor Landscapes

Second order dynamics:

\[ \dot{z} = \alpha \dot{z} \left( \beta \left( g - y \right) - z \right) \]

\[ \dot{y} = z \]
Can one create more complex dynamics by nonlinearly modifying the simple second order system?

\[
\begin{align*}
\dot{z} &= \alpha_z (\beta_z (g - y) - z) \\
\dot{y} &= f(?) + z
\end{align*}
\]
A globally stable learnable nonlinear point attractor:

\[
\begin{align*}
\dot{z} &= \alpha_z (\beta_z (g - y) - z) \\
\dot{y} &= \alpha_y (f(x,v) + z)
\end{align*}
\]

where

\[
\dot{v} = \alpha_v (\beta_v (g - x) - v)
\]

\[
\dot{x} = \alpha_x v
\]

Local Linear Model Approx.

\[
f(x,v) = \frac{\sum_{i=1}^{k} w_i b_i v}{\sum_{i=1}^{k} w_i}
\]

\[
w_i = \exp \left( -\frac{1}{2} d_i (x - c_i)^2 \right) \text{ and } x = \frac{x - x_0}{g - x_0}
\]
Example: A Trajectory with Movement Reversal

![Graphs showing trajectory and movement reversal](image-url)
Example: A Minimum Jerk Trajectory
Learning The Attractor from Demonstration

- Given a demonstrated trajectory $y(t)_\text{demo}$ and a goal $g$
  - Extract movement duration
  - Adjust time constants of canonical dynamics to movement duration
  - Use LWL to learn supervised problem

$$\dot{y}_\text{target} = \frac{\dot{y}_\text{demo}}{\alpha_y} - z = f(x,v)$$

- Usually 1-5 learning epochs suffice to get good approximation
Imitation Learning of a Tennis Forehand

Note: All 30 joint space trajectories are fitted independently.
Imitation Learning of a Tennis Backhand
Limit Cycle Dynamics for Rhythmic Movement

- A globally stable learnable limit cycle:

\[
\begin{align*}
\dot{z} &= \alpha_z \left( \beta_z \left( g - y_m \right) - z \right) \\
\dot{y} &= \alpha_y \left( f(r, \phi) + z \right)
\end{align*}
\]

where

- Trajectory Plan Dynamics

\[
\begin{align*}
\dot{r} &= \alpha_r \left( A - r \right) \\
\dot{\phi} &= \omega
\end{align*}
\]

- Canonical System

\[
\begin{align*}
f(x, v) &= \frac{\sum_{i=1}^{k} w_i b_i^T x}{\sum_{i=1}^{k} w_i} \quad \text{where} \quad x = \begin{bmatrix} r \cos \phi \\ r \sin \phi \end{bmatrix} \\
w_i &= \exp \left( d_i \left( \cos(\phi - c_i) - 1 \right) \right)
\end{align*}
\]

- Local Linear Models using van Mises bases
Example: A Complex Rhythmic Trajectory
Imitation Learning of a Drumming Motion
Imitation Learning of a Figure-8 Motion
Pattern Generators for Rhythmic and Discrete Movement

Discrete Movement

\[
\begin{align*}
\Delta v_1 &= \left[t_1 - p_{1r}\right] \\
\dot{v}_1 &= a_r(-v_1 + \Delta v_1) \\
\hat{v}_1 &= a_r(-v_1 + \Delta v_1) \\
\dot{x}_1 &= -a_1 x_1 + (v_1 - x_1) c_r + C_{1r} \\
\dot{y}_1 &= -a_1 y_1 + (x_1 - y_1) c_r \\
\dot{\hat{z}}_1 &= a_r(-r_1 + (1-r_1) b v_1) \\
\ddot{p}_{1r} &= a_r c_r (z_1 - z_2)
\end{align*}
\]

Rhythmic Movement

\[
\begin{align*}
\Delta \omega_2 &= \left[A - (p_2 - p_{2r})\right] \\
\dot{\xi}_2 &= a_r(\xi_2 + \Delta \omega_2) \\
\dot{\psi}_1 &= -a_r \psi_1 + \left(\xi_2 - \psi_1 - b \xi_2 - w [\psi_1] + C_{1r}\right) c_2 \\
\dot{\xi}_1 &= \frac{1}{5} \left(-a_2 \xi_1 + (\psi_1) - \xi_1\right) c_2 \\
\dot{p}_1 &= c_2 (\psi_1 - \psi_2) \\
\dot{p}_2 &= c_2 (\psi_2) \\
\theta &= p_1 = -p_2 \\
\dot{\theta} &= \dot{p}_1 = -\dot{p}_2
\end{align*}
\]
Example from the Discrete Pattern Generator
Discrete Movements at Different Speeds
Example from the Rhythmic Pattern Generator

![Graph showing rhythmic patterns](image-url)