AI & Robotics: Lab Course

Motion Generation

Marc Toussaint
Technical University of Berlin
Summer 2020

Outline

- Motion generation as optimization
- Designing features
- Advanced: equalities & inequalities, path optimization
- Glimpse on available features

Basic Control as Optimization

- Notation

$q \in \mathbb{R}^n$	vector of joint angles (robot configuration)
$\dot q \in \mathbb{R}^n$	vector of joint angular velocities
$\phi :\ q \mapsto y \in \mathbb{R}^d$	**feature** (or fwd kinematic) e.g. position $\in \mathbb{R}^3$ or vector $\in \mathbb{R}^3$
$J(q) = \frac{\partial \phi}{\partial q} \in \mathbb{R}^{d \times n}$	Jacobian
$\Vert v \Vert^2_W = v^\top W v$	squared norm of v w.r.t. metric W

Basic Control as Optimization

- Notation

$q \in \mathbb{R}^n$	vector of joint angles (robot configuration)
$\dot q \in \mathbb{R}^n$	vector of joint angular velocities
$\phi : \ q \mapsto y \in \mathbb{R}^d$	**feature** (or fwd kinematic)
	e.g. position $\in \mathbb{R}^3$ or vector $\in \mathbb{R}^3$
$J(q) = \frac{\partial \phi}{\partial q} \in \mathbb{R}^{d \times n}$	Jacobian
$\|v\|^2_W = v^\top W v$	squared norm of v w.r.t. metric W

- Inverse Kinematics:

$$y^* \mapsto q^* = \underset{q}{\operatorname{argmin}} \|\phi(q) - y^*\|^2_C + \|q - q_0\|^2_W$$

(Solution with linearization at q_0: $q^* = q_0 + J^\sharp(y^* - y_0)$ with $J^\sharp = W^{\text{-}1} J^\top (J W^{\text{-}1} J^\top + C^{\text{-}1})^{\text{-}1}$)

Basic Control as Optimization

- Notation

$q \in \mathbb{R}^n$	vector of joint angles (robot configuration)
$\dot q \in \mathbb{R}^n$	vector of joint angular velocities
$\phi:\ q \mapsto y \in \mathbb{R}^d$	**feature** (or fwd kinematic) e.g. position $\in \mathbb{R}^3$ or vector $\in \mathbb{R}^3$
$J(q) = \frac{\partial \phi}{\partial q} \in \mathbb{R}^{d \times n}$	Jacobian
$\|v\|^2_W = v^\top W v$	squared norm of v w.r.t. metric W

- Inverse Kinematics:

$$y^* \mapsto q^* = \operatorname*{argmin}_q \|\phi(q) - y^*\|^2_C + \|q - q_0\|^2_W$$

(Solution with linearization at q_0: $q^* = q_0 + J^\sharp (y^* - y_0)$ with $J^\sharp = W^{\text{-}1} J^\top (J W^{\text{-}1} J^\top + C^{\text{-}1})^{\text{-}1}$)

- Operational Space Control:

$$\ddot y^* \mapsto u^* = \operatorname*{argmin}_u \|\ddot\phi(q) - \ddot y^*\|^2_C + \|u\|^2_H$$

It's all about features

- Let's rewrite IK a bit:

$$q^* = \underset{q}{\operatorname{argmin}} \, \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2$$

  - q_0 is the current state
  - We want to compute q_1, the next state
  - Let $\Phi_1 = \sqrt{C}(\phi(q) - y^*)$, and $\Phi_2 = \sqrt{W}(q - q_0)$
  - And let $\Phi = (\Phi_1; \Phi_2; ...)$, stacking all features in a single vector

- Then IK is a **sum-of-squares problem**

$$q^* = \underset{q}{\operatorname{argmin}} \, \Phi^\top \Phi$$

$\to$ To design motion
  - think of all kinds of features you want to penalize,
  - zero calibrate them (subtract the target),
  - scale them (multiply with some $\sqrt{C}$),
  - stack them into a big feature vector,
  - call an efficient SOS optimization method.

Hard constraints: beyond just penalizing

- We can not only solve SOS problems, but also

$$\min_q \sum_{k \in S} \phi_k(q)^\top \phi_k(q) \quad \text{s.t.} \quad \forall_{k \in I} \, \phi_k(q) \le 0, \ \ \forall_{k \in E} \, \phi_k(q) = 0 \ ,$$

  where
  - some features $\phi_k, k \in S$, are SOS
  - some features $\phi_k, k \in I$, impose inequalities
  - some features $\phi_k, k \in E$, impose equalities

$\to$ To design motion
  - define features as above
  - but also specify the type of each feature: if sos, eq, or ineq

Generalizing this to dynamics

- In Operational Space Control we solve for acceleration/torques of the robot. But we discretize time anyway. So we can also just optimize the next configuration:
- Let q_0 be the current configuration, q_{-1} be the *previous* configuration, we want to solve for the next configuration q_1 subject to costs that depend on the acceleration $(q_1 + q_{-1} - 2q_0)/\tau^2$

- Optimizing for q_1 falls exactly into the same category of optimization problems $\rightarrow$ we can now do hard-constrained operational space control. One of the hard constraints can be about the dynamics constraints

k-order Features

- In IK we had to define a feature $\Phi_2 = \sqrt{W}(q_1 - q_0)$ to penalize motion/velocity – this is a feature over 2 configurations (q_0, q_1)
- In Operational Space Control we have to define a feature that depends on the acceleration $(q_1 + q_{-1} - 2q_0)/\tau^2$ – this is a feature over 3 configurations (q_{-1}, q_0, q_1)
- In general we can have k**-order features**, which depend on $k - 1$ consecutive configurations and typically compute – in some feature space – finite difference velocities, accelerations, jerks, etc.

$\rightarrow$ To design motion

  - define features
  - specify the type of each feature (sos, eq, or ineq)
  - but also specify the k-order of each feature (i.e., if the feature is meant to be a finite difference velocity over 2 configurations, or the finite difference acceleration over 3 configurations)

Finally, Path Optimization

- All the above generalizes to not only solve for the next configuration q_1, but also a whole sequence of future configurations $q_1, .., q_T$.

$$\min_{x_1,..,x_n} \sum_{k\in S} \phi_k(x_{\pi_k})^\top \phi_k(x_{\pi_k}) \quad \text{s.t.} \quad \underset{k\in I}{\forall}\ \phi_k(x_{\pi_k}) \le 0, \quad \underset{k\in E}{\forall}\ \phi_k(x_{\pi_k}) = 0 \ , \tag{1}$$

  - Each feature computes something for (at most $k-1$) consecutive configurations π_k
  - Each feature ϕ_k penalizes some aspect of the path locally in time

$\to$ To design motion

  - define features
  - specify their type (sos, ineq, eq)
  - specify their order (velocity?, acceleration?)
  - specify at which time $t \in \{1, .., T\}$ in the path they apply

Predefined features in KOMO

- Symbols for pre-defined features

  position, positionDiff, positionRel,
  quaternion, quaternionDiff, quaternionRel,
  pose, poseDiff, poseRel, [avoid these]
  vectorX, vectorXDiff, vectorXRel,
  vectorY, vectorYDiff, vectorYRel,
  vectorZ, vectorZDiff, vectorZRel,
  scalarProductXX, scalarProductXY, scalarProductXZ, scalarProductYX, scalarProductYY, scalarProductYZ, scalarProductZZ,
  gazeAt, angularVel,
  accumulatedCollisions, jointLimits, **distance**, oppose,
  qItself,
  aboveBox, insideBox,
  standingAbove,
  physics, contactConstraints, energy,
  transAccelerations, transVelocities,
  qQuaternionNorms,

- Full objective specification

```
addObjective(times, featureSymbol, frameNames, objectiveType, scale, target, order)
```

(There are many more features defined in the code, but not interfaced with a symbol.)