Task Spaces

M Toussaint

October 9, 2015

1 Purpose

Task spaces are defined by a mapping $\phi : q \to y$ from the joint state $q \in \mathbb{R}^n$ to a task space $y \in \mathbb{R}^m$. They are central in designing motion and manipulation, both, in the context of trajectory optimization as well as in specifying position/force/impedance controllers:

For **trajectory optimization**, cost functions are defined by costs or constraints in task spaces. Given a single task space ϕ, we may define

– costs $\|\phi(q)\|^2$,

– an inequality constraint $\phi(q) \le 0$ (element-wise inequality),

– an equality constraint $\phi(q) = 0$.

All three assume that the 'target' is at zero. For convenience, the code allows to specify an additional linear transform $\tilde\phi(q) \leftarrow \rho(\phi(q) - y_{\text{ref}})$, defined by a target reference y_{ref} and a scaling ρ. In KOMO, costs and constraints can also be defined on $k+1$-tuples of consecutive states in task space, allowing to have cost and constraints on task space velocities or accelerations. Trajectory optimization problems are defined by many such costs/constraints in various task spaces at various time steps.

For simple **feedback control**, in each task space we may have

– a desired linear acceleration behavior in the task space

– a desired force or force constraint (upper bound) in the task space

– a desired impedance around a reference

All of these can be fused to a joint-level force-feedback controller (details, see controller docu). On the higher level, the control mode is specified by defining multiple task spaces and the desired behaviors in these. (The activity of such tasks (on the symbolic level) is controlled by the RelationalMachine, see its docu.)

In both cases, defining task spaces is the core.

2 Task Spaces

The notation is as in the robotics lecture slides. In particular, $T_{W\to i}$ is the 4-by-4 homogeneous transform from world frame to the frame of shape i; and $R_{W\to i}$ is its rotation matrix only.

2.1 Position

2.1.1 absolute

parameters: shape index i, point offset v

$\phi^{\text{pos}}_{iv}(q) = T_{W\to i}\ v$
$J^{\text{pos}}_{iv}(q)_{\cdot k} = [k \prec i]\ a_k \times (\phi^{\text{pos}}_{iv}(q) - p_k)$

– a_k, p_k are axis and position of joint k
– $[k \prec i]$ indicates whether joint k is between root and shape i
– $J_{\cdot k}$ is the kth column of J

2.1.2 difference

parameters: shape indices i, j, point offset v in i and w in j
$\phi^{\text{pos}}_{iv-jw}(q) = \phi^{\text{pos}}_{iv} - \phi^{\text{pos}}_{jw}$ $\phi^{\text{pos}}_{iv-jw}(q) = J^{\text{pos}}_{iv} - J^{\text{pos}}_{jw}$

2.1.3 relative

parameters: shape indices i, j, point offset v in i and w in j
$\phi^{\text{pos}}_{iv|jw}(q) = R^{-1}_j(\phi^{\text{pos}}_{iv} - \phi^{\text{pos}}_{jw})$
$J^{\text{pos}}_{iv|jw}(q) = R^{-1}_j[J^{\text{pos}}_{iv} - J^{\text{pos}}_{jw} - A_j \times (\phi^{\text{pos}}_{iv} - \phi^{\text{pos}}_{jw})]$
Derivation: For $y = Rp$ the derivative w.r.t. a rotation around axis a is $y' = Rp' + R'p = Rp' + a \times Rp$. For $y = R^{-1}p$ the derivative is $y' = R^{-1}p' - R^{-1}(R')R^{-1}p = R^{-1}(p' - a \times p)$. (For details see `http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/3d-geometry.pdf`)

2.2 Vector

2.2.1 absolute

parameters: shape index i, attached vector v
$\phi^{\text{vec}}_{iv}(q) = R_{W\to i}\ v$
$J^{\text{vec}}_{iv}(q) = A_i \times \phi^{\text{vec}}_{iv}(q)$

– A_i is a matrix with columns $(A_i)_{\cdot k} = [k \prec i]\ a_k$ containing the joint axes or zeros
– the short notation "$A \times p$" means that each *column* in A takes the cross-product with p.

2.2.2 difference

parameters: shape indices i, j, attached vectors v in i and w in j
$\phi^{\text{vec}}_{iv-jw}(q) = \phi^{\text{vec}}_{iv} - \phi^{\text{vec}}_{jw}$
$J^{\text{vec}}_{iv-jw}(q) = J^{\text{vec}}_{iv} - J^{\text{vec}}_{jw}$

2.2.3 relative

parameters: shape indices i, j, attached vector v in i
$\phi^{\text{vec}}_{iv|j}(q) = R^{-1}_j \phi^{\text{vec}}_{iv}$
$J^{\text{vec}}_{iv|j}(q) = R^{-1}_j[J^{\text{vec}}_{iv} - A_j \times \phi^{\text{vec}}_{iv}]$

2.3 Quaternion

2.3.1 absolute

parameters: shape index i
$\phi^{\text{quat}}_{i}(q) = \text{quaternion}(R_{W\to i})$
$J^{\text{quat}}_{i}(q) = \text{special}$

2.3.2 difference

parameters: shape indices i, j

$\phi^{\text{quat}}_{i-j}(q) = \phi^{\text{quat}}_i - \phi^{\text{quat}}_j$

$J^{\text{quat}}_{i-j}(q) = J^{\text{quat}}_i - J^{\text{quat}}_j$

2.3.3 relative

parameters: shape indices i, j

$\phi^{\text{quat}}_{i|j}(q) = \phi^{\text{quat-1}}_j \circ \phi^{\text{quat}}_i$

$J^{\text{quat}}_{i-j}(q) =$ not implemented

2.4 Alignment

parameters: shape indices i, j, attached vectors v, w

$\phi^{\text{align}}_{iv|jw}(q) = (\phi^{\text{vec}}_{jw})^\top \phi^{\text{vec}}_{iv}$

$J^{\text{align}}_{iv|jw}(q) = (\phi^{\text{vec}}_{jw})^\top J^{\text{vec}}_{iv} + \phi^{\text{vec}\top}_{iv} J^{\text{vec}}_{jw}$

Note: $\phi^{\text{align}} = 1 \leftrightarrow$ align $\phi^{\text{align}} = -1 \leftrightarrow$ anti-align $\phi^{\text{align}} = 0 \leftrightarrow$ orthog.

2.5 Gaze

2D orthogonality measure of object relative to camera plane

parameters: eye index i with offset v; target index j with offset w

$$\phi^{\text{gaze}}_{iv,jw}(q) = \begin{pmatrix} \phi^{\text{vec}\ \top}_{i,e_x}(\phi^{\text{pos}}_{jw} - \phi^{\text{pos}}_{iv}) \\ \phi^{\text{vec}\ \top}_{i,e_y}(\phi^{\text{pos}}_{jw} - \phi^{\text{pos}}_{iv}) \end{pmatrix} \in \mathbb{R}^2 \tag{1}$$

Here $e_x = (1, 0, 0)$ and $e_y = (0, 1, 0)$ are the camera plane axes.

Jacobians straight-forward

2.6 qItself

$\phi^{\text{qItself}}_{iv,jw}(q) = q$

2.7 Joint limits measure

parameters: joint limits $q_{\text{low}}, q_{\text{hi}}$, margin m

$\phi_{\text{limits}}(q) = \frac{1}{m}\sum_{i=1}^n [m - q_i + q_{\text{low}}]^+ + [m + q_i - q_{\text{hi}}]^+$

$J_{\text{limits}}(q)_{1,i} = -\frac{1}{m}[m - q_i + q_{\text{low}} > 0] + \frac{1}{m}[m + q_i - q_{\text{hi}} > 0]$

$[x]^+ = x > 0 ? x : 0$ $[\cdots]$: indicator function

2.8 Collision limits measure

parameters: margin m

$\phi_{\text{col}}(q) = \frac{1}{m}\sum_{k=1}^K [m - |p^a_k - p^b_k|]^+$

$J_{\text{col}}(q) = \frac{1}{m}\sum_{k=1}^K [m - |p^a_k - p^b_k| > 0](-J^{\text{pos}}_{p^a_k} + J^{\text{pos}}_{p^b_k})^\top \frac{p^a_k - p^b_k}{|p^a_k - p^b_k|}$

A collision detection engine returns a set $\{(a, b, p^a, p^b)_{k=1}^K\}$ of potential collisions between shape a_k and b_k, with nearest points p^a_k on a and p^b_k on b.

2.9 Shape distance measures (using SWIFT)

- allPTMT, //phi=sum over all proxies (as is standard)
- listedVsListedPTMT, //phi=sum over all proxies between listed shapes
- allVsListedPTMT, //phi=sum over all proxies against listed shapes
- allExceptListedPTMT, //as above, but excluding listed shapes
- bipartitePTMT, //sum over proxies between the two sets of shapes (shapes, shapes2)
- pairsPTMT, //sum over proxies of explicitly listed pairs (shapes is n-times-2)
- allExceptPairsPTMT, //sum excluding these pairs
- vectorPTMT //vector of all pair proxies (this is the only case where dim(phi)¿1)

2.10 GJK pairwise shape distance (including negative)

2.11 Plane distance

3 Application

Just get a glimpse on how task space definitions are used to script motions, here is a script of a little PR2 dance. (The 'logic' below the script implements kind of macros – that's part of the RAP.) (`wheels` is the same as qItself, but refers only to the 3 base coordinates)

```
cleanAll #this only declares a novel symbol...

Script {
  (FollowReferenceActivity wheels){ type=wheels, target=[0, .3, .2], PD=[.5, .9, .5, 10.]}
  (MyTask endeffR){ type=pos, ref2=base_footprint, target=[.2, -.5, 1.3], PD=[.5, .9, .5, 10.]}
  (MyTask endeffL){ type=pos, ref2=base_footprint, target=[.2, +.5, 1.3], PD=[.5, .9, .5, 10.]}
  { (conv FollowReferenceActivity wheels)  (conv MyTask endeffR) }  #this waits for convergence of activities
  (cleanAll) #this switches off the current activities
  (cleanAll)! #this switches off the switching-off

  (FollowReferenceActivity wheels){ type=wheels, target=[0, -.3, -.2], PD=[.5, .9, .5, 10.]}
  (MyTask endeffR){ type=pos, ref2=base_footprint, target=[.7, -.2, .7], PD=[.5, .9, .5, 10.]}
  (MyTask endeffL){ type=pos, ref2=base_footprint, target=[.7, +.2, .7], PD=[.5, .9, .5, 10.]}
  { (conv FollowReferenceActivity wheels)  (conv MyTask endeffL) }
  (cleanAll)
  (cleanAll)!

  (FollowReferenceActivity wheels){ type=wheels, target=[0, .3, .2], PD=[.5, .9, .5, 10.]}
  (MyTask endeffR){ type=pos, ref2=base_footprint, target=[.2, -.5, 1.3], PD=[.5, .9, .5, 10.]}
  (MyTask endeffL){ type=pos, ref2=base_footprint, target=[.2, +.5, 1.3], PD=[.5, .9, .5, 10.]}
  { (conv MyTask endeffL) }
  (cleanAll)
  (cleanAll)!

  (FollowReferenceActivity wheels){ type=wheels, target=[0, 0, 0], PD=[.5, .9, .5, 10.]}
  (HomingActivity)
  { (conv HomingActivity) (conv FollowReferenceActivity wheels) }
}


Rule {
    X, Y,
    { (cleanAll) (conv X Y) }
    { (conv X Y)! }
}

Rule {
    X,
    { (cleanAll) (MyTask X) }
    { (MyTask X)! }
}

Rule {
    X,
    { (cleanAll) (FollowReferenceActivity X) }
    { (FollowReferenceActivity X)! }
}
```