# 優先度付き逆運動学による動作生成
# リファレンスマニュアル

平成 34 年 2 月 9 日

平岡直樹
hiraoka@jsk.t.u-tokyo.ac.jp

## 目 次

# 1　優先度付き逆運動学の基礎

## 1.1　優先度付き逆運動学について

hogehoge

## 1.2　章の構成

第 2 章では，クラスを説明する．

# 2　優先度付き逆運動学

**prioritized-inverse-kinematics** *tasks variables &key (regular 1.000000e-06) (regular-rel 5.0) (regular-max 0.1) (task0 nil) (stop 50) (min-loop 0) (revert-if-fail t) (debug-view :no-message) (qp-solver #'solve-osqp-qp) (qp-args (list :solve-qp-mode :sqp-with-hotstart)) (viewer (send *irtviewer*:viewer))* [function]

- tasks : (list (list task1-1 task1-2 ... )
  (list task2-1 task2-2 ... )
  (list task3-1 task3-2 ... )
  ...)
  各 *task* は *inverse-kinematics-task* クラスである．*prioritized-inverse-kinematics* は次の問題を解く．priority 1 のタスクが最も優先度が高い．

$$
\begin{aligned}
\text{priority1} \quad &: \quad \min_{\boldsymbol{x}} \ \text{task1-1} + \text{task1-2} + ... + \boldsymbol{x}^{\mathrm{T}}\mathbf{W}\boldsymbol{x} \\
\text{priority2} \quad &: \quad \min_{\boldsymbol{x}} \ \text{task2-1} + \text{task2-2} + ... + \boldsymbol{x}^{\mathrm{T}}\mathbf{W}\boldsymbol{x} \\
\text{priority3} \quad &: \quad \min_{\boldsymbol{x}} \ \text{task3-1} + \text{task3-2} + ... + \boldsymbol{x}^{\mathrm{T}}\mathbf{W}\boldsymbol{x} \\
&\vdots
\end{aligned}
$$

- variables : (list variable1 variable2 ...)
  探索変数のリストである．各 *variable* は *joint* クラスまたは *inverse-kinematics-variable* クラスである．*joint* クラスが与えられた場合，その *joint-angle* が探索変数となる．同一の探索変数を意味する *variable* を *2* 回以上与えてはならない．
- *regular, regular-rel, regular-max : weight or (list weight1 weight2 weight3 ...)*
  各 priority の QP において解を一意に定めるための正則化重みの大きさである．各 *weight* は *float* クラスである．*regular* として *float* が与えられた場合は全 priority 共通でその重みを用い，*list* が与えられた場合は各 priority ごとに指定された重みを用いる．$w$ = regular，$w_r$ = regular-rel，$w_{max}$ = regular-max とし，各 priority の現在のエラーの 2 乗和を $e$ とおくと，重みは $\min(w + w_r e, w_{max})$ となる．

- task0 : (list task0-1 task0-2 ... )
  各 *task* は *inverse-kinematics-task* クラスである．*task0* が与えられた場合，*prioritized-inverse-kinematics* は次の問題を解く．

$$\begin{aligned} \text{priority0} &: \min\ \text{task0-1} + \text{task0-2} + ... \\ \text{priority1} &: \min\ \text{task1-1} + \text{task1-2} + ... \\ \text{priority2} &: \min\ \text{task2-1} + \text{task2-2} + ... \\ \text{priority3} &: \min\ \text{task3-1} + \text{task3-2} + ... \\ &\vdots \end{aligned}$$

  ただし，priority 0 の各タスクの最適値を同時に満たす解が必ず存在すると仮定し，priority 0 では QP を解かないことで高速化を図る．したがって，*task0* としてこの仮定が成り立たないタスクを与えるべきでない．

- *stop*
  *stop* 回の反復計算後，直ちに終了する．*stop* は *interger* クラスである．

- *min-loop*
  *min-loop* 回の反復計算後から，各反復終了時に全 *task* の終了条件を満たすなら直ちに終了する．*min-loop* は *interger* クラスである．

- *revert-if-fail : t or nil*
  *t* ならば，終了時にある task の終了条件を満たして無い場合に，初期状態に戻してから *prioritized-inverse-kinematics* から返る

- *debug-view*
  *t* ならば反復計算中に状態を描画しデバッグメッセージを表示する．*:no-message* ならば反復計算中に状態を描画する．*nil* ならば何もしない．

- *qp-solver*
  *QP* のソルバを指定する．*qp-solver* は *function* クラスである．

- *qp-args*
  *prioritized-inverse-kinematics* 内で，*(apply qp-solver ... qp-args)* の形で *qp-solver* が呼ばれる．

## inverse-kinematics-task [class]

:super **propertied-object**
:slots (a $\mathbf{A}$)
(b $\boldsymbol{b}$)
(c $\mathbf{C}$)
(dl $\mathbf{d}_l$)
(du $\mathbf{d}_u$)
(wa $\mathbf{W}_A$)
(wc $\mathbf{W}_C$)
(asparce $\mathbf{A}_{sparce}$, used for sparce matrix calculation)
(csparce $\mathbf{C}_{sparce}$, used for sparce matrix calculation)
(equality-rows size of row of $\mathbf{A}$)
(inequality-rows size of row of $\mathbf{C}$)
(cols size of column of $\mathbf{A}$ (constant))

各タスクを表すクラス.
*prioritized-inverse-kinematics* 中で次の問題に変換される.

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{\omega}} \quad & (\mathbf{A}\boldsymbol{x}-\boldsymbol{b})^{\mathrm{T}}\mathbf{W}_A(\mathbf{A}\boldsymbol{x}-\boldsymbol{b}) + \boldsymbol{\omega}^{\mathrm{T}}\mathbf{W}_C\boldsymbol{\omega} \\ \text{subject to} \quad & \boldsymbol{d}_l \leqq \mathbf{C}\boldsymbol{x}+\boldsymbol{\omega} \leqq \boldsymbol{d}_u \end{aligned}$$

**:init** [method]
Initialize instance

**:initialize** *variable-length variables* [method]
called at the start of prioritized-inverse-kinematics

**:update** [method]
called at the start of each iteration.
update A, b , Wa, Asparce, equality-rows, C, dl, du , Wc, Csparce, inequality-rows

**:is-satisfied** *&optional force-check* [method]
終了判定に用いる.

**:draw** *&key ((:viewer vwer) ∗viewer∗)* [method]
debug view

**:debug** [method]
debug message

**:a** [method]
return $\mathbf{A}$

**:b** [method]
return $\boldsymbol{b}$

**:wa** [method]
return $\mathbf{W}_A$

**:asparce** [method]
return $\mathbf{A}_{sparce}$

**:c** [method]
return $\mathbf{C}$

**:dl** [method]
return $\boldsymbol{d}_l$

**:du** [method]
return $\boldsymbol{d}_u$

**:wc** [method]
return $\mathbf{W}_C$

**:csparce** [method]
return $\mathbf{C}_{sparce}$

**:additional-variables** [method]

return list of additional variable

**:additional-task0s** [method]

return list of additional task0

## minmax-angle-task [class]

:super **inverse-kinematics-task**
:slots (j , target joint)
(max-angle)
(min-angle)
(target-variable , inverse-kinematics-variable corresponding to target joint)
(check 終了判定を行うか否か)
(check-margin 終了判定時のマージン)

*joint* の角度上下限制約を表現するクラス.
*joint* の角度上下限を $\boldsymbol{\theta}_{max}$，$\boldsymbol{\theta}_{min}$，現在の角度を $\boldsymbol{\theta}$ とすると，*prioritized-inverse-kinematics* 中で次の問題に変換される.

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{\omega}} \quad & \boldsymbol{\omega}^{\mathrm{T}} W \boldsymbol{\omega} \\ \text{subject to} \quad & \boldsymbol{\theta}_{min} \leqq \boldsymbol{\theta} + \boldsymbol{x} + \boldsymbol{\omega} \leqq \boldsymbol{\theta}_{max} \end{aligned} \tag{2.1}$$

$\boldsymbol{\theta}$ の単位は m，rad である.

**:init** *_joint &key ((:w _w) 1.0)* [method]
*((:max-angle _max-angle))*
*((:min-angle _min-angle))*
*((:check _check) t)*
*((:check-margin _check-margin))*

- joint
  *joint* クラス. 上下限を考える対象の関節である．仮想関節の場合は，子の *cascaded-link* クラス，それがなければ *bodyset-link* クラスを与える.
- *W*
  *float* クラスの場合 **W** はその値を対角成分に並べた行列になる
  *vector* クラスの場合 **W** は各値を対角成分に並べた行列になる
  *matrix* クラスの場合 **W** としてそのまま使用される.
- *max-angle, min-angle*
  *float* または *float-vector* クラス．関節上下限を表す．単位は mm，deg．デフォルトは (send joint :max-angle) (send joint :min-angle).
- check : t or nil
  終了判定を行うか否か
- check-margin
  *float* クラス．終了判定時にこの値以下の侵犯を許容する．デフォルトは *1e-5* だが，*sphere* 関節は *1e-3*．単位は m,rad

**:initialize** *variable-length variables* [method]

探索変数 *variables* のうち，該当 *joint* に影響を与える成分を調べ，記録する．*0* または *1* 個の *variables* が発見されなければならない.

**:update** [method]

現在の関節角度に応じて $\boldsymbol{d}_l$，$\boldsymbol{d}_u$ を更新する

**:is-satisfied** *&optional force-check* [method]

関節角度上下限を満足しているかどうかを判定

**:debug** [method]

現在の関節角度と上下限を表示

## joint-velocity-task [class]

:super **inverse-kinematics-task**
:slots (j , target joint)
(max-vel $\dot{\boldsymbol{\theta}}_{max}$[m/s][rad/s])
(min-vel $\dot{\boldsymbol{\theta}}_{min}$[m/s][rad/s])
(periodic-time $\Delta t$[s], 一回の反復計算の周期)
(target-variable , inverse-kinematics-variable corresponding to target joint)

*joint* の角速度上下限制約を表現するクラス．
*joint* の角速度上下限を $\dot{\boldsymbol{\theta}}_{max}$，$\dot{\boldsymbol{\theta}}_{min}$，一回の反復計算の周期を $\Delta t$ とすると，*prioritized-inverse-kinematics* 中で次の問題に変換される．

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{\omega}} \quad & \boldsymbol{\omega}^{\mathrm{T}} W \boldsymbol{\omega} \\ \text{subject to} \quad & \dot{\boldsymbol{\theta}}_{min} * \Delta t \leqq \boldsymbol{x} + \boldsymbol{\omega} \leqq \dot{\boldsymbol{\theta}}_{max} * \Delta t \end{aligned} \tag{2.2}$$

$\dot{\boldsymbol{\theta}}$ の単位は m/s，rad/s である．

**:init** *_joint &key ((:w _w) 1.0)* [method]
*((:max-vel _max-vel))*
*((:min-vel _min-vel))*
*((:periodic-time _periodic-time) 0.05)*

- joint
  *joint* クラス. 上下限を考える対象の関節である．仮想関節の場合は，子の *cascaded-link* クラス，それがなければ *bodyset-link* クラスを与える．
- *W*
  *float* クラスの場合 **W** はその値を対角成分に並べた行列になる
  *vector* クラスの場合 **W** は各値を対角成分に並べた行列になる
  *matrix* クラスの場合 **W** としてそのまま使用される．
- *max-vel, min-vel*
  *float* または *float-vector* クラス．関節速度上下限を表す．単位は m/s，rad/s．デフォルトは (send joint :max-joint-velocity).
- periodic-time
  *float* クラス一回の反復計算の周期.

**:initialize** *variable-length variables* [method]

探索変数 *variables* のうち，該当 *joint* に影響を与える成分を調べ，記録する．*0* または *1* 個の *variables* が発見されなければならない．

**:update** [method]

現在の関節角度に応じて $\boldsymbol{d}_l$，$\boldsymbol{d}_u$ を更新する

**:debug** [method]

現在の速度上下限を表示

## joint-angle-task [class]

:super **inverse-kinematics-task**

:slots (j1 , target joint 1)
(j2 , target joint 2)
(target-variable1 , inverse-kinematics-variable corresponding to target joint 1)
(target-variable2 , inverse-kinematics-variable corresponding to target joint 2)
(check 終了判定を行うか否か)
(check-margin 終了判定時のマージン)

2 つの関節角度を一致させる制約を表現するクラス．
2 つの関節角度の誤差 (joint2 - joint1) を $\boldsymbol{e}$ とおき，そのヤコビアンを $\mathbf{J}$ とすると，*prioritized-inverse-kinematics* 中で次の問題に変換される．

$$\min_{\boldsymbol{x}} \quad (\mathbf{J}\boldsymbol{x}+\boldsymbol{e})^{\mathrm{T}}\mathbf{W}(\mathbf{J}\boldsymbol{x}+\boldsymbol{e})$$

$\boldsymbol{e}$ の単位は m，rad である．

**:init** *_joint1 _joint2 &key ((:w _w) 1.0) ((:check _check) t) ((:check-margin _check-margin))* [method]

- joint1, joint2
  *joint* クラスまたは *float,float-vector* クラス (mm，deg). 一致させる 2 つの関節角度を表す．仮想関節の場合は，子の *cascaded-link* クラス，それがなければ *bodyset-link* クラスを与える．
- *W*
  *float* クラスの場合 $\mathbf{W}$ はその値を対角成分に並べた行列になる
  *vector* クラスの場合 $\mathbf{W}$ は各値を対角成分に並べた行列になる
  *matrix* クラスの場合 $\mathbf{W}$ としてそのまま使用される．
- *check : t or nil*
  終了判定を行うか否か
- *check-margin*
  *float* クラス．終了判定時にこの値以下の侵犯を許容する．デフォルトは *1e-5* だが，*sphere* 関節は *1e-3*．単位は m,rad

**:initialize** *variable-length variables* [method]

探索変数 *variables* のうち，該当 *joint* に影響を与える成分を調べ，記録する．それぞれ *0* または *1* 個の *variables* が発見されなければならない．

**:update** [method]

現在の関節角度に応じて $\boldsymbol{d}_l$，$\boldsymbol{d}_u$ を更新する

**:is-satisfied** *&optional force-check* [method]

関節角度が一致しているかどうかを判定

**:debug** [method]

現在の角度誤差を表示

## infeasible-angle-vector-task [class]

:super **inverse-kinematics-task**
:slots (joints , target joints)
(infeasible-angle-vector )
(min-distance)
(targets , list of (joint infeasible-angle inverse-kinematics-variable))
(check 終了判定を行うか否か)

対象の複数関節の関節角度列を関節角度空間のある一点からの距離が閾値以上にさせる制約を表現するクラス.
距離を $d$ とおき，そのヤコビアンを $\mathbf{J}$ とすると，*prioritized-inverse-kinematics* 中で次の問題に変換される.

$$\begin{aligned} \min_{\boldsymbol{x},\omega} \quad & \omega W \omega \\ \text{subject to} \quad & d_{min} \leqq \mathbf{J}\boldsymbol{x} + d + \omega \end{aligned}$$

$d$ の単位は m，rad である.

**:init** *_joints _infeasible-angle-vector &key ((:min-distance _min-distance) 0.1) ((:w _w) 1.0) ((:check _check) t)* [method]

- joints
  *joint* クラスのリスト. 対象の関節を表す．仮想関節非対応.
- *infeasible-angle-vector*
  *joint-vector* クラス. 回避したい関節角度列を表す．*[deg][mm]*
- *min-distance*
  *float* クラス.*infeasible-angle-vector* からの距離の最小値を表す．*[rad][m]*
- *W*
  *float* クラス
- *check : t or nil*
  終了判定を行うか否か

**:initialize** *variable-length variables* [method]
探索変数 *variables* のうち，該当 *joint* に影響を与える成分を調べ，記録する．それぞれ *0* または *1* 個の *variables* が発見されなければならない.

**:update** [method]
現在の関節角度に応じて $\boldsymbol{d}_l$，$\mathbf{C}$ を更新する

**:is-satisfied** *&optional force-check* [method]
距離が最小値以上かどうかを判定

**:debug** [method]
現在の距離を表示

## avoid-angle-vectors-task [class]

:super **inverse-kinematics-task**
:slots (joints , target joints)
(avoid-angle-vectors )
(min-variable)
(min-variable-task)
(targets , list of list of (joint avoid-angle inverse-kinematics-variable))

対象の複数関節の関節角度列と，関節角度空間の複数点からの距離の最小値を最大化するタスクを表現するクラス.
関節角度空間の点 $i$ からの距離を $d_i$ とおき，そのヤコビアンを $\mathbf{J}_i$ とすると，*prioritized-inverse-kinematics* 中で次の問題に変換される.

$$\begin{aligned} \min_{\boldsymbol{x},d} \quad & -dWd \\ \text{subject to} \quad & 0 \leqq d \\ & d \leqq \mathbf{J}_i\boldsymbol{x} + d_i \end{aligned}$$

$d$ の単位は m，rad である.

**:init** *_joints _avoid-angle-vectors &key ((:w _w) 1.0) (value-scale 1.0)* [method]

- joints
  *joint* クラスのリスト. 対象の関節を表す．仮想関節非対応.
- *infeasible-angle-vector*
  *joint-vector* クラス. 回避したい関節角度列を表す．*[deg][mm]*
- *min-distance*
  *float* クラス.*infeasible-angle-vector* からの距離の最小値を表す．*[rad][m]*
- *W*
  *float* クラス
- *check : t or nil*
  終了判定を行うか否か

**:initialize** *variable-length variables* [method]
探索変数 *variables* のうち，影響を与える成分を調べ，記録する.

**:update** [method]
現在の関節角度に応じて $\boldsymbol{b}$，$\mathbf{A}$ を更新する

**:debug** [method]
現在の距離を表示

**:additional-variables** [method]

**:additional-task0s** [method]

## avoid-angle-vectors-task-task0 [class]

:super **inverse-kinematics-task**
:slots (joints , target joints)

(avoid-angle-vectors )
(min-variable)
(targets , list of list of (joint avoid-angle inverse-kinematics-variable))

avoid-angle-vectors-task クラスの不等式制約を担当するクラス

**:init** *_joints _avoid-angle-vectors _min-variable* [method]

**:initialize** *variable-length variables* [method]
探索変数 *variables* のうち，該当 *joint* に影響を与える成分を調べ，記録する．それぞれ *0* または *1* 個の *variables* が発見されなければならない．

**:update** [method]
現在の関節角度に応じて $\boldsymbol{d}_l$，$\mathbf{C}$ を更新する

**:debug** [method]
現在の距離を表示

## move-target-task [class]

:super **inverse-kinematics-task**
:slots (target-coords)
(move-target)
(translation-axis ,represented in translation-coords)
(translation-coords)
(wtrans $\mathbf{W}_{trans}$, represented in translation-coords)
(rows-trans size of row of $\boldsymbol{e}_{trans}$)
(rotation-axis ,represented in rotation-coords)
(rotation-coords)
(wrot $\mathbf{W}_{rot}$, represented in rotation-coords)
(rows-rot size of row of $\boldsymbol{e}_{rot}$)
(target-coords-variables , target-coords に影響を与える variable のリスト)
(move-target-variables , move-target に影響を与える variable のリスト)
(check 終了判定を行うか否か)
(thre 終了判定時の並進許容誤差)
(rthre 終了判定時の回転許容誤差)
(b-raw , min-max 適用前の $\boldsymbol{b}$)
(p-limit 一回の反復計算で動く並進ノルムの大きさの上限 [m])
(r-limit 一回の反復計算で動く回転ノルムの大きさの上限 [rad])
(tmp-v0)
(tmp-v1)
(tmp-v2)
(tmp-v3)
(tmp-v3a)
(tmp-v3b)
(tmp-m66)
(tmp-m33)

2つの *coordinates* を一致させるタスクを表すクラス．*move-target* と *target-coords* を一致させる．*move-target* と *target-coords* はどちらも動いてよい．
*move-target* と *target-coords* の並進誤差を *translation-coords* の座標系で表現し，*translation-axis* によって抽出された成分を $\boldsymbol{e}_{trans}$ と表す．*move-target* と *target-coords* の回転誤差を *rotation-coords* の座標系で表現し，*rotation-axis* によって抽出された成分を $\boldsymbol{e}_{rot}$ と表す．$\boldsymbol{e}_{trans}$，$\boldsymbol{e}_{rot}$ のヤコビアンをそれぞれ $\mathbf{J}_{trans}$，$\mathbf{J}_{rot}$ とすると，*prioritized-inverse-kinematics* 中で次の問題に変換される．

$$\min_{\boldsymbol{x}} \quad (\mathbf{J}_{trans}\boldsymbol{x}+\boldsymbol{e}_{trans})^{\mathrm{T}}\mathbf{W}_{trans}(\mathbf{J}_{trans}\boldsymbol{x}+\boldsymbol{e}_{trans})+(\mathbf{J}_{rot}\boldsymbol{x}+\boldsymbol{e}_{rot})^{\mathrm{T}}\mathbf{W}_{rot}(\mathbf{J}_{rot}\boldsymbol{x}+\boldsymbol{e}_{rot})$$

$\boldsymbol{e}_{trans}$ の単位は m，$\boldsymbol{e}_{rot}$ の単位は rad である．

**:init** *_target-coords _move-target &key ((:translation-axis _translation-axis) t) ((:rotation-axis _rotation-axis) t) ((:translation-coords _translation-coords) :local) ((:rotation-coords _rotation-coords) :local) ((:wtrans _wtrans) 1.0) ((:wrot _wrot) 1.0) ((:check _check) t) ((:thre _thre) 0.001) ((:rthre _rthre) (deg2rad 1)) ((:p-limit _p-limit) 0.1) ((:r-limit _r-limit) 0.5)* [method]

- target-coords, move-target
  *coordinate* クラス. どちらも動いて良い
- *translation-axis : t :x :y :z :xy :yx :yz :zy :zx :xz nil*
- *rotation-axis : t :x :y :z nil*
- *translation-coords*
  *translation-axis* 及び *Wtrans* は *translation-coords* 系で表現される．*:local* なら *move-target* 系，*:world* なら *world* 系，*coordinates* クラスならその座標系である．
- *rotation-coords*
  *rotation-axis* 及び *Wrot* は *rotation-coords* 系で表現される．*:local* なら *move-target* 系，*:world* なら *world* 系，*coordinates* クラスならその座標系である．
- *Wtrans*
  *float* クラスの場合 $\mathbf{W}_{trans}$ はその値を対角成分に並べた行列になる
  *vector* クラスの場合 $\mathbf{W}_{trans}$ は各値を対角成分に並べた行列になる
  *matrix* クラスの場合 $\mathbf{W}_{trans}$ としてそのまま使用される．
- *Wrot*
  *float* クラスの場合 $\mathbf{W}_{rot}$ はその値を対角成分に並べた行列になる
  *vector* クラスの場合 $\mathbf{W}_{rot}$ は各値を対角成分に並べた行列になる
  *matrix* クラスの場合 $\mathbf{W}_{rot}$ としてそのまま使用される．
- *check : t or nil*
  終了判定を行うか否か
- *thre, rthre*
  *float* クラスまたは *float-vector* クラス．終了判定時にこの値以下の侵犯を許容する．単位は m,rad

- p-limit, r-limit
  一回の反復計算で動く並進ノルム・回転ノルムの上限．単位は m,rad. 特に回転については変位が大き過ぎると線形近似誤差の影響によって計算が収束しない．

**:initialize** *variable-length variables* [method]
探索変数 *variables* のうち，*target-coords,move-target* に影響を与える成分を調べ，記録する．

**:update** [method]
現在の状態近傍で線形近似し **A**, ***b*** を更新する

**:is-satisfied** *&optional force-check* [method]
move-target と target-coords が一致しているかどうかを判定

**:draw** *&key ((:viewer vwer) ∗viewer∗)* [method]
move-target と target-coords を描画

**:debug** [method]
現在のエラーを表示する

## cog-task [class]

| | |
|---|---|
| :super | **inverse-kinematics-task** |
| :slots | (target 重心を考える対象) |
| | (target-variables , 重心に影響を与える variable のリスト) |
| | (target-jacobis , 各 target-variables に対応した重心ヤコビアンのリスト) |
| | (tmp-v0) |
| | (tmp-v1) |
| | (tmp-v2) |
| | (tmp-va) |
| | (tmp-vb) |
| | (tmp-vc) |
| | (tmp-vd) |
| | (tmp-ma) |
| | (tmp-mb) |
| | (tmp-mc) |

重心を扱うための仮想クラス．単位は m

**:init** *_target* [method]

- target
  *cascaded-link* クラス．

**:initialize** *variable-length variables* [method]
探索変数 *variables* のうち，*target* に影響を与える成分を調べ，記録する．

**:update** [method]
現在の状態近傍で線形近似し target-jacobis に重心ヤコビアンを入れる

**:draw** *&key ((:viewer vwer) ∗viewer∗)* [method]

重心を描画

## target-centroid-pos-task [class]

:super **cog-task**
:slots (target-centroid-pos)
(cog-translation-axis ,represented in translation-coords)
(cog-translation-coords)
(check 終了判定を行うか否か)
(centroid-thre 終了判定時の並進許容誤差)
(b-raw , min-max 適用前の $\boldsymbol{b}$)
(p-limit 一回の反復計算で動く並進ノルムの大きさの上限 [m])

重心を目標位置に一致させるタスクを表すクラス.
重心の目標位置までの並進誤差を *cog-translation-coords* の座標系で表現し，*cog-translation-axis* によって抽出された成分を $\boldsymbol{e}$ と表す．$\boldsymbol{e}$ のヤコビアンをそれぞれ $\mathbf{J}_{cog}$ とすると，*prioritized-inverse-kinematics* 中で次の問題に変換される.

$$\min_{\boldsymbol{x}} \qquad (\mathbf{J}_{cog}\boldsymbol{x}+\boldsymbol{e})^{\mathrm{T}}\mathbf{W}(\mathbf{J}_{cog}\boldsymbol{x}$$

$\boldsymbol{e}$ の単位は m である.

**:init** *_target &key ((:target-centroid-pos _target-centroid-pos))* [method]
*((:cog-translation-axis _cog-translation-axis) :z)*
*((:cog-translation-coords _cog-translation-coords) (make-coords))*
*((:w _w) 1.0)*
*((:check _check) t)*
*((:centroid-thre _centroid-thre) 0.001)*
*((:p-limit _p-limit) 0.1)*

- target
  *cascaded-link* クラス. このオブジェクトの重心を考える.
- *target-centroid-pos*
  サイズ *3* の *float-vector* クラス．目標重心位置であり，*world* 系で表す．単位は mm.
- cog-translation-axis : t :x :y :z :xy :yx :yz :zy :zx :xz nil
- cog-translation-coords
  *cog-translation-axis* 及び *W* は *translation-coords* 系で表現される
- *W*
  *float* クラスの場合 **W** はその値を対角成分に並べた行列になる
  *vector* クラスの場合 **W** は各値を対角成分に並べた行列になる
  *matrix* クラスの場合 **W** としてそのまま使用される.
- *check : t or nil*
  終了判定を行うか否か
- *centroid-thre*
  *float* クラスまたは *float-vector* クラス．終了判定時にこの値以下の侵犯を許容する．単位は m
- p-limit
  一回の反復計算で動く並進ノルムの上限．単位は m.

**:initialize** *variable-length variables* [method]

探索変数 *variables* のうち，重心に影響を与える成分を調べ，記録する．

**:update** [method]

現在の状態近傍で線形近似し $\mathbf{A}$, $\boldsymbol{b}$ を更新する

**:is-satisfied** *&optional force-check* [method]

重心が目標位置と一致しているかどうかを判定

**:draw** *&key ((:viewer vwer) *viewer*)* [method]

重心と目標位置を描画

**:debug** [method]

現在のエラーを表示する

## centroid-support-polygon-task [class]

:super **cog-task**

:slots (polygons 各 support polygon を表す polygon のリスト)
(polygons-variables 各 polygon に影響を与える variable のリストのリスト)
(polygons-jacobis)
(normal 法線ベクトル)
(margin , support polygon をこの値だけ縮小する)
(check 終了判定を行うか否か)
(centroid-thre 終了判定時の許容誤差)
(w , 重み)
(hull , 現在の polygons の hull)

重心を support polygon の 2 次元の範囲内に位置させるタスクを表すクラス．
法線ベクトル $\boldsymbol{n}$ に対して半時計回りに並んだ凸包 $(\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, ...)$ の表す領域内に重心 $\boldsymbol{c}$ を存在させる．$\boldsymbol{n}$ に垂直な平面に各点を射影し $\boldsymbol{n}$ を z 軸とする local 座標系で各点を ${}^0\boldsymbol{p}_0, {}^0\boldsymbol{p}_1, {}^0\boldsymbol{p}_2, ..., {}^0\boldsymbol{c}$ と表現する．点 ${}^0\boldsymbol{p}_i$ と点 ${}^0\boldsymbol{p}_{i+1}$ の距離を $l_i$ とすると，直線の左側を正とした ${}^0\boldsymbol{c}$ と直線 ${}^0\boldsymbol{p}_i {}^0\boldsymbol{p}_{i+1}$ の符号付き距離は，$d_i = \frac{({}^0y_i - {}^0y_{i+1}){}^0x_c + ({}^0x_{i+1} - {}^0x_i){}^0y_c + {}^0x_i{}^0y_{i+1} - {}^0x_{i+1}{}^0y_i}{l}$ である．よって，$\boldsymbol{d}$ のヤコビアンを $\mathbf{J}$ とすると，*prioritized-inverse-kinematics* 中で次の問題に変換される．

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{\omega}} \quad & \boldsymbol{\omega}^{\mathrm{T}}\mathbf{W}\boldsymbol{\omega} \\ \text{subject to} \quad & \boldsymbol{d} + \mathbf{J}\boldsymbol{x} + \boldsymbol{\omega} \geqq \mathit{margin} \end{aligned}$$

なお,

$$\begin{aligned} \mathbf{J}_i \;=\; & \frac{1}{l}[({}^0\boldsymbol{p}_{i+1} - {}^0\boldsymbol{p}_i)\times]_{z\,\text{行}} \frac{\partial}{\partial \boldsymbol{x}} \begin{bmatrix} {}^0x_c \\ {}^0y_c \\ 0 \end{bmatrix} + \left\{ \frac{1}{l}[({}^0\boldsymbol{c} - {}^0\boldsymbol{p}_{i+1})\times]_{z\,\text{行}} + \frac{d}{l^2}({}^0\boldsymbol{p}_{i+1} - {}^0\boldsymbol{p}_i)^{\mathrm{T}} \right\} \frac{\partial}{\partial \boldsymbol{x}} \begin{bmatrix} {}^0x_i \\ {}^0y_i \\ 0 \end{bmatrix} \\ & + \left\{ \frac{1}{l}[({}^0\boldsymbol{p}_i - {}^0\boldsymbol{c})\times]_{z\,\text{行}} + \frac{d}{l^2}({}^0\boldsymbol{p}_i - {}^0\boldsymbol{p}_{i+1})^{\mathrm{T}} \right\} \frac{\partial}{\partial \boldsymbol{x}} \begin{bmatrix} {}^0x_{i+1} \\ {}^0y_{i+1} \\ 0 \end{bmatrix} \end{aligned}$$

である．単位は m である．

**:init** *_target &key ((:polygons _polygons))* [method]
*((:normal _normal) (float-vector 0 0 1))*

*((:margin _margin) 0.0)*
*((:w _w) 1.0)*
*((:check _check) t)*
*((:centroid-thre _centroid-thre) 0.001)*

- target
  *cascaded-link* クラス. このオブジェクトの重心を考える.
- *polygons*
  各 *support polygon* を表す *polygon* クラスのリスト.*woeld* 座標系で，単位は m. 各反復ごとにこれらの *polygon* の *convex hull* を計算し，この範囲内に重心を制限する.
- *normal*
  サイズ *3* の *float-vector* クラス．上方向を表す法線ベクトルであり，このベクトルで表される *2* 次元平面で *support polygon* を考える.
- *margin*
  *float* クラス．重心を *support polygon* の端からこの値以上離す．単位は *[m]*.
- $W$
  *float* クラス.$\mathbf{W}$ はこの値を対角成分に並べた行列になる

- *check : t or nil*
  終了判定を行うか否か
- *centroid-thre*
  *float* クラス．終了判定時にこの値以下の侵犯を許容する．単位は m

**:initialize** *variable-length variables* [method]
探索変数 *variables* のうち，重心や *polygons* に影響を与える成分を調べ，記録する.

**:update** [method]
現在の状態近傍で線形近似し $\mathbf{C}$,$\mathbf{C}_{sparce}$, $\boldsymbol{d}_l$，$\boldsymbol{d}_u$, $\mathbf{W}_c$ を更新する

**:is-satisfied** *&optional force-check* [method]
重心が領域内にあるかどうかを判定

**:draw** *&key ((:viewer vwer) ∗viewer∗)* [method]
重心と領域を描画

**:debug** [method]
現在のエラーを表示する

## centroid-scfr-task [class]

:super **cog-task**
:slots (contact-constraints)
(contact-coords)
(margin , support polygon をこの値だけ縮小する)
(check 終了判定を行うか否か)
(centroid-thre 終了判定時の許容誤差)
(w , 重み)
(a_eq)
(b_eq)

(a_ineq)
(b_ineq)
(epsilon)

重心を静的重心実行可能領域内に位置させるタスクを表すクラス．重心の位置は最適化するが，外界接触位置を最適化して静的重心実行可能領域を能動的に変化させるような処理はしない．
外界接触部位でロボットが受ける接触レンチ (6 自由度,world 系, 接触部位まわり) を並べたベクトルを $\boldsymbol{w}$ とし，接触力拘束を $\mathbf{C}\boldsymbol{w}+\boldsymbol{d} \geqq 0$，力の静的釣り合いを $\mathbf{G}\boldsymbol{w}+[\ 0 \quad 0 \quad -mg \quad -c_y mg \quad c_x mg \quad 0\ ]^{\mathrm{T}} = 0$ とする．これらの式によって制限される $\boldsymbol{w'} = [\ c_x \quad c_y \quad \boldsymbol{w}^{\mathrm{T}}\ ]^{\mathrm{T}}$ の存在領域は，

$$\left\{ \boldsymbol{w'} \in \mathbb{R}^d : \begin{bmatrix} \mathbf{0}_3 & \mathbf{0}_3 & \\ 0 & -mg & \mathbf{G} \\ mg & 0 & \\ 0 & 0 & \end{bmatrix} \boldsymbol{w'} = 0, \begin{bmatrix} \mathbf{0} & \mathbf{C} \end{bmatrix} \boldsymbol{w'} \geqq \boldsymbol{d} \right\}$$

となり，これは d 次元空間の凸多面体 (閉じていなくともよい) である．この多面体の頂点を考えることで，$\boldsymbol{w'}$ の存在領域は，

$$\left\{ \boldsymbol{w'} \in \mathbb{R}^d : {}^{\exists}\boldsymbol{y}, {}^{\exists}\boldsymbol{z}, \boldsymbol{w'} = \mathbf{V}\boldsymbol{y} + \mathbf{R}\boldsymbol{z}, \sum_i \boldsymbol{y}_i = 1, \boldsymbol{y} \geqq 0, \boldsymbol{z} \geqq 0 \right\}$$

とも表せる．この頂点 $\mathbf{V}$, $\mathbf{R}$ を $c_x$ - $c_y$ 平面に射影した $\mathbf{V}'$, $\mathbf{R}'$ を用いることで，静的重心実行可能領域が次のように得られる．

$$\left\{ c_x, c_y : {}^{\exists}\boldsymbol{y}, {}^{\exists}\boldsymbol{z}, \begin{bmatrix} c_x \\ c_y \end{bmatrix} = \mathbf{V}'\boldsymbol{y} + \mathbf{R}'\boldsymbol{z}, \sum_i \boldsymbol{y}_i = 1, \boldsymbol{y} \geqq 0, \boldsymbol{z} \geqq 0 \right\}$$

この多面体を不等式制約の形式で表現すれば，静的重心実行可能領域を表す不等式制約が得られる．

$$\left\{ c_x, c_y : \mathbf{A} \begin{bmatrix} c_x \\ c_y \end{bmatrix} + \boldsymbol{b} \geqq 0 \right\}$$

重心 $c_x$,$c_y$ のヤコビアンを $\mathbf{J}$ とすると，*prioritized-inverse-kinematics* 中で次の問題に変換される．

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{\omega}} \quad & \boldsymbol{\omega}^{\mathrm{T}}\mathbf{W}\boldsymbol{\omega} \\ \text{subject to} \quad & \mathbf{A}(\begin{bmatrix} c_x \\ c_y \end{bmatrix} + \mathbf{J}\boldsymbol{x}) + \boldsymbol{\omega} \geqq -\boldsymbol{b} + margin \end{aligned}$$

単位が m となるよう $\mathbf{A}$,$\boldsymbol{b}$ はスケーリングされる．

**:init** *_target &key ((:contact-constraints _contact-constraints))* [method]
*((:contact-coords _contact-coords))*
*((:margin _margin) 0.0)*
*((:w _w) 1.0)*
*((:check _check) t)*
*((:centroid-thre _centroid-thre) 0.001)*
*((:epsilon _epsilon) 5)*

- target
  *cascaded-link* クラス. このオブジェクトの重心を考える．

- *contact-constraints*
  *contact-constraint* クラスのリスト．これらによって定義される静的重心実行可能領域内に重心を制限する．
- *contact-coords*
  *coordinates* クラスのリスト．それぞれの *contact-constraint* の座標系を表す．
- *margin*
  *float* クラス．重心を静的重心実行可能領域の端からこの値以上離す．単位は *[m]*.
- *W*
  *float* クラス.**W** はこの値を対角成分に並べた行列になる

- *check : t or nil*
  終了判定を行うか否か
- *centroid-thre*
  *float* クラス．終了判定時にこの値以下の侵犯を許容する．単位は m
- epsilon
  *integer* クラス．多面体計算時の離散化に用いる．

**:initialize** *variable-length variables* [method]
探索変数 *variables* のうち，重心に影響を与える成分を調べ，記録する．

**:update** [method]
現在の状態近傍で線形近似し $\mathbf{A}$,$\mathbf{A}_{sparce}$, $\boldsymbol{b}$, $\mathbf{W}_a$,$\mathbf{C}$,$\mathbf{C}_{sparce}$, $\boldsymbol{d}_l$, $\boldsymbol{d}_u$, $\mathbf{W}_c$ を更新する

**:is-satisfied** *&optional force-check* [method]
重心が領域内にあるかどうかを判定

**:draw** *&key (:viewer vwer) *viewer**) [method]
重心と領域を描画

**:debug** [method]
現在のエラーを表示する

## inverse-kinematics-variable [class]

:super **propertied-object**
:slots (dim , dimension of variable)
(index , index から index + dim - 1] 番目の $\boldsymbol{x}$ の要素が対応する)
(initial-state , prioritized-inverse-kinematics の初期状態)

探索変数を表現するクラス

**:init** [method]
Initialize instance

**:index** *&optional idx* [method]
update or return index

**:apply-x** *x* [method]

計算された $\boldsymbol{x}$ の対応する要素を実際に適用する

**:revert** [method]

*prioritized-inverse-kinematics* の初期状態に戻す

**:init-form** [method]

*prioritized-inverse-kinematics* の開始時に一回呼ばれる.*inverse-kinematics-task* の*:initialize* より前に呼ばれる

**:cleanup-form** [method]

*prioritized-inverse-kinematics* の終了時に *unwind-protect* を用いて一回呼ばれる.

**:dim** [method]

return dim

## value-variable [class]

:super **inverse-kinematics-variable**
:slots (value)
(value-scale)

value のクラス.

**:init** *&key (initial-value 0.0)* [method]
*((:value-scale _value-scale) 1.0)*

最大値等を表現するための探索変数. $\boldsymbol{x}$ に *value-scale* を乗じたものが *value* の更新量に相当する. ユーザが直接使うことはない.

**:apply-x** *x* [method]

$\boldsymbol{x}$ の値だけ joint-angle を相対的に更新する.

**:revert** [method]

prioritized-inverse-kinematics の初期状態に戻す

**:value** *&optional _value* [method]

return value

**:value-scale** [method]

return value-scale

## joint-variable [class]

:super **inverse-kinematics-variable**
:slots (j , joint)

joint のクラス.

**:init** *_joint* [method]

*joint* の *joint-angle* に相当する探索変数. $\boldsymbol{x}$ は *joint* の変位に相当する. $\boldsymbol{x}$ の単位は *degree, rad.*

**:apply-x** *x* [method]

$\boldsymbol{x}$ の値だけ joint-angle を相対的に更新する.

**:revert** [method]

prioritized-inverse-kinematics の初期状態に戻す

**:joint** [method]

return joint

## virtual-joint-variable [class]

:super **joint-variable**
:slots (child , virtual joint の子リンク)
(parent , virtual joint の親リンク)

仮想関節の joint-angle に相当する探索変数．$\boldsymbol{x}$ は仮想関節の変位に相当する．$\boldsymbol{x}$ の単位は degree, rad.

**:init** *_child &key (joint-type 6dof-joint)* [method]
*((:parent _parent) (instance bodyset-link :init (make-cascoords :pos (copy-object (send _child :world*
*joint-args*

*child* は仮想関節の子リンクとなる．*child* は *cascaded-link* クラスか *bodyset-link* であり，*cascaded-link* クラスのオブジェクトであるなら *(car (send *robot* :links))* を与えるのではなくそのまま **robot** を与えること．

**:init-form** [method]

仮想関節を取り付ける

**:cleanup-form** [method]

仮想関節を除去する