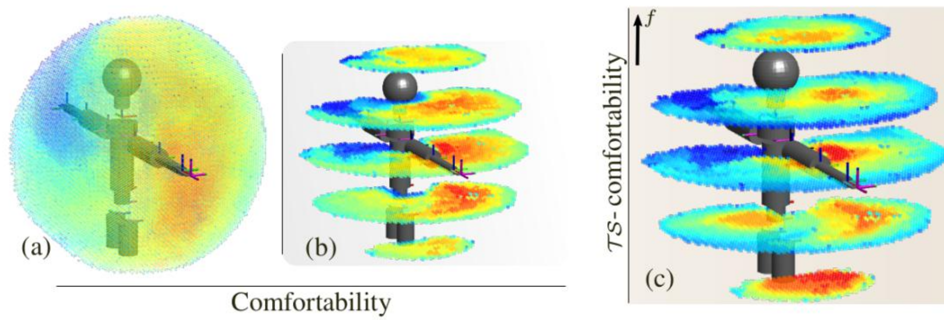
VERSION: VS. 0.3.0
2020-06
AUTHORS: Luis F.C. Figueredo, Mehmet Dogar, Anthony Cohn
University of Leeds
Contact information: figueredo@ieee.org

RHuMAn - Documentation

Release 0.3.0

Rapid Human Manipulability Assessment
(RHuMAn)



Contents

Introduction to Human Comfortability....5
Rapid Human Manipulability Assessment....6
rHuManModel....9
    Examples - Constructor....10
    Variables....10
    rHuManModel / getFKM....13
    rHuManModel / getIK....14
    rHuManModel / getPos....15
    rHuManModel / getOrientation....15
    rHuManModel / getJacobGeom....16
    rHuManModel / checkSelfCollision....17
    rHuManModel / getSelfCollisionPenalties....17
    rHuManModel / getRandJoints....18
    rHuManModel / getRandGaussianJoints....19
rHuManManipulability....22
    Examples - Constructor....23
    Variables....24
    rHuManManipulability / build_AugmentedComfortDataset....28
    rHuManManipulability / build_TSComfortability....29
    rHuManManipulability / compute_gainMuscle....30
    rHuManManipulability / get_ComfortCost....31
    rHuManManipulability / get_muscularCost....33
    rHuManManipulability / get_RULA....34
    rHuManManipulability / get_TSComfortDataset....36
    rHuManManipulability / handleErgoIndex2RULA....37
    rHuManManipulability / handleRula2ErgoIndex....37
    rHuManManipulability / plot_comfortability....38
    RHuManManipulability / reshape_Comfortability....39
    rHuManManipulability / reshape_TSComfortability....40
    rHuManManipulability / setForces4Manipulability....41
rHuMAn for USERS....43
License....50
People....51

Introduction to Human Comfortability

The ability of computing a quality index for different configurations and manipulation tasks has been widely used in robotics yet little explored in human manipulation and physical human-robot collaboration (pHRC).

Most of the scarce works that have so far addressed the quality of human arm posture have either focused on task heuristics, human biomechanics capabilities or ergonomics alone leading to elaborate methods that are narrow to task-specific applications. Indeed, predicting and/or proactively shaping human posture through pHRC is a complex procedure that often involves different optimization techniques over a highly redundant muscular- and joint-spaces.

This software instead proposes solving this problem through a combination of extensive offline analysis of the workspace capabilities of the human arm (in terms of muscular load, ergonomics, and joint/kinematic/force constraints) with efficient online processing. The proposed approach builds a human manipulation performance distribution in terms of muscular and ergonomy-based metrics in workspace which can be quickly tailored to specific tasks and filtered for design purposes. This methodology considerably simplifies human manipulability assessment for both general and task-specific applications and, in contrast to existing works, is suitable for real-time and/or resource-limited applications.

Numerical evidence shows the proposed human manipulability analysis largely outperforms previous results in terms of computing time and even in finding optimal configurations. The developments draw upon existing musculoskeletal modeling and ergonomic postural analysis through RULA, and respond to key trends in human-centred robotics.

Rapid Human Manipulability Assessment

(RHuMAn)

RHuMAn (Rapid Human-Manipulability Assessment): is a tool for collaborative AI applications that allows computation of a human comfort quality index distribution in the workspace. RHuMAn draws upon extensive offline analysis of human biomechanics capabilities with ergonomics assessment from industrial-standard directives to produce an efficient online comfort-quality assessment that can be quickly tailored to specific tasks and purposes.

For further details on the methods, readers are referred to the attached ArXiv paper that comes with this software.

The aims of RHuMAn is to provide an easy-to-use platform to analyze human kinematics and geometric aspects combining them with ergonomics and musculoskeletal assessement associated with different posture and tasks. The main goal is to facilitate the analysis of elaborate algorithms that combines knowledge from biomechanics, extensive experimental data, and optimization techniques into one single framework. For instance, to analyze muscular-activity, one needs to be familiar with biomechanics software, e.g., Opensim, and adapt models using its API or GUI (only available for windows) to be used in different applications (e.g., using in robotics requires for someone to design the D-H parameters of the Opensim model (which is not trivial) into a robotics kinematics tool). These software are computationally heavy and hard to combine with different human metric analysis (e.g., ergonomic concepts like RULA). RHuMAn simplifies biomechanics and ergonomics analysis making it straightforward for user from different backgrounds.

RHuMAn brings an easy-to-use toolbox combined with a database of biological parameters extracted from OpenSim and a tool that enables one to extract different parameters (for different model and anthropometry features). A typical usage of RHuMAn is to analyze motion performance and to design physical human-robot collaborative planners that take human-informed features into account.

Finally, an implicit objective of RHuMAn is to introduce musculoskeletal model-based methods to practical applications since despite recent advances, these methods have not been tested in industry—in constrast to ergonomics—hence, there are still resistance in its use regardless the advantages. In this sense, a solution that combines both strategies may lead the way of muscular-informed methods to real-world industrial and pHRC domains, e.g., a factory may replace their assessment from fully ergonomic-based (e..g, RULA) to one that increasingly relies on muscular-activity as designer acquires confidence on musculoskeletal model-based methods.

rHuManModel

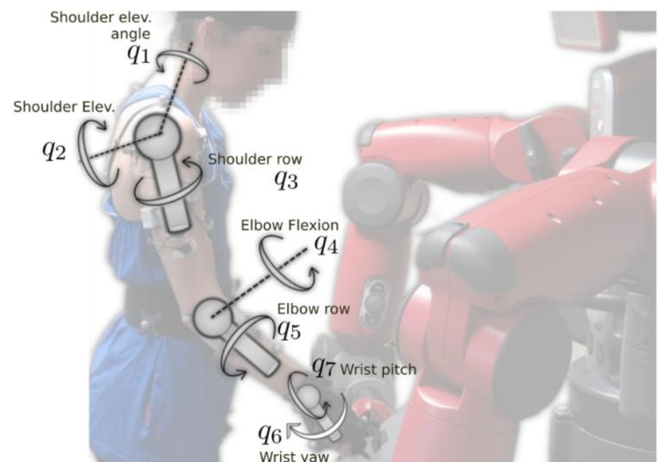
RapidHuman-Manipulability Assessment (**RHuMAn**)

rHuManModel

RapidHuman-Manipulability Assessment (**RHuMAn**)
CLASS Definition for **rHuManModel**

Description

rHumanModel is an object with Kinematics according to Saul et al.[1] combined with the [DQ_Robotics toolbox] to the computation of the serial chain kinematics according to the Figure in right.



Note the **world-frame** used in rHumanModel is **Z-up** with **X** being sideways (positive from right shoulder out) and **Y** facing the front of the person.
For the hand (wrist-frame), **Z** is the axis from wrist to raised fingers (base of the fingers), **X** depicts the palm down, and **Y** the axis formed from the thumbs up.

Output

RHuMAn object with variables and methods (see below).

Inputs [optional]

All following inputs are optional and follow the struct: "input",data

------------[Kinematics based inputs]

- **'dq_shoulderBase',double(8,1):**
  - (DQ-pose) indicating the pose of the shoulder wrt the world in unit dual-quaternions [default: [1;zeros(7,1)]]

[1] Saul et al., "Benchmarking of dynamic simulation predictions in two software platforms using an upper limb musculoskeletal model Benchmarkig", CMBBE, (2015).

- **'shoulderHeight',double:**
  - (height) indicating the position of shoulder wrt the world (in the z-axis). It will be overturned by the option shoulderBase [default: 0]
- **'upperArm',double:**
  - (length) of the upper-arm limb. [default: 0.302]
- **'foreArm',double:** (length) of the fore-arm limb. [default: 0.2795]
- **'hand',double:** (length) to the center of the hand (where forces will be applied/exerted).[default: 0.05]
- **'jointlimits',double(7,2):** (joints) block matrix formed by [joints_min joints_max]
- **'joint_lowerlimits',double(7,1):** (joints) vector of minimum joint limits. It will be overturned by the option jointlimits
  - [default: [-90 0 -80 0 -90 0 -60]'*(pi/180)]
- **'joint_upperlimits',double(7,1):** (joints) vector of maximum joint limits. It will be overturned by the option jointlimits
  - [default: [130 180 +40 130 +90 +25 +60]'*(pi/180)]

------------[Human collision assessment]

- **'humanbox',double(3,2):** (range) block matrix building a 3D box for the body [xmin xmax; ymin ymax; zmin zmax] - [default [-0.3145 -0.0355; -0.065 0.065; -0.60 0.025]]
- 'humanheadCenter',double(3,1): (position) vector [x;y;z] with center position for the head [default: [-0.20; 0; 0.15]]
- **'humanheadRadius',double:** (length) radius for the head (depicted as a 3D ball) [default: 0.14]

------------[Extra]

- **'verbose',logical:** Print all points and steps

Examples - Constructor

```
newhumanmodel = rHuManModel();
newhumanmodel = rHuManModel('verbose',true);
newhumanmodel = rHuManModel('dq_shoulderBase',[1 zeros(1,7)]);
newhumanmodel = rHuManModel('verbose',true,'upperArm',0.35);
newhumanmodel = rHuManModel('verbose',true,'upperArm',0.35,'foreArm',0.30);
newhumanmodel = rHuManModel('hand',0);
newhumanmodel = rHuManModel('shoulderHeight',1.35,'humanheadCenter',[-0.2 0 1.50],'humanbox',[-0.35 0; -0.05 0.05;    0.75 1.3750])
newhumanmodel = rHuManModel('humanbox',[-0.3145 -0.0355; -0.065 0.065;    -0.60 0.025],'humanheadRadius',0.15);
newhumanmodel = rHuManModel('joint_lowerlimits',zeros(7,1),'joint_upperlimits',ones(7,1));
```

Variables

kine	DQ_kinematics representation for the arm (need external package)
kineconfig	Struct with configurations for the object
hand2tool	Additional transformation from hand to tool (in unit dual quaternions)
pointsInArm	Array of cell with positions along the arm (for collision detection)

Methods

getFKM

poseout = getFKM(theta)
=> returns the Forward Kinematics for a given joint config theta

getIK

jointout = getIK(pose);
=> Returns the Inverse Kinematics for a given pose (in unit dual quaternions w DQ_kinematics package)
getIK(pose); => pose => class(DQ) OR OR P=double(7,1) for pose [orientation; position]
getIK(position); => position => double(3,1) for position-only [x;y;z]
getIK(orientation); => orientation => double(4,1) fororientation-only (quaternion: [a;b;c;d]= a + bi + cj + dk)
[jointout, output] = getIK(.); => output is the result from IK Optimization (multistart SQP)

getPos

position = getPos(theta) => returns the position [x;y;z] from FKM for a given joint config theta

getOrientation

orientation=getOrientation(theta) => returns the orientation (quaternion [a;b;c;d]= a + bi + cj + dk) from FKM for a given joint config theta

getJacobGeom

geomJ = getJacobGeom(theta); => Returns geometric Jacobian (at joint configuration theta)

checkSelfCollision

boolcollision = checkSelfCollision(theta) => Check for collision (returns true if collision is detected) and updates the pointsInArm (at joint configuration theta)

getSelfCollisionPenalties

penalties = getSelfCollisionPenalties(theta); => Compute penalties (internally calls checkSelfCollision) from (0 = collision) to (1 = no penalty).
= getSelfCollisionPenalties(theta,forces); => [Optional:forces] Adds forces (6xn) array of n-wrenches in task-space

getRandGaussianJoints

randnjoints = getRandGaussianJoints(); => Returns a 7-joint vector with random (Gaussian) with mean: joint-mean (between limits) and std (from joint-range) (within joint limits).
= getRandGaussianJoints('seed',seedopt); => Defines the seed for the random generator (e.g., getRandGaussianJoints('seed','shuffle')) [default: shuffle]

getRandJoints

randjoints = getRandJoints(); => Returns a 7-joint vector with random (Uniform) values (within joint limits).

= getRandJoints(logvec); => [Optional:logvec] => logical(7,1) where 0 implies joint=mean(between limits) and 1 implies rand values.

= getRandJoints('seed',seedopt); => [Optional:2args] Defines the seed for the random generator (e.g., getRandGaussianJoints('seed','shuffle')) [default: shuffle]

= getRandJoints('length',N); => [Optional:2args] Using 'length' followed by N where N is the length for output random values, i.e., randjoints double(7,N)

Change log

[2020-06-01]%%=> rHuManModel.m created
[2020-06-29]%%=> class rHuManModel vs(0.3.0) published

rHuManModel / getFKM

return pose in dual quaternion (forward kinematics)
Compute forward kinematics from joints

Inputs

joints - double(7,1)

Outputs

pose - class DQ with orientation and position (in unit dual quaternions)

Usage

```
%Computing FKM    (return class DQ)

pose = getFKM(joints)
```

rHuManModel / getIK

return joint configuration (inverse kinematics)
Compute inverse kinematics from pose (or position)

Inputs

pose	-	DQ (class DQ included in this package - from DQrobotics package)
OR pose	-	Double (7,1) - for [quaternion;position] = [[a;b;c;d]; x;y;z]
OR orientation	-	Double (4,1) - for quaternion only [a;b;c;d]= a + bi + cj + dk
OR position	-	Double (3,1) - [x;y;z] (for position only)

Outputs

- **joints** - double(7,1) of joints
- **output** - optimization output (multistart sqp)

Usage

```
%   Example (orientation and position)
        % Building pose (Example)
        phi=pi/2;
         position = [0.25; 0; 0.25];
        % Defining rotation quaternion
        pose = DQ([cos(phi) sin(phi) 0 0]);
        % Defining position quaternion
        pose = pose + DQ.E*(0.5)*DQ([0;position]);

   % Finding IK
   q = getIK(pose)
   [q, output] = getIK(pose)

  %   Example (only position (any orientation))
  % Finding IK
   position = [0.25; 0; 0.25];
   q = getIK([0; position])
   [q, output] = getIK([0; position])
```

rHuManModel / getPos

return position [x;y;z] from forward kinematics
Compute position forward kinematics from joints

Inputs

joints - double(7,1)

Outputs

position - double(3,1) with position

Usage

```
%Computing FKM (return [x;y;z])
position = getPos(joints)
```

rHuManModel / getOrientation

return orientation (quaternion [a;b;c;d]= a + bi + cj + dk) from forward kinematics
Compute orientation forward kinematics from joints

Inputs

joints - double(7,1)

Outputs

orientation - double(4,1) with orientation in quaternion [a;b;c;d]= a + bi + cj + dk

Usage

```
% Computing FKM    (return quaternion [a;b;c;d]= a + bi + cj + dk)
  rot = getOrientation(joints)
```

rHuManModel / getJacobGeom

return geometric Jacobian
Compute geometric Jacobian from joint inputs

Inputs

joints - double(7,1)

Outputs

geometric jacobian (6,7) matrix

Usage

J = **getJacobGeom**(theta)

rHuManModel / checkSelfCollision

update points in the arm (for collision detection)
Divides the arm cylinders in different balls and check distance to balls
if it respects a given radius then there is no collision

Inputs

joints - double(7,1)

Outputs

% Returns 1 if collision is detected.

Usage

[collisiondetected] = **checkSelfCollision**(theta7)

rHuManModel / getSelfCollisionPenalties

get self collision penalties
Compute penalties (internally calls checkSelfCollision)
Read referenced paper for further details on details of how it works

Inputs

theta - double(7,1) with joint values
[Optional:forces] - Adds forces (6xn) array of n-wrenches in task-space

Outputs

penaltOut - Returns penalty value (for single input or single force)
- Returns penalty vector for multiple-forces (1 for each)
- Returns 0 or (vector of zeros) if self-collision is detected.

rHuManModel / getRandJoints

Returns a 7-joint vector with uniformly random values (within joint limits).
Get random joints (uniform distribution within joint limits).

Inputs - Optional

[logical(7,1)] - where 0 implies joint=mean(between limits) and 1 implies rand values.

['seed',seedopt] - Defines the seed for the random generator (e.g., getRandGaussianJoints('seed','shuffle')) [default: shuffle]
After use it returns to previous rng.

['length',N] - Using 'length' followed by N where N is the length for output random values, i.e., randjoints double(7,N)

Outputs

joints - double(7,1)
OR
double(7,N) if ['length',N] is used.

Usage

joints = **getRandJoints**(joints)
= **getRandJoints**(joints, [0; 0; 0; 0; 1; 1; 1]) %only wrist random with joint-mean(between limits) in shoulder and elbow-flexion joints
= **getRandJoints**('seed','default'); => Initializes Mersenne Twister generator with seed 0. This is the default setting at the start of each MATLAB session.
= **getRandJoints**('length',1000); => Returns 1000 random joints double(7,1000).
= **getRandJoints**([0; 0; 0; 0; 1; 1; 1],'length',1000); => Returns 1000 random joints double(7,1000) where only the last 3-joints are random..

rHuManModel / getRandGaussianJoints

Returns a 7-joint vector with random (Gaussian) values (within joint limits).
Get random joints (normal distribution with mean in joint-mean (between limits) and std (from joint-range) that satisfies the joint limits.

Outputs

joints - double(7,1)

rHuManManipulability

Rapid-Human-Manipulability Assessment (**RHuMAn**)

rHuManManipulability

RapidHuman-Manipulability Assessment (**rHuMAn**)
CLASS Definition for **rHuManManipulability**

Description

rHuManManipulability is an object with the tools and methods described in the comfortability analysis paper attached to this software. The aim is to provide **muscular-informed analysis** and **ergonomics analysis** in an unified framework and to combine both metrics into a **comfortability dataset** into a **comfortability distribution** (through a voxalized analysis of the dataset) or **task-comfortability (TS-comfortability) dataset** or **TS-comfortability distribution.**

The solution is build over a human model described using the class rHuManModel (which takes Kinematics according to Saul et al.[2] combined with the DQ_Robotics toolbox to the computation of the serial chain kinematics), an ergonomics analysis based on rapid upper-limb assessment (RULA),[3] common to industrial applications, and a muscular-informed analysis using Saul et al model combined with Opensim tool for analysis. This software also allows precomputed human biomechanics parameters to be obtained offline through Opensim and stored for independent software analysis.

*Note the **world-frame** used in rHumanModel is **Z-up** with **X** being sideways (positive from right shoulder out) and **Y** facing the front of the person.*
***For the hand (wrist-frame)**, **Z** is the axis from wrist to raised fingers (base of the fingers), **X** depicts the palm down, and **Y** the axis formed from the thumbs up.*

[2] Saul et al., "Benchmarking of dynamic simulation predictions in two software platforms using an upper limb musculoskeletal model Benchmarkig", CMBBE, (2015).
[3] M. Lynn and N. Corlett, "RULA: A survey method for the investigation of work-related upper limb disorders," Applied Ergonomics, vol. 24, no. 2, pp. 91–99, 1993

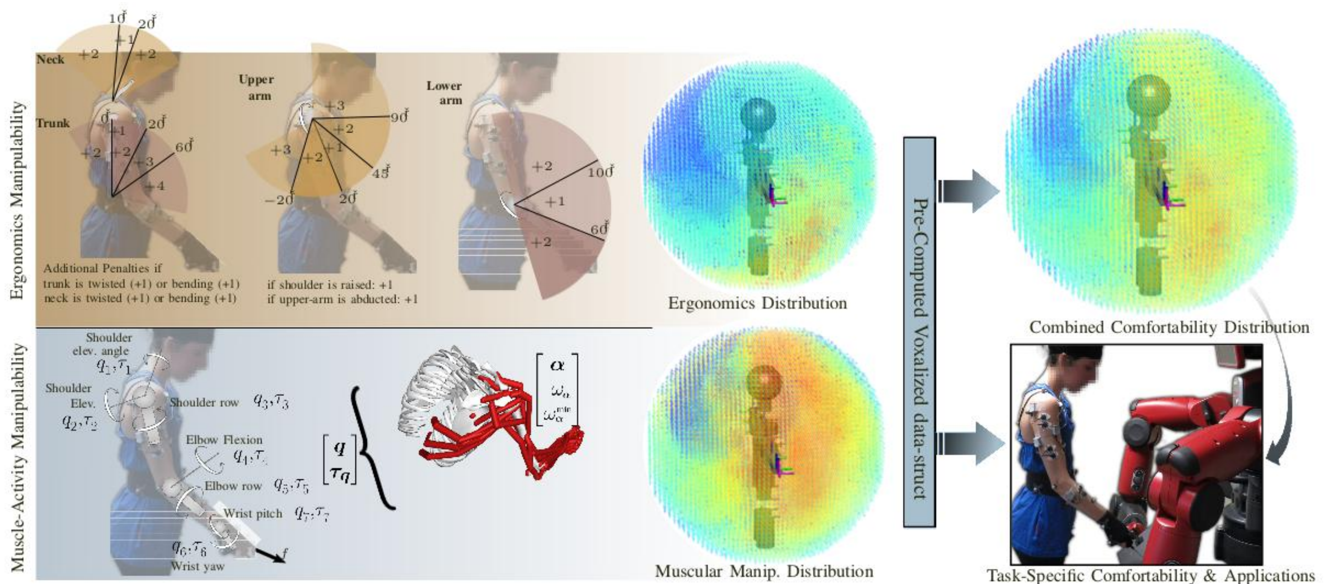


Figure: Outline for human manipulability assessment according to **rHuManManipulability**. In the top left figure, the Ergonomics comfortability is computed (with an example of a purely ergonomics manipulabilty) while in the bottom left describe the muscular-informed manipulability (MiM) analysis with one example of a purely MiM distribution. In the right, we show how both manipulabilities can be used to build a precomputed dataset (voxalized or not) and that can be used to build a combined comfortability distribution or applied to task-specific application.

Output

rHuManManipulability object with RapidHuman-Manipulability Assessment

Constructor Inputs

- **humanModel** from the class rHuManModel

All following inputs are optional and follow the struct: “input”,data

- **opensim',logical**
  - [NOT-AVAILABLE IN THIS VERSION] Assessment using opensim connection directly from matlab.
- **'localdata',char:**
  - DIR (absolute or relative) with relative or absolute path for dataset containing muscleMaxForce.mat and fct_getmuscleActivation.m functions (available within software)

Examples - Constructor

```
comfManip = rHuManManipulability(rhuman,'localdata','./OpenSimData')
comfManip = rHuManModel(rhuman,'localdata','./OpenSimData','verbose', true)
```

Variables

- **rhuman** humanModel from the class rHuManModel
- **datasetSize** Number (N) of inputs in the dataset
- **joints** double(7,N) of N joint positions. For the human model see classdef rHuManModel
- **pos** double(3,N) of N task-space positions for the user hand
- **rot** double(4,N) of N task-space orientation (quaternion) for the user hand
- **ergoManip** double(1,N) of ergonomics manipulability (based on RULA or REBA scores)
- **muscInfoManip** double(F,N) of N muscular-informed manipulability (MiM) where) F=1 for general MiM and F = number of wrenches(force/torques)/accelerations for the augmented MiM (default: 58)
- **penalty_selfCol** double (F,N) of self-collision penalty values for each N entry and F force/acceleration directions. Penalties defined at classdef rHuManModel
- **comfortIndex** Resulting comfort index (only available when calling reshape functions)

Both muscInfoManip and penalty_selfCol size will be updated according to calls to buildComfortability or buildAugmentedComfortability functions among other functions

- **voxConfig** Voxalized data configuration (Cartesian distance (5cm default], rotationDivisions [10 default], maxMuscle-Transm Rate and MaxMuscle-Inform-Manip. to be defined)
- **configDataset** Configuration for curr. dataset (includes forces and if they are defined in wrist or task-space)
- **verbose** Default: false;

Methods

build_AugmentedComfortDataset

[dataset] = build_AugmentedComfortDataset(N, [options]);
=> Builds the comfortability data-struct with N entries. N>100.
Stores data-struct in object-class and returns data-struct (optionally)
=> Additionally: accepts the same inputs from (setForces4Manipulability)
and 'default' to use the set of default forces (58)

build_TSComfortability

[dataset] = build_TSComfortability(N, force, kMuscle, [options])
=> Builds the task-specific comfortability manipulability distribution with N entries. N>100.
Stores data-struct in object-class and returns data-struct (optionally)
'force' is the task-specific force (or acceleration) direction.
'kMuscle' is the gain between [0,1] to weight muscular (1) and ergonomics (0)
=> Optional: 'wrist',logical : If forces/torques are defined in the end-effector (not in DEFAULT: task-space). Default=false.

compute_gainMuscle

[gain_Muscle] = compute_gainMuscle(Speed,Intensity,Repetitive)
=> Compute muscular-gain for comfortability. Ranges from [1]: only muscle to [0] only ergonomics

=> Inputs: Speed - Ranges between [0,1] (for very slow to very fast task executions)
Intensity- Intensity ranging between expected minimum and maximum task-space forces [0,1]
Repetitive - Repetitive has 3 modes: 0 (unique task, 0.5: repetes from time to time, 1 very repetitive (more than 4x per minute).

get_ComfortCost

[comfortCost, MuscleTransRate, RULA, penalties] = get_ComfortCost(joints, forcevector, kMuscle, maxTransRate, options)

=> Function compute comfort cost from both ergonomics and muscular (according to KMuscle).
Ranges from [0 -> 1] (from uncomfortable => max. comfortable)
MuscleTransRate (normalized). RULA score (least the better).
Returns -1 if self-collision is detected (and if self-collision detected is enabled [default])

Input: Joints (double7,1); forcevector (either double(6,N) ou double(3,N);
kMuscle: Gain between [0,1] to weight muscular (1) and ergonomics (0)
maxTransRate: Max Transmission Rate (vector) per force analyzed.
IF NOT DEFINED: Algorithm will search for an approximate value (increasing time)
[Optional:Inputs]: 'wrist',logical
If forces/torques are defined in the end-effector (not in DEFAULT: task-space). Default=false.
'selfCollision',logical : Default: True : Add selfCollision penalties to the final cost.

get_muscularCost

[muscleActEffort, MuscleTransRate, musclesAct, penalties] = get_muscularCost(joints, forcevector, options)

=> Returns: muscular-activity effort
=> denotes the sqrt(alpha^T*alpha) where alpha = muscle-activity vector)
muscular-transmission-rate => 1/max(alpha)
muscle-act-vector => alpha = double (50,1) with muscle-activity values
penalties => Self-collision penalties vector (for each force)
Input: Joints (double7,1); forcevector (either double(6,N) ou double(3,N);
[Optional:Inputs]: 'wrist',logical : If forces/torques are defined in the end-effector (not in DEFAULT: task-space). Default=false.

get_RULA

RULA = get_RULA(joints); Returns RULA score[4]
RULA = get_RULA(joints, 'shoulderRaised',true)
=> [optional: Extra pontuation for RULA (if shoulder is raised), see referenced paper for further details
= get_RULA(joints, 'shoulderPos',[0.10 0 1.3])
=> [optional: Extra pontuation for RULA (pos of shoulder to check if it is raised)]
= get_RULA(joints, 'armSupported', true)
=> [optional: Extra pontuation for RULA (if arm is supported), see referenced paper for further details]
= get_RULA(joints, 'repetitive', true)
=> [optional: Extra pontuation for RULA (if task is repetitive), see referenced paper for further details]

---

[4] RULA score computed according to McAtamney, L., & Hignett, S. (2004). Rapid Entire Body Assessment. Handbook of Human Factors and Ergonomics Methods, 31, 8-1-8–11. https://doi.org/10.1201/9780203489925.ch8

= fget_RULA(joints, 'heldstatic', true)
=> [optional: Extra pontuation for RULA (if task is held for long), see referenced paper for further details]
= get_RULA(joints, 'abrupt', true)
=> [optional: Extra pontuation for RULA (if task is abrupt), see referenced paper for further details]
= get_RULA(joints, 'heavyload', true)
=> [optional: Extra pontuation for RULA (if task involves heavy load (e.g., tool is heavy)), see referenced paper for further details]

get_TSComfortDataset

database = get_TSComfortDataset(force, options)
=> Function outputs a task-specific comfortability dataset from augmented data-struct with muscular and ergonomics data
=> 'force' is the task-specific force (or acceleration) direction.
=> [Optional:Input]: 'wrist',logical : If force/torque is defined in the end-effector (not in DEFAULT: task-space). Default=false.
=> [Optional:Input]: 'external',augmentedDataStruct : Explores an external datastruct instead of the one in the object itself [default].
External datastruct must contain fields: {size,joints,pos,rot,ergoManip,muscInfoManip,penalty_selfCol,configDataset}

handleRula2ErgoIndex

ergoIndex = handleRula2ErgoIndex(RULA)
=> Transform RULA to Ergonomic Index (normalized values)

handleErgoIndex2RULA

RULA = handleErgoIndex2RULA(ergoIndex) => Transform Ergonomic Index to RULA

plot_comfortability

plot_comfortability(comfDataDist,options) => plots a human model and scatter plot showing comfort values.

reshape_Comfortability

comfDist = reshape_Comfortability(kMuscle, options)

Function reshapes (and outputs) a general comfortability distribution from exsiting augmented data-struct with muscular and ergonomics data
=> **KMuscle**: Gain between [0,1] to weight muscular (1) and ergonomics (0)
comfortability integrates muscular data and Ergonomics according KMuscle.
=> [Optional:Input]: **'external'**,**augmentedDataStruct** : Explores an external datastruct instead of the one in the object itself [default].
External datastruct must contain fields: {size,joints,pos,rot,ergoManip,muscInfoManip,penalty_selfCol}

reshape_TSComfortability

comfDist = reshape_Comfortability(force, kMuscle, options)

Function reshapes (and outputs) a task-specific comfortability distribution from exsiting augmented data-struct with muscular and ergonomics data

=> **'force'** is the task-specific force (or acceleration) direction.

=> [Optional:Inputs]: **'wrist',logical**

: If force/torque is defined in the end-effector (not in DEFAULT: task-space). Default=false.

=> Other inputs follow the same from (reshape_Comfortability)

setForces4Manipulability

setForces4Manipulability()

=> Updates configuration forces to be used on comfortability data-struct construction. (default 58 wrenches).

setForces4Manipulability('length')

=> defines size of force-vector (if below 52, all wrenches above 6 will be make random). Min: 6 (if below, use specific 'forces','torques','wrenches' entries).

setForces4Manipulability('forces',forcevec)

=> [forcevec:double(3,N)] -> defines set of N forces for augmented manipulability assessment (This makes entry 'lenght' void)

setForces4Manipulability('torques',torquevec)

=> [torquevec:double(3,N)] -> defines set of N torques for augmented manipulability assessment (This makes entry 'lenght' void)

setForces4Manipulability('forces',forcevec, 'torques',torquevec)

=> Defines both force and torques for assessment.

setForces4Manipulability('wrenches',wrenchvec)

=> [wrenchvec:double(6,N)] -> defines set of N wrenches=[torque;force] for augmented manipulability assessment (This makes previous entries void)

setForces4Manipulability('wrist',true)

=> Defines reference frame of the forces to be in-hand (DEFAULT: false which implies forces defined in task-space) Alternative names: 'wristframe','end','endeffector'. Can be combined with previous entries (e.g., setForces4Manipulability('endeffector',true,'forces',forcevec)

rHuManManipulability / build_AugmentedComfortDataset

Build a Comfortability Dataset with Muscular-Informed Manip and Ergonomics

Inputs

datasize (double) : defines size of the datastruct (minimum is 100)

Inputs [Optional]

- **'default'** : When adding the string 'default', forces will be set for default mode in (58 forces)

-----------------------[Optional: INPUTs]:[Format: String followed by values]

- **'forces',double(3,N)** : defines set of N forces for augmented manipulability assessment (This makes entry 'lenght' void)
- **'torques',double(3,N)** : defines set of N torques for augmented manipulability assessment (This makes entry 'lenght' void)
- **'wrenches',double(6,N)** : defines set of N wrenches [torque;force] for augmented manipulability assessment (This makes entries 'lenght','forces','torques' void)

- **'wrist',logical** : If forces/torques are defined in the end-effector (DEFAULT: task-space). Default=false.
- **'wristframe',logical** : same as above. Default=false.
- **'end',logical** : same as above. Default=false.
- **'endeffector',logical** : same as above. Default=false.

Outputs

[Optional: datastruct] returns the a datastruct with joints, pos, rot, ergonomics, muscular-informed manip, and penalties for human workspace.
Method also updates variables within the object (thus, no need to safe output if it is not going to be used)

Usage

```
build_AugmentedComfortDataset(600000);
build_AugmentedComfortDataset(300000,'forces',randn(3,30),'torques',randn(3,10));
build_AugmentedComfortDataset(600000,'wrenches',randn(6,30));
build_AugmentedComfortDataset(1200000,'forces',randn(3,6),'wrist');
```

rHuManManipulability / build_TSComfortability

Build a Task-Specific Comfortability Dataset with Muscular-Informed Manip and Ergonomics
It builds the task-specific comfortability manipulability distribution.
See referenced paper for further details

Inputs

datasize (double) : defines size of the datastruct (minimum is 100)
force (double(6,1) OR double(3,1)) : Defines the task-specific force for analysis.
It accepts a Double(6,1) - for a wrench [torque;force]
OR Double(3,1) - for a wrench [0;force] with only a force.

kMuscle (double) : Gain between [0,1] to weight muscular (1) and ergonomics (0)
Example 0.5 (equal weights), while 1 implies only
muscular-assessment [Default = 0.5]

Inputs [Optional]

-----------------------[Optional: INPUTs]:[Format: String followed by values]

- **'wrist',logical** : If forces/torques are defined in the end-effector (DEFAULT: task-space). Default=false.
- **'wristframe',logical** : same as above. Default=false.
- **'end',logical** : same as above. Default=false.
- **'endeffector',logical** : same as above. Default=false.

Outputs

[Optional: datastruct] returns the a datastruct with joints, pos, rot, ergonomics, muscular-informed manip, and penalties for human workspace.
Method also updates variables within the object (thus, no need to safe output if it is not going to be used)

Usage

```
build_TSComfortability(300000,randn(6,1));
build_TSComfortability(600000,randn(3,1));
build_TSComfortability(600000,randn(3,1),'wrist');
```

rHuManManipulability / compute_gainMuscle

returns gain for comfortability (ranging from [1]: only muscle to [0] only ergonomics)
Compute gains for computing comfortability.
Returns gain for muscle (assuming gain for ergonomics is convex (1-gain_muscle)
Ranges from [1]: only muscle to [0] only ergonomics

Inputs

1. **Speed** - Ranges between [0,1] (for very slow to very fast task executions)
2. **Intensity** - Intensity ranging between expected minimum and maximum task-space forces [0,1]
3. **Repetitive** - Repetitive has 3 modes: 0 (unique task, 0.5: repetes from time to time, 1 very repetitive (more than 4x per minute).

Outputs

gain_Muscle - Gain for muscle (assuming gain for ergonomics is convex (1-gain_muscle)
Ranges from [1]: only muscle to [0] only ergonomics

Usage

```
gain_Muscle = compute_costs(0.5, 0.75, 0)
gain_Muscle = compute_costs(0.65, 0.25, 1)
```

rHuManManipulability / get_ComfortCost

Retrieves comfortCost (and: [comfortCost, MuscleTransRate, RULA, penalties])
This function returns the combined ergonomics (RULA) and muscular assessment

* Muscular assessment from the kinematics & biomechanics (see Saul et al.[5])
* RULA/REBA points are computed according to RULA[6]

Note that z:height, x:sideways (right shoulder out), y:face-front
for the hand: z:out of fingers, x:palm down, y:thumbs up

Inputs

joints (joints): A 7x1 double with joint values in rad (according to Saul's model)
Default joints limits (can be changed):
Joint limits (lower): [-90 0 -90 0 -90 -15 -75]'*(pi/180)
Joint limits (upper): [130 180 +45 145 +90 +25 +75]'*(pi/180)

forcevector (double 6,N) with N >= 1
Defines the unique force or a set of forces to be analyzed.
If N>1 (multiple-forces), then the output is also a [vector, vector, matrix]
(double 3,N) : similar but with wrench defined by: [0;forcevector]

kMuscle (double) : Gain between [0,1] to weight muscular (1) and ergonomics (0)
Example 0.5 (equal weights), while 1 implies only muscular-assessment

Inputs [Optional]

- **maxTransRate (double(N):** Maximum Transmission Rate per force analyzed
(if N=1, the same value will be applied to all forces
IF NOT DEFINED: Algorithm will search for an approximate value (increasing time)

------------------------[Optional: INPUTs]:[Format: String followed by values]

- **'wrist',logical** : If forces/torques are defined in the end-effector (DEFAULT: task-space). Default=false.
- **'wristframe',logical** : same as above. Default=false.
- **'end',logical** : same as above. Default=false.

---

[5] Saul et al., "Benchmarking of dynamic simulation predictions in two software platforms using an upper limb musculoskeletal model Benchmarkin…", CMBBE, (2015).

[6] McAtamney, L., & Hignett, S. (2004). Rapid Entire Body Assessment. Handbook of Human Factors and Ergonomics Methods, 31, 8-1-8-11. https://doi.org/10.1201/9780203489925.ch8

- **'endeffector',logical** : same as above. Default=false.

- **'selfCollision',logical** : Default: True : Add selfCollision penalties to the final cost.

Outputs

[comfortCost,   muscular-transmission-rate,   RULA_value]

comfortCost :   ranges from [0 -> 1] (from uncomfortable => max. comfortable)
Values will vary according to gain kappa (input)
muscular-transmission-rate :   1/max(alpha) where alpha = muscle-activity vector
RULA: RULA/REBA points. Returns error when 0.

Usage

[comfortCost, MuscleTransRate, RULA, penalties] = **get_ComfortCost**(joints, force, 1.0)
[comfortCost, MuscleTransRate, RULA, penalties] = **get_ComfortCost**(joints, force_vec, 0.5, 'wrist',true)
[comfortCost, MuscleTransRate, RULA, penalties] = **get_ComfortCost**(joints, force_vec, 0.5, 250)
[comfortCost, MuscleTransRate, RULA, penalties] = **get_ComfortCost**(joints, force_vec, 0.5, 'selfCollision',false)

rHuManManipulability / get_muscularCost

Retrieves Muscular-Cost : [muscleActEffort, MuscleTransRate, musclesAct-vec, penalties]
This function returns muscular assessment from the kinematics & biomechanics.
* Muscular assessment from the kinematics & biomechanics (see Saul et al.)

Note that z:height, x:sideways (right shoulder out), y:face-front
for the hand: z:out of fingers, x:palm down, y:thumbs up

Inputs

joints (joints): A 7x1 double with joint values in rad (according to Saul's model)
forcevector (double 6,N) with N >= 1
Defines the unique force or a set of forces to be analyzed.
If N>1 (multiple-forces), then the output is also a [vector, vector, matrix]

Inputs [Optional]

-----------------------[Optional: INPUTs]:[Format: String followed by values]

- **'wrist',logical** : If forces/torques are defined in the end-effector (DEFAULT: task-space). Default=false.
- **'wristframe',logical** : same as above. Default=false.
- **'end',logical** : same as above. Default=false.
- **'endeffector',logical** : same as above. Default=false.

Outputs

muscular-activity effort : ranges from [0 ->no effort to sqrt(ones(50,1)'*ones(50,1))]
denotes the sqrt(alpha^T*alpha) where alpha = muscle-activity vector
muscular-transmission-rate : 1/max(alpha)
muscle-act-vector : double (50,1) with muscle-activity values
penalties : self-collision penalties vector (for each force)

Usage

[muscleActEffort, MuscleTransRate, musclesAct, penalties] = get_muscularCost(joints, force)
[muscleActEffort, MuscleTransRate, musclesAct, penalties] = get_muscularCost(joints, force_vec, 'wrist',true)

rHuManManipulability / get_RULA

Retrieves RULA / REBA Scores

This function returns RULA (or REBA) assessment (points) taking the kinematics from (fctLoadHuman_opensim_simplified) according to Saul et al. RULA points are computed according to McAtamney, L., & Hignett, S. (2004). Rapid Entire Body Assessment. Handbook of Human Factors and Ergonomics Methods, 31, 8-1-8–11. https://doi.org/10.1201/9780203489925.ch8

Note that z:height, x:sideways (right shoulder out), y:face-front
for the hand: z:out of fingers, x:palm down, y:thumbs up

Inputs

joints (joints): A 7x1 double with joint values in rad (according to Saul's model)

Inputs [Optional]

-----------------------[Optional: INPUTs]:[Format: String followed by values]

- **'shoulderRaised',boolean** : If Shoulder is raised
- **'armSupported',boolean** : If arm is supported
- **'shoulderPos',vec3** **:** Used to check if shoulder is raised (together with shoulderBase)

-----------------------[Activities score]

- **'repetitive',boolean** : [Default: false] Extra pontuation for RULA (if task is repetitive), see referenced paper for further details
- **'heldstatic',boolean** : [Default: false] Extra pontuation for RULA (if task is held for long)
- **'abrupt',boolean** : [Default: false] Extra pontuation for RULA (if task is abrupt)
- **'heavyload',boolean** : [Default: false] Extra pontuation for RULA (if task involves heavy load (e.g., tool is heavy))

[deprecated] * 'shoulderBase',vec3 : Used to check if shoulder is raised
[deprecated] * 'handCrossLine',boolean : If working across midline of the body (chest)
[deprecated] * 'handOutSideBody',boolean : If arm out to side of body (check wrt to elbow)
[deprecated] * 'elbowPos',vec3 : Used to check if wrist is crossing or outside body (together with wristPos and shoulderPos(OR)shoulderBase)
[deprecated] * 'wristPos',vec3 : Used to check if wrist is crossing or outside body (together with elbowPos and shoulderPos(OR)shoulderBase)

[deprecated] * 'kine',DQ_kinematics : Passing the DQ_Kinematics of the human upper limb allow the function to compute (1) the shoulderBase; (2) elbowPos; and (3) wristPos which in turn allow the function to compute (a) if shoulderRaised (with shoulderPos); (b) if handCrossLine or handOutSideBody

[deprecated] * 'verbose',boolean : Print all points and steps

Outputs

RULA: RULA points (Results between 1-13 ==> Best=1, Worst=13). Returns error when 0.

Usage

```
RULA_SCORE = fct_get_RULA(joints)
RULA_SCORE = fct_get_RULA(joints, 'shoulderRaised',true)
RULA_SCORE = fct_get_RULA(joints, 'shoulderPos',[0.10 0 1.3], 'armSupported', true)

[ deprecated ] * fct_get_RULA(joints, 'handOutSideBody',true, 'shoulderBase',[0 0 1.350],'shoulderPos',[0 0 1.375])    % which is true for shoulderRaised
[ deprecated ] * fct_get_RULA(joints, 'elbowPos',[0.15 0 0],'wristPos',[0.16 0 0])                                       % which is true for handOutSideBody
[ deprecated ] * fct_get_RULA(joints, 'wristPos',[-0.30 0.10 1.3], 'shoulderPos',[-0.09 0 1.3])                % which is true for handCrossLine
[ deprecated ] * fct_get_RULA(joints, 'kine',humankine, 'shoulderPos',[0.10 0 1.3])     % Using the DQ_kinematics variable to compute
                                                                                          % (1) the shoulderBase; (2) elbowPos; and (3) wristPos
```

rHuManManipulability / get_TSComfortDataset

Returns a Task-Specific Comfortability database from an Augmented comfortability dataset. It builds the Task-Specific comfortability datastruct from an existing augmented data-struct with muscular and ergonomics data (those are not to be updated).

See referenced paper for further details

Inputs

forcevector (double 6,1) or (double(3,1)

Defines the task-specific force for analysis.

It accepts a Double(6,1) - for a wrench [torque;force]

OR Double(3,1) - for a wrench [0;force] with only a force.

Inputs [Optional]

-----------------------[Optional: INPUTs]:[Format: String followed by values]

'external',datastruct : Explores an external datastruct instead of the one in the object itself. [default]
External datastruct must contain:
{datasetSize,joints,pos,rot,ergoManip,muscInfoManip,penalty_selfCol}

- **'wrist',logical** : If forces/torques are defined in the end-effector (DEFAULT: task-space). Default=false.
- **'wristframe',logical** : same as above. Default=false.
- **'end',logical** : same as above. Default=false.
- **'endeffector',logical** : same as above. Default=false.

Outputs

[Optional: datastruct] returns the a datastruct with joints, pos, rot, ergonomics, muscular-informed manip, and penalties for human workspace.

Method also updates variables within the object (thus, no need to safe output if it is not going to be used)

Usage

tsDatabase = **get_TSComfortDataset**(randn(3,1));

Returns a comfortability datastruct from data in the obj (expects prior use of build_AugmentedComfortDataset())

tsDatabase = **get_TSComfortDataset**(randn(6,1), 'external',extDataset);

Returns a comfortability datastruct from extDataset

tsDatabase = **get_TSComfortDataset**(force,'wrist',true);

Similar assessment but with force defined in end-effector frame

rHuManManipulability / handleErgoIndex2RULA

return RULA Score from Ergonomic Index. Compute RULA SCORE from Ergonomic Index

Inputs

ergoIndex - double with normalized Ergonomic Index [0,1]

Outputs

RULA SCORE - double (integer)

Usage

RULA = **handleErgoIndex2RULA**(ergoIndex)

rHuManManipulability / handleRula2ErgoIndex

return Ergonomic cost from RULA Score
Compute Ergonomic Index from RULA SCORE

Inputs

RULA SCORE - double

Outputs

ergoIndex - double with normalized Ergonomic Index [0,1]

Usage

ergoIndex = **handleRula2ErgoIndex**(RULA)

Inputs

datasize (double) : defines size of the datastruct (minimum is 100)

rHuManManipulability / plot_comfortability

Plots a human model with the comfortability distribution
This method plots a human model and scatter plot showing comfort values.

Inputs

'external',ComfDataDist
Explores an external comfortability distribution. Different from dataset, the structu must have a 'comfortIndex'
External datastruct must contain: {pos, comfortIndex}

Inputs [Optional]

-----------------------[Optional: INPUTs]:[Format: String followed by values]
*** 'animation',logical** : Defines if animation is on/off
*** 'animationDelay',double** : Animation time (between pauses). If ('animationDelay',0) then, it pauses until user press a key [Default: 0.1].

*** 'savefig',char** : If saving figure, please specifiy the address

*** 'section',logical** : If plot will be presented in transversal sections or full (false=default).
*** 'section_num',double** : Number os sections (default=5).
*** 'section_axis',double** : Defines axis for section x=1; y=2; z=3; (default=3).

*** 'noModel',logical** : False; % Set this to true if you do not wish to plot human model

Outputs

-

Usage

plot_comfortability('animation',true)
plot_comfortability('section',true)
plot_comfortability('noModel',logical)

RHuManManipulability / reshape_Comfortability

Reshape an Augmented Comfortability Dataset into a comfortability distribution
It builds the comfortability general manipulability from an existing augmented data-struct with muscular and ergonomics data (those are not to be updated).
See referenced paper for further details

Inputs

kMuscle (double) : Gain between [0,1] to weight muscular (1) and ergonomics (0)
Example 0.5 (equal weights), while 1 implies only
muscular-assessment [Default = 0.5]

Inputs [Optional]

------------------------[Optional: INPUTs]:[Format: String followed by values]
'external',datastruct : Explores an external datastruct instead of the one in the object itself. [default]
External datastruct must contain:
{datasetSize,joints,pos,rot,ergoManip,muscInfoManip,penalty_selfCol}

Outputs

datastruct: returns the comfortability distribution datastruct with joints, pos, rot, ergonomics, muscular-informed manip, and penalties for human workspace.

Usage

comfDistribution = **reshape_Comfortability**(0.5);
Returns a comfortability distribution datastruct from data in the obj
(expects prior use of build_AugmentedComfortDataset())
comfDistribution = **reshape_Comfortability**(0.75, 'external',extDataset);
Returns a comfortability distribution datastruct from extDataset

rHuManManipulability / reshape_TSComfortability

Reshape an Augmented Comfortability Dataset into a Task-Specific comfortability distribution. It builds the Task-Specific comfortability distribution from an existing augmented data-struct with muscular and ergonomics data (those are not to be updated).
See referenced paper for further details

Inputs

forcevector (double 6,1) or (double(3,1)
Defines the task-specific force for analysis.
It accepts a Double(6,1) - for a wrench [torque;force]
OR Double(3,1) - for a wrench [0;force] with only a force.
kMuscle (double) : Gain between [0,1] to weight muscular (1) and ergonomics (0)
Example 0.5 (equal weights), while 1 implies only
muscular-assessment [Default = 0.5]

Inputs [Optional]

-----------------------[Optional: INPUTs]:[Format: String followed by values]
'external',datastruct : Explores an external datastruct instead of the one in the object itself. [default]
External datastruct must contain:
{datasetSize,joints,pos,rot,ergoManip,muscInfoManip,penalty_selfCol}

- **'wrist',logical** : If forces/torques are defined in the end-effector (DEFAULT: task-space). Default=false.
- **'wristframe',logical** : same as above. Default=false.
- **'end',logical** : same as above. Default=false.
- **'endeffector',logical** : same as above. Default=false.

Outputs

datastruct: returns the task-specific comfortability distribution datastruct with joints, pos, rot, ergonomics, muscular-informed manip, and penalties for human workspace.

Usage

tscomfDist = reshape_TSComfortability(randn(3,1),0.5);
Returns a comfortability distribution datastruct from data in the obj
(expects prior use of build_AugmentedComfortDataset())
tscomfDist = reshape_TSComfortability(randn(6,1),0.75, 'external',extDataset);
Returns a comfortability distribution datastruct from extDataset
tscomfDist = reshape_TSComfortability(force,0.5, 'wrist',true);
Similar assessment but with force defined in end-effector frame

rHuManManipulability / setForces4Manipulability

Set ForceVector for Muscular Comfortability Analysis. It updates the set of forces used when building (augmented) muscular-informed manipulability

Inputs [Optional]

- **'default'** : When adding the string 'default', forces will be set for default mode in (58 forces)

-----------------------[Optional: INPUTs]:[Format: String followed by values]

- **'length',double** : defines size of forces for assessment (if below 52, all wrenches above 6 will be make random). Min: 6 (if below, use specific 'forces','torques','wrenches' entries)
- **'forces',double(3,N)** : defines set of N forces for augmented manipulability assessment (This makes entry 'lenght' void)
- **'torques',double(3,N)** : defines set of N torques for augmented manipulability assessment (This makes entry 'lenght' void)
- **'wrenches',double(6,N)** : defines set of N wrenches [torque;force] for augmented manipulability assessment (This makes entries 'lenght','forces','torques' void)

- **'wrist',logical** : If forces/torques are defined in the end-effector (DEFAULT: task-space). Default=false.
- **'wristframe',logical** : same as above. Default=false.
- **'end',logical** : same as above. Default=false.
- **'endeffector',logical** : same as above. Default=false.

Outputs

- VOID: Update list for forcevector - class variable

Usage

```
setForces4Manipulability();
setForces4Manipulability('length',30);
setForces4Manipulability('forces',randn(3,30),'torques',randn(3,10));
setForces4Manipulability('wrenches',randn(6,30));
setForces4Manipulability('forces',randn(3,6),'wrist');
```

rHuMAn for USERS

The following figure presents a flowchart with the main steps processes for computing comfortability information via the library.

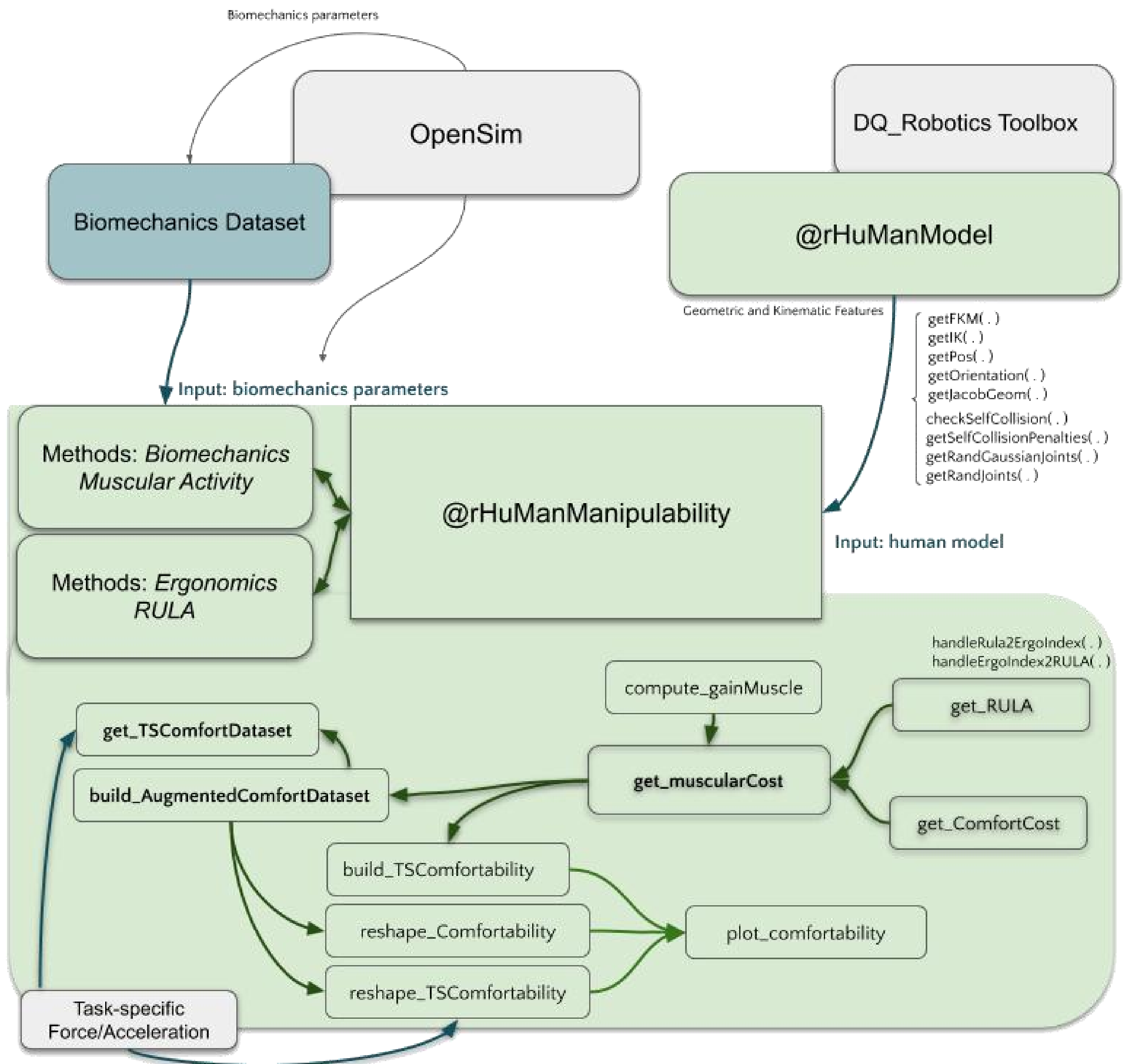


Figure: Outline for the library structure and how to explore methods and variables available within the rHuMAn tool. Both OpenSim and DQ_Robotics are libraries dependencies needed to run the software, yet rHuMAn can also use previously constructed datasets from OpenSim Models. Classes and most relevant methods available in the tool are depicted in green with connection between methods.

The rHuMAn (Rapid Human-Manipulability Assessment) tool provides the interface to human kinematics from DQ_Robotics toolbox with relevant functions and additional features specific for biomechanics analysis. It already embeds the kinematics model commonly used in biomechanics (model used in OpenSim Upper-Body model). This kinematics is then fully integrated with human actuation according to human kinesiology. In other words, the kinematics is integrated with tools to process muscular forces and activity based on biomechanics parameters that maps task-space accelerations and forces to muscle-space (e.g., muscle-length, muscle maximum isometric force, moment-arm between muscle and joints, etc). These features are extracted from a large database (available within this software) acquired from opensim experiments. In additional to both kinematics and biomechanics, rHuMAn also outputs human postural ergonomics. This postural analysis technique draw a quantitative measure for ergonomics (mostly focused and applied to industrial workflows) from experiments and evaluation from experts—ergonomists and physiotherapists among others.

The rHuMAn provides the tools to analyze each feature (kinematics, biomechanics, and ergonomics) isolated, to create a specific muscle or ergonomics manipulability datastruct or distribution, to visualize them, and to shape those to task-specific applications.

Simple Working Example

The following simple working example illustrates on of the main applications of the rHuMAn AI tool that is to build a workspace comfortability distribution from an augmented dataset.
To this aim, we first build a human model using rHuManModel, then load an object for rHuManManipulability assessment. The detailed code with comments is presented below

Simple Working Example : rHuMAn Library

Contents

- Constructing human model from rHuManModel
- Constructing human Manipulability Object from rHuManManipulability
- Building the augmented datastruct
- Building a General Comfortability Distribution
- Building a Task-Specific Comfortability Distribution

Constructing human model from rHuManModel

First, we construct a human model using rHuManModel with height (shoulder!) of 1.35 m. Enabling verbose for illustration.

```
rhuman = rHuManModel('shoulderHeight',1.35,'verbose',true);
```

```
**************************************************
****** Loading human model...
*** kinematics:
        upper-arm: 0.302
        fore-arm: 0.2795
        point of force at hand: 0.05
        shoulder-base position (xyz): [0           0        1.35]
        shoulder-base orientation (quat): [1  0  0  0]
*** Body Geometry for self collision
        Body 3D-box: x \in [-0.3145     -0.0355], y \in [-0.065       0.065], z \in [0.75       1.375]
        Head 3D-ball with radius: [0.14] centered at [-0.2           0        0.15]
        Upper-arm Cylinder-Radius: [0.035] with [4] discrete equally spaced points.
        Fore-arm Cylinder-Radius: [0.02] with [3] discrete equally spaced points.
*** Definitions
        Shoulder orientation: (+x) lateral towards right, (+y) front, (+z) upwards
        Wrist orientation: (+x) Palm-in, (+y) Thumbs-ub, (+z) defined from wrist to fingers
```

To visualize human model, display its properties. Note that 'kine' depicts the human kinematics, and 'kineconfig' the general kinematics and geometric configurations (e.g., bounding box defining human body and ball defining human head that are used for fast self-collision analysis, and functions as checkJointLim to verify if a prescribed joint is over the prescrived limits)

```
rhuman
```

```
rhuman =

  rHuManModel with properties:

           kine: [1×1 DQ_kinematics]
     kineconfig: [1×1 struct]
      hand2tool: [1×1 DQ]
    pointsInArm: []
```

To have a quick visualization method to human upper-limb (now, only avaiable for right-handed) You can define a joint configuration.

The following is one configuration with shoulder elevated 30o at 90o angle (facing in front of the person) with 40o of elbow flexion. You can also print in the screen its position and orientation (quaternion)

```
theta = [90; 30; 0; 40;   0; 0; 0]*pi/180;
rhuman.plot(theta);
view(-70,22);
axis([-0.6 0.6 -0.3 0.6 0.0 1.8]);

position = rhuman.getPos(theta)
orientation = rhuman.getOrientation(theta)
```

```
position =

   -0.0000
    0.4606
    0.9758


orientation =
```

```
    0.0000
    0.0000
   -0.8192
   -0.5736
```

Constructing human Manipulability Object from rHuManManipulability

We take the human model and the existing database located in ./OpenSimData folder

```
rhmanip = rHuManManipulability(rhuman,'localdata','./OpenSimData','verbose', true);
```

```
****** Human Manipulability Loaded.
*** Copying data to workspace and enabling functions for assessment.
*** muscleMaxForce already loaded in workspace with muscular-force database from opensim.
*** System configured to be used.
```

Building the augmented datastruct

Lets build the augmented datastruct with a set of forces (default 58) Input is simply the number of joints. But, to see more options check the help for build_AugmentedComfortDataset

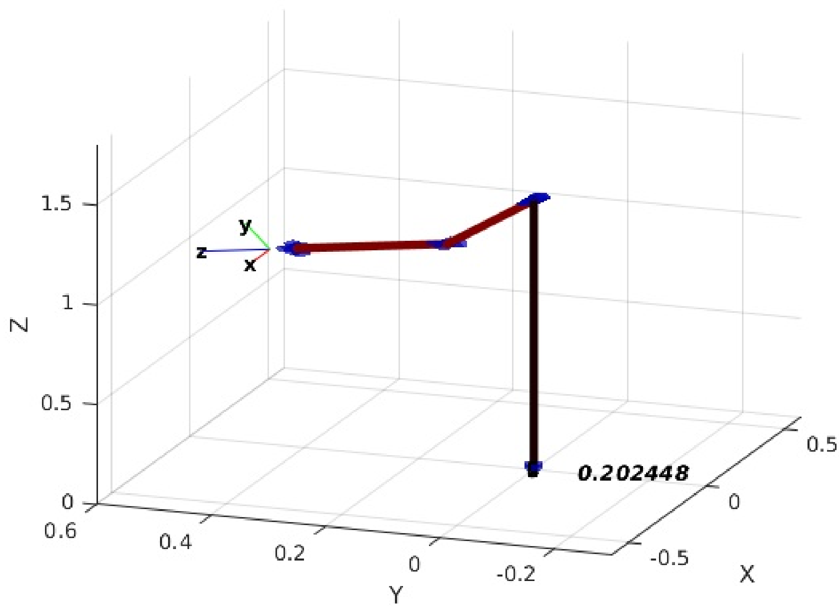
```
datastruct = rhmanip.build_AugmentedComfortDataset(100000);
```

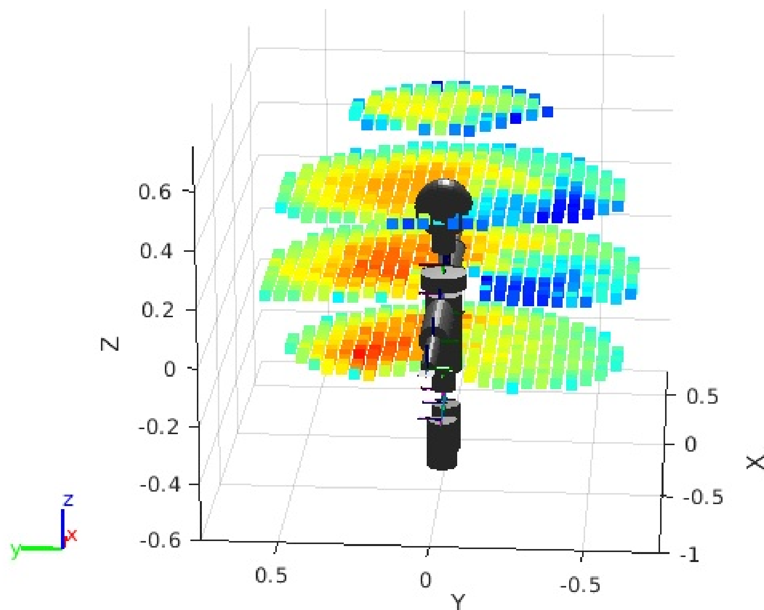
Building a General Comfortability Distribution

Now we reshape the datastruct to have a comfortability distribution.

```
genManip = rhmanip.reshape_Comfortability('external',datastruct);

% Now plot the comfortability to observe the best configurations considering all possible forces (in all directions)
% and all accelerations. That is, the distribution of comfortability (assuming the worst possible task) along human workspace
rhmanip.plot_comfortability(genManip,'section',true)
```

```
*** Using external augmented datastruct for shaping a comfortability distribution analysis.

*** Maximum muscular-informed manipulability (MiM) index computed in the workspace is: 28.0438

*** shaping the comfortability distribution...
*** Percentage completed: 10% at 7.7799 sec
*** Percentage completed: 20% at 17.5311 sec
*** Percentage completed: 30% at 28.295 sec
*** Percentage completed: 40% at 39.7008 sec
*** Percentage completed: 50% at 51.3379 sec
*** Percentage completed: 60% at 62.7263 sec
*** Percentage completed: 70% at 73.4877 sec
*** Percentage completed: 80% at 83.5182 sec
*** Percentage completed: 90% at 92.8396 sec
*** Percentage completed: 100% at 101.2667 sec

ans =

  Axes (Primary) with properties:

             XLim: [-2 2]
             YLim: [-2 2]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Use GET to show all properties
```





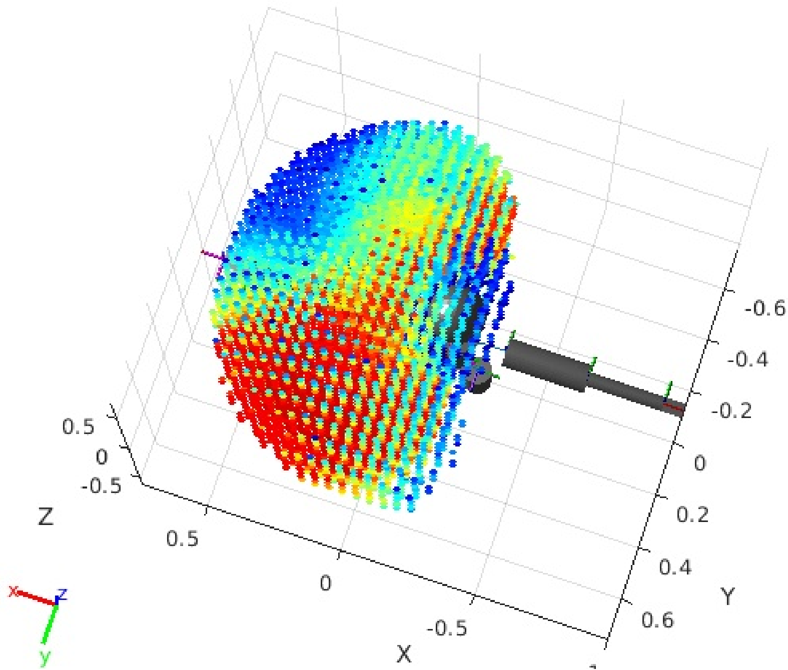
Building a Task-Specific Comfortability Distribution

Now we reshape the datastruct to have a task-specific comfortability distribution. Let's assume that a given task is defined by the following wrench oracceleration

```
force = [0; 5; 0;  10; 10; 0];
% In other words, a wrench with a torque in y axis and a diagonal force
% pointing 45o in between X and Y axis (that is, in front and outwards the person)
%
% Now, let's compute the gain between muscular-informed assessmetn and ergonomics.
task.speed     = 0.5;  % Medium speed
task.Intensity = 0.9;  % Highly intensive activity (see the forces)
task.repeat    = 0;    % Let' s assume that this is a 1 time task.
kMuscle = rhmanip.compute_gainMuscle(task.speed, task.Intensity, task.repeat)

% kMuscle must always return a value between [1]: only muscle to [0] only ergonomics
%
% Now we reshape the augmented datastruct to have a task-specific comfortability distribution.
tsManip = rhmanip.reshape_TSComfortability(force, kMuscle, 'external',datastruct);

% Now plot the comfortability to observe the best configurations considering the specific force
% That is, the distribution along workspace where the human can better apply such force/acceleration
rhmanip.plot_comfortability(tsManip)
```

```
kMuscle =

    0.8000

*** Normalizing input force.
```

```
*** Using external augmented datastruct for shaping a comfortability distribution analysis.

*** shaping the comfortability distribution...
*** Percentage completed: 10% at 7.5527 sec
*** Percentage completed: 20% at 15.8049 sec
*** Percentage completed: 30% at 24.554 sec
*** Percentage completed: 40% at 33.6427 sec
*** Percentage completed: 50% at 42.7891 sec
*** Percentage completed: 60% at 51.8905 sec
*** Percentage completed: 70% at 61.0451 sec
*** Percentage completed: 80% at 70.1853 sec
*** Percentage completed: 90% at 78.9115 sec
*** Percentage completed: 100% at 86.8048 sec

ans =

  Axes (Primary) with properties:

             XLim: [-2 2]
             YLim: [-2 2]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Use GET to show all properties
```



Top-view (Y positive = front of the person).

License

The following figure presents a flowchart with the main steps processes for computing comfortability information via the library.

The MIT License (MIT)

Copyright (c) 2020 rHuMAn Project

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND ONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

People

- Luis F C Figueredo <figueredo@ieee.org>, University of Leeds
- Mehmet Dogar <m.r.dogar@leeds.ac.uk>, University of Leeds
- Anthony Cohn <A.G.Cohn@leeds.ac.uk>, University of Leeds

We would like to grateful acknowledge the financial support of the EC under Horizon2020 AI4EU Project and Horizon 2020 Marie Sklodowska-Curie grant agreement No 795714.