SPART Documentation

Release 0.2.0

Josep Virgili-Llop

Feb 15, 2018

Contents

SPART is an open-source modeling and control toolkit for mobile-base robotic multibody systems with kinematic tree topologies (*i.e.*, open-loop multi-branched systems). SPART is MATLAB-based and ROS-compatible, allowing to prototype in simulation and deploy to hardware controllers for robotic systems.

Given a URDF description of a multibody system, SPART computes the system's:

- Kinematics – pose of the links and joints (*i.e.*, rotation matrices and position vectors).
- Differential kinematics – operational space velocities and accelerations, as well as the geometric Jacobians and their time derivatives.
- Dynamics – generalized inertia and convective inertia matrices.
- Forward/Inverse dynamics – solves both problems, including the floating-base case.

SPART supports symbolic computation and analytic expressions for all kinematic and dynamic quantities can be obtained.

CHAPTER 1

Contents

1.1 Installing SPART

First you need to download the SPART toolkit from the SPART Github repository:

```
git clone https://github.com/NPS-SRL/SPART.git
```

Once you have the SPART source code, navigate to the SPART root folder and execute the following command in the MATLAB command window:

```
SPART2path
```

This adds all the required SPART folders into your MATLAB path and saves it.

Note: In Linux, running MATLAB with root privileges may be required (*i.e.*, `sudo`) to permanently save the MATLAB path.

1.1.1 Dependencies

Some MATLAB toolboxes are required to fully exploit SPART:

- The Robotics System Toolbox is required to interface MATLAB with ROS.
- The Matlab Coder and the Simulink Coder are required to generate standalone ROS nodes or portable C/C++ code from SPART MATLAB code or Simulink models.
- The Symbolic Math Toolbox is required to obtain analytic expressions of the kinematic and dynamic quantities.
- The Simulink 3D Animation is useful to visualize the robotic systems.

1.2 SPART Tutorial – Introduction

This tutorial covers the basic functionality of SPART and introduces some concepts of multibody systems. The tutorial is structured in three sections:

- Introduction – Covers the nomenclature and conventions used by SPART.
- *SPART Tutorial – Robot Model* – Covers the URDF description of a multibody system and the SPART `robot` structure.
- *SPART Tutorial – Kinematics* – Covers the kinematics – positions, orientations, velocities, and accelerations – of the system.
- *SPART Tutorial – Dynamics* – Covers how to obtain the inertia matrices, equations of motion, and solve the forward/inverse dynamic problem.

The code in this tutorial can be found in `examples/URDF_Tutorial/URDF_Tutorial.m`.

Note: Before starting this tutorial make sure you have correctly installed and configured SPART. See *Installing SPART* for instructions.

1.2.1 Kinematic tree topology

A multibody system is defined as a collection of bodies coupled by massless **joints**. The bodies of the system – also known as **links** – are arranged in one of two basic types of kinematic chains:

- Kinematic trees, when the path between any pair of links is unique. This are also known as open-loop kinematic chains.
- Closed-loop kinematic chains, when the path between any pair of links is not unique.

SPART is only able to handle multibody systems composed of **rigid bodies** arranged in **kinematic trees**.

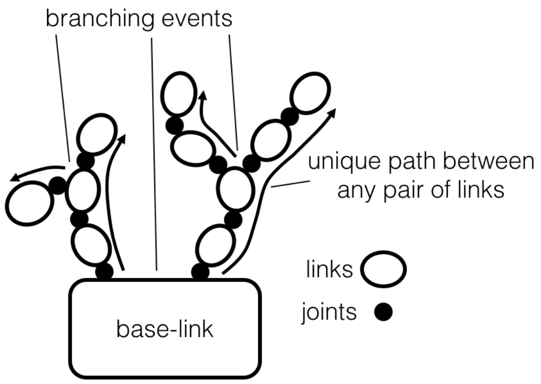

Fig. 1.1: Kinematic tree.

1.2.2 Joint/Link nomenclature and numbering scheme

A link is denoted by $\mathcal{L}_i$ and a joint by $\mathcal{J}_i$, with the number i used as a unique identifier of each link or joint. In SPART, the number i is used to access the results associated with a specific link or joint.

In a kinematic tree, one of the links is designated as the **base-link**, with $i = 0$ and $\mathcal{L}_0$. The base-link can be selected arbitrarily among all the links, yet an obvious choice usually exists.

A link $\mathcal{L}_i$ can be connected with an arbitrary number of other links via an equal number of joints. Only one of these other links lies within the path connecting $\mathcal{L}_i$ and the base-link $\mathcal{L}_0$. This *previous/upstream* link $\mathcal{L}_{i-1}$ is known as the *parent link* of $\mathcal{L}_i$ and the joint connecting these two links is $\mathcal{J}_i$. The rest of links directly connected to link $\mathcal{L}_i$ are *child links* $\mathcal{L}_{i+1}$. A *branching event* occurs when a link has multiple children.

In the regular number scheme used by SPART, each children link is given a higher number i than its parent, with the base link given the number $i = 0$. In a branching event multiple numbering options exist and they can be chosen arbitrarily among them. The notation $i + 1$ and $i - 1$ is here abused to denote the child and parent link or joint, even when they are not sequentially numbered. Additionally, the total number of joints n is also used to refer to the last joint and link of a branch. The last link of a branch is also commonly referred to as an **end-effector**.

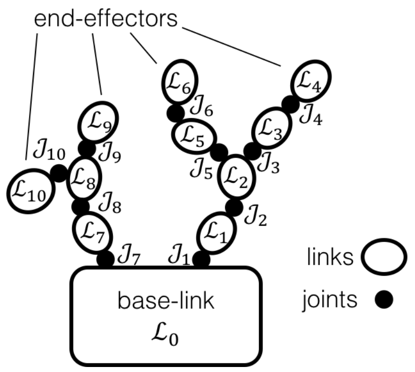

Fig. 1.2: Regular numbering scheme.

1.2.3 Joints types

Three type of joints primitives can be modeled with SPART:

- Fixed – a joint rigidly connecting the pair of links (zero degrees-of-freedom).
- Revolute – a joint that allows a rotation around a common rotation axis $\hat{e}_i$ (one degree-of-freedom).
- Prismatic – a joint that allows a translation along a common sliding axis $\hat{e}_i$ (one degree-of-freedom).

More complex joints (*e.g.*, planar, spherical, helical, . . .) can be constructed as a combination of these primitive joints connected by massless and dimensionless links.

Only revolute and prismatic joints are active joints, with the rotation or translation **displacement** denoted by q_i.

1.2.4 Cartesian Coordinate Systems (CCS)

Each link and joint has an associated Cartesian Coordinate System (CCS). The origin of the link CCS is located at the link's center-of-mass, and the origin of the joint CSS is located on the rotation/sliding axis. The orientations of the CCS are arbitrary.

1.2.5 Joint displacements

Another convention in SPART is that a displacement on a joint affects all the elements downstream, but it doesn't affect the orientation or position of that joint CCS.

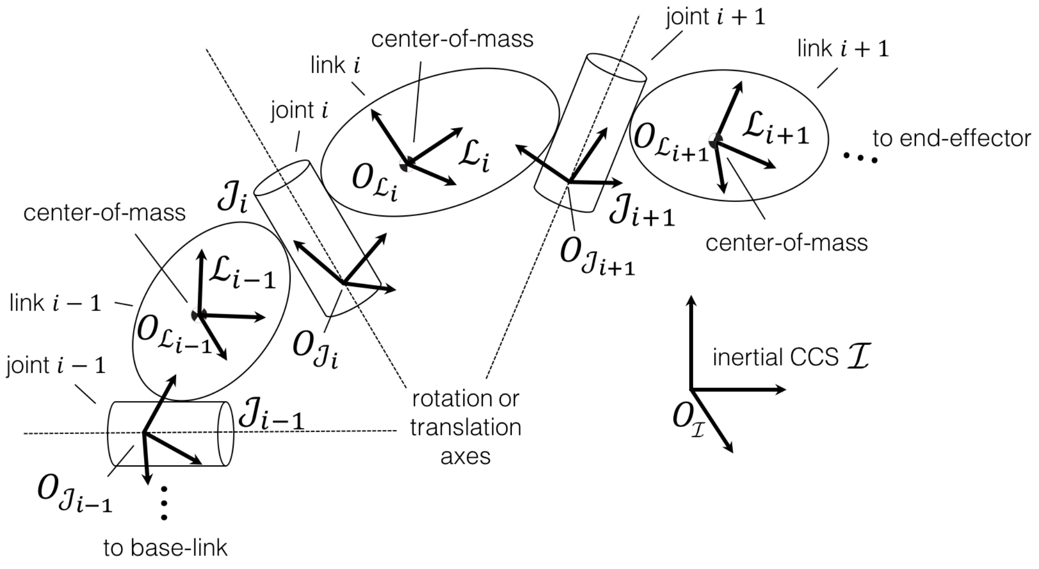

Fig. 1.3: Cartesian Coordinate Systems.

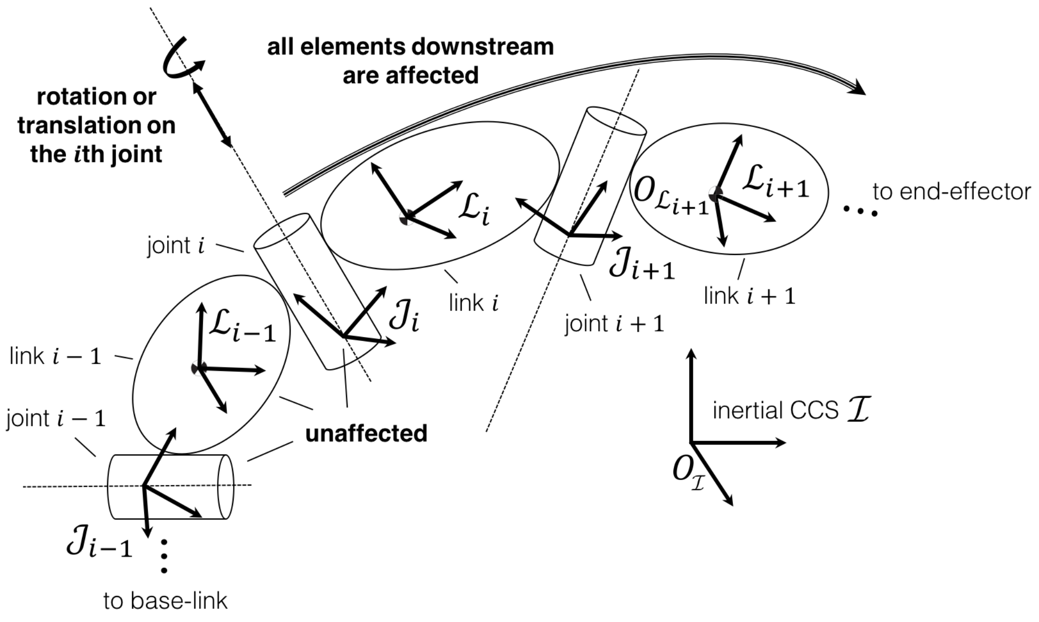

Fig. 1.4: Effects of a joint displacement.

1.3 SPART Tutorial – Robot Model

1.3.1 Example of a multibody system

This tutorial uses the following spacecraft equipped with a 3 degree-of-freedom manipulator as example.

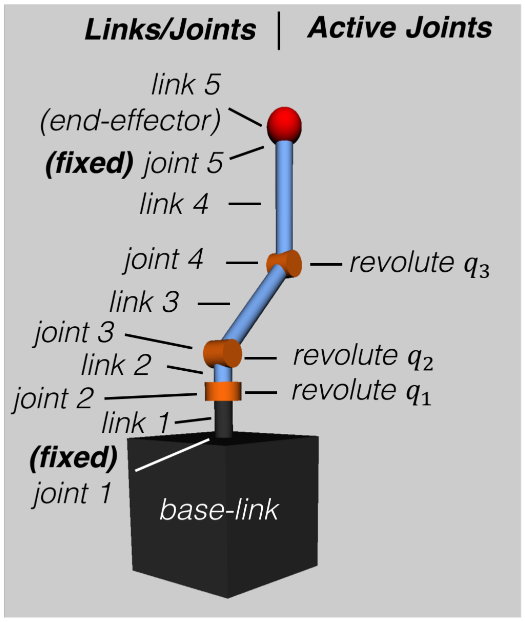

Fig. 1.5: Example of a multibody system – spacecraft with a 3 degree-of-freedom manipulator.

1.3.2 Using a URDF to describe a multibody system

In SPART, the preferred method to describe a multibody system is via a URDF file. URDF files are also the descriptor files used in ROS. To learn how to write URDF files, refer to the URDF documentation on the ROS wiki.

It is important to note that in SPART:

- Only fixed, revolute, continuous, and prismatic joints are accepted.
- Continuous and revolute joints are treated equally. Joint displacement, velocity, and effort limits are not taken into consideration.
- Joint dynamics (*e.g.*, damping) are not modeled.

The URDF of the spacecraft with the manipulator can be found in `URDF_models\SC_3DoF.urdf`. Other examples are included in the *URDF_models‘* folder, and a list of these models can be found in *Examples of URDF Models*.

Note: xacro files can be converted to a *pure* URDF file by running the following command

```
rosrun xacro xacro --inorder -o robot.urdf robot.urdf.xacro
```

1.3.3 Reading a URDF file

SPART takes a URDF file and converts it to the SPART `robot` structure.

```
%URDF filename
filename='SC_3DoF.urdf';

%Create robot model
[robot,robot_keys] = urdf2robot(filename);
```

1.3.4 The `robot` structure

The `robot` structure contains all the required kinematic and dynamic information of the multibody system. Build as a MATLAB structure it contains the following fields:

- robot.name – Name of the robot.
- robot.n_q – Number of active joints.
- robot.n_links_joints – Number of manipulator links and joints (includes fixed joints, and excludes the base-link).
- robot.links – Structure with the data of the links.
  - robot.links(i).id – Link id.
  - robot.links(i).parent_joint – Parent joint id.
  - robot.links(i).T – Homogeneous transformation matrix from parent joint [4x4].
  - robot.links(i).mass – Link's mass.
  - robot.links(i).inertia – Link's inertia matrix [3x3], projected in the body-fixed link's CCS.
- robot.joints – Structure with the data of the joints.
  - robot.joints(i).id – Joint id.
  - robot.joints(i).type – Joint type (0 – fixed, 1 – revolute, 2 – prismatic)
  - robot.joints(i).q_id – Active joint id (if q_id=-1 indicates a fixed joint).
  - robot.joints(i).parent_link – Parent link id.
  - robot.joints(i).child_link – Child link id.
  - robot.joints(i).axis – Joint rotation or sliding axis [3x1].
  - robot.joints(i).T – Homogeneous transformation matrix from parent link [4x4].
- robot.base_link – Structure with the data of the base-link.
  - robot.base_link.mass – Base-link's mass.
  - robot.base_link.inertia – Base-link's inertia matrix [3x3], projected in the body-fixed base-link's CCS.
- robot.con – Structure with additional connectivity information.
  - robot.con.branch – Branch connectivity map. This is a [nxn] lower triangular matrix. If the i,j element is 1 it means that the ith and jth link are on the same branch.
  - robot.con.child – A [nxn] matrix. If the i,j element is 1, then the ith link is a child of the jth link.
  - robot.con.child_base – A [nx1] matrix. If the ith element is 1, the ith link is connected to the base-link.

1.3.5 Understanding the `robot` structure and the `robot_keys` map

After reading the URDF of the spacecraft with the robotic manipulator, we obtain the `robot` structure and we can start querying it.

For example let's check how many acting joints there are:

```
>> robot.n_q

ans =

  3
```

As the manipulator has only 3 revolute joints, this results is correct. Now let's check how many joints and links there are in the system:

```
>> robot.n_joints_links

ans =

  5
```

which coincides with the number of revolute and fixed joints added together.

The `robot_keys` map can be used to map the names used in the URDF to the numbers used in SPART to uniquely identify the joints and links.

To know the URDF names of all links, joints, and active joints:

```
link_names=keys(robot_keys.link_id);
joint_names=keys(robot_keys.joint_id);
qm_names=keys(robot_keys.q_id);
```

To get the corresponding id of a URDF joint/link name use:

```
robot_keys.link_id('Link_name')
robot_keys.joint_id('Joint_name')
robot_keys.q_id('Joint_name')
```

The properties of the ith link (*e.g.*, $i = 2$) can be inspected as follows:

```
i=2;
robot.links(i)
```

The properties of the joints can be inspected in an equivalent manner:

```
i=2;
robot.joints(i)
```

1.3.6 Alternative methods to create the `robot` structure

SPART also allows to create this `robot` structure using two other input methods:

- With the Denavit-Hartenberg (DH) parameters. Refer to the /DH page for the procedure to create a SPART `robot` structure using DH parameters.
- Manually populate the `robot` structure.

1.4 SPART Tutorial – Kinematics

1.4.1 Direct Kinematics

SPART can compute the position and orientation of all the links and joints. The definitions of the kinematic quantities of a generic link and joint are notionally shown in the following figure.

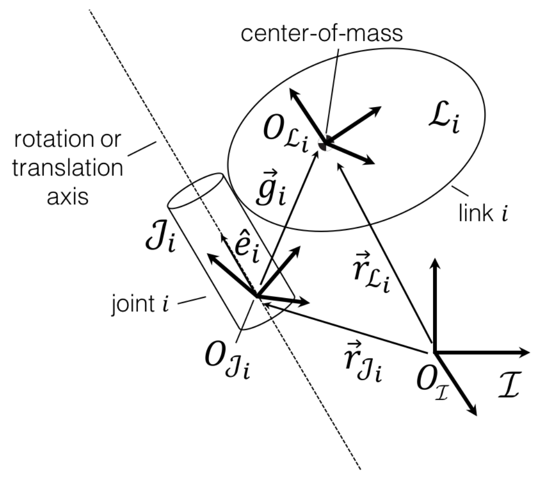

Fig. 1.6: Schematic disposition of a generic link and its associated joint.

To obtain the kinematics of the system, the base-link position $\mathrm{r}_0 \in \mathbb{R}^3$ and orientation, as a rotation matrix $\mathrm{R}_0 \in \mathrm{SO}(3)$, with respect to the inertial CCS are first specified.

```
%Base-link position and orientation
R0=eye(3);  %Rotation from base-link with respect to the inertial CCS.
r0=[0;0;0]; %Position of the base-link with respect to the origin of the inertial␣
↪frame, projected in the inertial CCS.
```

In SPART, the vectors are represented by a 3-by-1 column matrix containing the components of the vector projection to the inertial CCS. Projections to other CCS are explicitly marked.

Note: The position of the base-link `r0` refers to the base-link center-of-mass, corresponding with the origin of the URDF `inertial` tag. The base-link orientation `R0` also corresponds with the orientation of the CCS specified in the URDF `inertial` tag.

The joint displacements, $\mathbf{q}_m \in \mathbb{R}^n$, also also defined as a n-by-1 column matrix.

$$\mathbf{q}_m = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix}$$

```
%Joint displacements
qm=[0;0;0];
```

If the ith joint is revolute, q_i denotes an angular displacement around the rotation axis $\hat{e}_i$, whether if the ith joint is prismatic, q_i denotes a translational displacement along the sliding axis $\hat{e}_i$.

The set of $\mathbf{R}_0, \mathbf{r}_0, \mathbf{q}_m$ constitute a set of variables $\mathcal{Q}$, known as *generalized variables*, which full define the state of the multibody system,

$$\mathcal{Q} = \{\mathbf{R}_0, \mathbf{r}_0, \mathbf{q}_m\}$$

With the generalized variables specified, SPART is ready to compute the kinematics of the system.

```
%Kinematics
[RJ,RL,rJ,rL,e,g]=Kinematics(R0,r0,qm,robot);
```

The output of the `Kinematics` function is as follows:

- RJ – Joint 3x3 rotation matrices – as a [3x3xn] matrix.
- RL – Links 3x3 rotation matrices – as a [3x3xn] matrix.
- rJ – Positions of the joints projected in the inertial CCS – as a [3xn] matrix.
- rL – Positions of the links projected in the inertial CCS – as a [3xn] matrix.
- e – Joint rotation/sliding axis projected in the inertial CCS – as a [3xn] matrix.
- g – Vector from the origin of the ith joint CCS to the origin of the ith link CCS, projected in the inertial CCS – as a [3xn] matrix.

The results for each link/joint are stacked into a single variable. For example, to get the position of the second link center-of-mass:

```
%Position of the center-of-mass of a link
i=2;
rL(1:3,i)
```

and the rotation matrix corresponding to the second link CCS:

```
%Position of the center-of-mass of a link
i=2;
RL(1:3,1:3,i)
```

If your Matlab installation includes the Symbolic Math Toolbox SPART is able to obtain the analytic expressions of these kinematic quantities. To do so, just define the generalized variables as *symbolic expressions*.

```
%Base-link position
r0=sym('r0',[3,1],'real');

%Base-link orientation
Euler_Ang=sym('Euler_Ang',[3,1],'real');
R0 = Angles321_DCM(Euler_Ang)';

%Joint displacements
qm=sym('qm',[robot.n_q,1],'real');

%Kinematics
[RJ,RL,rJ,rL,e,g]=Kinematics(R0,r0,qm,robot);
```

Warning: To obtain analytic expressions, all inputs must be symbolic. Otherwise, errors may occur.

1.4.2 Differential kinematics

The angular and linear velocities of the ith link with respect to the inertial frame, projected in the inertial CCS, are encapsulated into the **twist** variable $\mathbf{t}_i \in \mathbb{R}^6$.

$$\mathbf{t}_i = \begin{bmatrix} \omega_i \\ \dot{\mathbf{r}}_i \end{bmatrix}$$

The twist can be recursively propagated outward from one link to the next using the 6-by-6 $\mathbf{B}_{ij}$ twist–propagation matrix and the 6-by-1 $\mathbf{p}_i$ twist–propagation "vector":

$$\mathbf{t}_i = \mathbf{B}_{ij}\mathbf{t}_j + \mathbf{p}_i\dot{q}_i \quad \text{for} \quad j = i-1$$

The twist–propagation matrices and "vectors", which form the basis of the differential kinematics, can be computed with the `DiffKinematics` function.

```
%Differential kinematics
[Bij,Bi0,P0,pm]=DiffKinematics(R0,r0,rL,e,g,robot);
```

The output of the differential kinematics is as follows:

- Bij – Twist–propagation [6x6xn] matrix (for manipulator i>0 and j>0).
- Bi0 – Twist–propagation [6x6xn] matrix (for i>0 and j=0).
- P0 – Base–link twist–propagation [6x6] matrix.
- pm – Manipulator twist–propagation [6xn] vector.

The set of generalized velocities $\mathbf{u} \in \mathbb{R}^{6+n}$ (joint-space velocities) contains the base-link velocities $\mathbf{u}_0 \in \mathbb{R}^6$ and the joint velocities $\mathbf{u}_m \in \mathbb{R}^n$.

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_m \end{bmatrix}$$

With the base-link and joint velocities defined as:

$$\mathbf{u}_0 = \begin{bmatrix} \omega_0^{\{\mathcal{L}_0\}} \\ \dot{\mathbf{r}}_0 \end{bmatrix}$$

$$\mathbf{u}_m = \begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_n \end{bmatrix}$$

Note that $\omega_0^{\{\mathcal{L}_0\}}$ denotes the angular velocity of the base-link, with respect to the inertial frame, projected in the base-link body-fixed CCS (this is the angular velocity that is obtained when using an onboard rate-gyro).

For the base-link, the twist is computed only using a modified 6-by-6 $\mathbf{P}_0$ twist-propagation matrix.

$$\mathbf{t}_0 = \mathbf{P}_0 u_0$$

With the twist-propagation quantities and the generalized velocities, the twists of all the links (operational-space velocities) can be determined.

```
%Generalized velocities (joint-space velocities)
u0=zeros(6,1); %Base-link angular (projected in the base-link body-fixed CCS) and␣
↪linear velocities.
um=[4;-1;5]*pi/180; %Joint velocities

%Twist (operational-space velocities)
[t0,tL]=Velocities(Bij,Bi0,P0,pm,u0,um,robot);
```

The output of the operational space velocities are as follows:

- t0 – Base–link twist projected in the inertial CCS – as a [6x1] matrix.
- tL – Manipulator links twist projected in the inertial CCS – as a [6xn] matrix.

1.4.3 Jacobians

The geometric Jacobian of a point p maps the joint-space velocities $\mathbf{u}$ into operational-space velocities of that point $\mathbf{t}_p$.

$$\mathbf{t}_p = \mathbf{J}_p \mathbf{u}$$

The contribution from the base-link and from the joints can be written more explicitly as:

$$\mathbf{J}_p = \begin{bmatrix} \mathbf{J}_{0p} & \mathbf{J}_{mp} \end{bmatrix}$$
$$\mathbf{t}_p = \mathbf{J}_{0p}\mathbf{u}_0 + \mathbf{J}_{mp}\mathbf{u}_m$$

The Jacobian of a point p, fixed to the ith link, can be obtained as follows:

```
%Jacobian of a point p in the ith link
%rp is the position of the point p, projected in the inertial CCS -- as a [3x1]␣
↪matrix.
[J0p, Jmp]=Jacob(rp,r0,rL,P0,pm,i,robot);
```

The Jacobians corresponding to the center-of-mass of the the ith link of the multibody system are then computed as follows:

```
%Jacobian of the ith Link
[J0i, Jmi]=Jacob(rL(1:3,i),r0,rL,P0,pm,i,robot);
```

1.4.4 Accelerations

The accelerations of a link can be encapsulated in a twist-rate $\dot{\mathbf{t}}_i \in \mathbb{R}^6$:

$$\dot{\mathbf{t}}_i = \begin{bmatrix} \dot{\omega}_i \\ \ddot{\mathbf{r}}_i \end{bmatrix}$$

The generalized accelerations $\dot{\mathbf{u}} \in \mathbb{R}^{6+n}$ of the system are defined as:

$$\dot{\mathbf{u}} = \begin{bmatrix} \dot{\mathbf{u}}_0 \\ \dot{\mathbf{u}}_m \end{bmatrix}$$

The twist-rate can then be computed as follows:

```
%Define generalized accelerations
u0=zeros(6,1); %Base-link angular (projected in the base-link body-fixed CCS) and␣
↪linear accelerations
um=[-0.1;0.2;0.1]*pi/180; %Joint accelerations

%Accelerations, twist-rate
[t0dot,tLdot]=Accelerations(t0,tL,P0,pm,Bi0,Bij,u0,um,u0dot,umdot,robot)
```

1.4.5 Jacobian time derivative

The time derivatives of the Jacobians can also be obtained:

```
%Jacobain time derivative
%rp is the position of the point p, projected in the inertial CCS -- as a [3x1]␣
↪matrix.
%tp is the twist of the point p -- as a [6x1] matrix.
[J0pdot, Jmpdot]=Jacobdot(rp,tp,r0,t0,rL,tL,P0,pm,i,robot)
```

The Jacobian time derivative can be used to obtain the twist-rate of a point on the multibody system.

$$\dot{\mathbf{t}}_p = \mathbf{J}_p\dot{\mathbf{u}} + \dot{\mathbf{J}}_p\mathbf{u}$$

1.5 SPART Tutorial – Dynamics

1.5.1 Equations of motion and inertia matrices

The equations of motion of a multibody system take the following form:

$$\mathbf{H}\dot{\mathbf{u}} + \mathbf{C}\mathbf{u} = \tau$$

with $\mathbf{H}\left(\mathcal{Q}\right) \in \mathbb{R}^{(6+n)\times(6+n)}$ being the symmetric, positive-definite Generalized Inertia Matrix (GIM), $\mathbf{C}\left(\mathcal{Q},\mathbf{u}\right) \in \mathbb{R}^{(6+n)\times(6+n)}$ the Convective Inertia Matrix (CIM), and $\tau \in \mathbb{R}^{6+n}$ the generalized forces (joint-space forces).

The contributions of the base-link and the manipulator can be made explicit when writing the equations of motion.

$$\begin{bmatrix} \mathbf{H}_0 & \mathbf{H}_{0m} \\ \mathbf{H}_{0m}^T & \mathbf{H}_m \end{bmatrix} \begin{bmatrix} \dot{\mathbf{u}}_0 \\ \dot{\mathbf{u}}_m \end{bmatrix} + \begin{bmatrix} \mathbf{C}_0 & \mathbf{C}_{0m} \\ \mathbf{C}_{m0} & \mathbf{C}_m \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_m \end{bmatrix} = \begin{bmatrix} \tau_0 \\ \tau_m \end{bmatrix}$$

These GIM and CIM are computed as follows:

```
%Inertias projected in the inertial frame
[I0,Im]=I_I(R0,RL,robot);
%Mass Composite Body matrix
[M0_tilde,Mm_tilde]=MCB(I0,Im,Bij,Bi0,robot);
%Generalized Inertia Matrix
[H0, H0m, Hm] = GIM(M0_tilde,Mm_tilde,Bij,Bi0,P0,pm,robot);
%Generalized Convective Inertia Matrix
[C0, C0m, Cm0, Cm] = CIM(t0,tL,I0,Im,M0_tilde,Mm_tilde,Bij,Bi0,P0,pm,robot);
```

Although the equations of motion can be used to solve the forward dynamic problem (determining the motion of the system given a set of applied forces $\tau \to \dot{\mathbf{u}}$) and the inverse dynamic problem (determining the forces required to produce a prescribe motion $\dot{\mathbf{u}} \to \tau$) there are more computationally efficient ways of doing so.

1.5.2 Forward dynamics

To solve the forward dynamics, the forces acting on the multibody system are specified as an input. The generalized forces τ are the forces acting on the joints $\tau_m \in \mathbb{R}^n$ and on the base-link $\tau_\mathbf{0} \in \mathbb{R}^6$. Specifically, the generalized forces τ act upon the generalized velocities $\mathbf{u}$.

In τ_0, as in the twist vector, the torques $\mathbf{n}_0^{\{\mathcal{L}_0\}} \in \mathbb{R}^3$, projected in the base-link body-fixed CCS, come first and are followed by forces $\mathbf{f}_0 \in \mathbb{R}^3$, applied to the base-link center-of-mass.

$$\tau_0 = \begin{bmatrix} \mathbf{n}_0^{\{\mathcal{L}_0\}} \\ \mathbf{f}_0 \end{bmatrix}$$

The wrench applied to the ith link, $\mathbf{w}_i \in \mathbb{R}^6$, encapsulates the torques $\mathbf{n}_i \in \mathbb{R}^3$ and forces $\mathbf{f}_i \in \mathbb{R}^3$, projected in the inertial CCS, applied to the center-of-mass of each link.

$$\mathbf{w}_i = \begin{bmatrix} \mathbf{n}_i \\ \mathbf{f}_i \end{bmatrix}$$

Here is an example of how to define them:

```
%Wrenches
wF0=zeros(6,1);
wFm=zeros(6,robot.n_links_joints);

%Generalized forces
tau0=zeros(6,1);
taum=zeros(robot.n_q,1);
```

After these forces are defined, a forward dynamic solver is available.

```
%Forward dynamics
[u0dot_FD,umdot_FD] = FD(tau0,taum,wF0,wFm,t0,tL,P0,pm,I0,Im,Bij,Bi0,u0,um,robot);
```

As an example, if you need to incorporate the weight of the links (with the z-axis being the vertical direction), set the wrenches as follows:

```
%Gravity
g=9.8; %[m s-2]

%Wrenches (includes gravity and assumes z is the vertical direction)
wF0=zeros(6,1);
wF0(6)=-robot.base_link(i).mass*g;
wFm=zeros(6,robot.n_links);
for i=1:robot.n_links
        wFm(6,i)=-robot.links(i).mass*g;
end
```

1.5.3 Inverse dynamics

For the inverse dynamics, the acceleration of the base-link $\dot{\mathbf{u}}_0$ and of the joints $\dot{\mathbf{u}}_m$ are specified, then the `ID` function computes the inverse dynamics, providing the required forces to obtain these accelerations.

```
%Generalized accelerations
u0dot=zeros(6,1);
umdot=zeros(robot.n_q,1);

%Oprational-space accelerations
[t0dot,tLdot]=Accelerations(t0,tL,P0,pm,Bi0,Bij,u0,um,u0dot,umdot,robot);

%Inverse Dynamics - Flying base
[tau0,taum] = ID(wF0,wFm,t0,tL,t0dot,tLdot,P0,pm,I0,Im,Bij,Bi0,robot);
```

If the base-link is left uncontrolled $\dot{\tau}_0 = \mathbf{0}$ (floating-base case) and thus the base-link acceleration is unknown, the `Floating_ID` function is available.

```
%Accelerations
umdot=zeros(robot.n_q,1);

```

```
%Inverse Dynamics - Floating Base
[taum_floating,u0dot_floating] = Floating_ID(wF0,wFm,Mm_tilde,H0,t0,tL,P0,pm,I0,Im,
↪Bij,Bi0,u0,um,umdot,robot);
```

1.5.4 Finding more information

The *Function Reference* provides more documentation on the SPART functions. If you don't find what you need you can always *Get in touch*.

1.6 Function Reference

Here is the documentation for all of SPART functions (the current list is incomplete).

1.6.1 Kinematics

Kinematics (*R0*, *r0*, *qm*, *robot*)

Computes the kinematics – positions and orientations – of the multibody system.

[RJ,RL,rJ,rL,e,g]=Kinematics(R0,r0,qm,robot)

Parameters

- R0 – Rotation matrix from the base-link CCS to the inertial CCS – [3x3].
- r0 – Position of the base-link center-of-mass with respect to the origin of the inertial frame, projected in the inertial CCS – [3x1].
- qm – Displacements of the active joints – [n_qx1].
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- RJ – Joints CCS 3x3 rotation matrices with respect to the inertial CCS – as a [3x3xn] matrix.
- RL – Links CCS 3x3 rotation matrices with respect to the inertial CCS – as a [3x3xn] matrix.
- rJ – Positions of the joints, projected in the inertial CCS – as a [3xn] matrix.
- rL – Positions of the links, projected in the inertial CCS – as a [3xn] matrix.
- e – Joint rotation/sliding axes, projected in the inertial CCS – as a [3xn] matrix.
- g – Vector from the origin of the ith joint CCS to the origin of the ith link CCS, projected in the inertial CCS – as a [3xn] matrix.

Remember that all the ouput magnitudes are projected in the **inertial frame**.

Examples on how to retrieve the results from a specific link/joint:

To retrieve the position of the ith link: `rL(1:3,i)`.

To retrieve the rotation matrix of the ith joint: `RJ(1:3,1:3,i)`.

See also: *`src.robot_model.urdf2robot()`* and *`src.robot_model.DH_Serial2robot()`*.

`DiffKinematics`(*R0*, *r0*, *rL*, *e*, *g*, *robot*)

Computes the differential kinematics of the multibody system.

[Bij,Bi0,P0,pm]=DiffKinematics(R0,r0,rL,e,g,robot)

Parameters

- R0 – Rotation matrix from the base-link CCS to the inertial CCS – [3x3].
- r0 – Position of the base-link center-of-mass with respect to the origin of the inertial frame, projected in the inertial CCS – [3x1].
- rL – Positions of the links, projected in the inertial CCS – as a [3xn] matrix.
- e – Joint rotation/sliding axes, projected in the inertial CCS – as a [3xn] matrix.
- g – Vector from the origin of the ith joint CCS to the origin of the ith link CCS, projected in the inertial CCS – as a [3xn] matrix.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- Bij – Twist-propagation matrix (for manipulator i>0 and j>0) – as a [6x6xn] matrix.
- Bi0 – Twist-propagation matrix (for i>0 and j=0) – as a [6x6xn] matrix.
- P0 – Base-link twist-propagation “vector” – as a [6x6] matrix.
- pm – Manipulator twist-propagation “vector” – as a [6xn] matrix.

Use *`src.kinematics_dynamics.Kinematics()`* to compute `rL,e`, and `g`.

See also: *`src.kinematics_dynamics.Kinematics()`* and *`src.kinematics_dynamics.Jacob()`*.

`Velocities`(*Bij*, *Bi0*, *P0*, *pm*, *u0*, *um*, *robot*)

Computes the operational-space velocities of the multibody system.

[t0,tL]=Velocities(Bij,Bi0,P0,pm,u0,um,robot)

Parameters

- Bij – Twist-propagation matrix (for manipulator i>0 and j>0) – as a [6x6xn] matrix.
- Bi0 – Twist-propagation matrix (for i>0 and j=0) – as a [6x6xn] matrix.
- P0 – Base-link twist-propagation “vector” – as a [6x6] matrix.
- pm – Manipulator twist-propagation “vector” – as a [6xn] matrix.
- u0 – Base-link velocities [omega,rdot]. The angular velocity is projected in the body-fixed CCS, while the linear velocity is projected in the inertial CCS – [6x1].
- um – Joint velocities – [n_qx1].
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- t0 – Base-link twist [omega,rdot], projected in the inertial CCS – as a [6x1] matrix.
- tL – Manipulator twist [omega,rdot], projected in the inertial CCS – as a [6xn] matrix.

Use *src.kinematics_dynamics.DiffKinematics()* to compute `Bij`, `Bi0`, `P0`, and `pm`.

See also: *src.kinematics_dynamics.Jacob()*

Jacob (*rp*, *r0*, *rL*, *P0*, *pm*, *i*, *robot*)

Computes the geometric Jacobian of a point *p*.

[J0, Jm]=Jacob(rp,r0,rL,P0,pm,i,robot)

Parameters

- rp – Position of the point of interest, projected in the inertial CCS – [3x1].
- r0 – Position of the base-link, projected in the inertial CCS – [3x1].
- rL – Positions of the links, projected in the inertial CCS – as a [3xn] matrix.
- P0 – Base-link twist-propagation “vector” – as a [6x6] matrix.
- pm – Manipulator twist-propagation “vector” – as a [6xn] matrix.
- i – Link id where the point *p* is located – int 0 to n.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- J0 – Base-link geometric Jacobian – [6x6].
- Jm – Manipulator geometric Jacobian – [6xn_q].

Examples:

To compute the velocity of the point *p* on the ith link:

```
%Compute Jacobians
[J0, Jm]=Jacob(rp,r0,rL,P0,pm,i,robot);
%Twist of that point
tp=J0*u0+Jm*um;
```

See also: *src.kinematics_dynamics.Kinematics()*, *src.kinematics_dynamics.DiffKinematics()*

Accelerations (*t0*, *tL*, *P0*, *pm*, *Bi0*, *Bij*, *u0*, *um*, *u0dot*, *umdot*, *robot*)

Computes the operational-space accelerations (twist-rate) of the multibody system.

[t0dot,tLdot]=Accelerations(t0,tL,P0,pm,Bi0,Bij,u0,um,u0dot,umdot,robot)

Parameters

- t0 – Base-link twist [omega,rdot], projected in the inertial CCS – as a [6x1] matrix.
- tL – Manipulator twist [omega,rdot], projected in the inertial CCS – as a [6xn] matrix.
- Bij – Twist-propagation matrix (for manipulator i>0 and j>0) – as a [6x6xn] matrix.
- Bi0 – Twist-propagation matrix (for i>0 and j=0) – as a [6x6xn] matrix.
- P0 – Base-link twist-propagation “vector” – as a [6x6] matrix.
- pm – Manipulator twist-propagation “vector” – as a [6xn] matrix.
- u0 – Base-link velocities [omega,rdot]. The angular velocity is projected in the body-fixed CCS, while the linear velocity is projected in the inertial CCS – [6x1].

- um – Joint velocities – [n_qx1].
- u0dot – Base-link accelerations [omegadot,rddot]. The angular acceleration is projected in a body-fixed CCS, while the linear acceleration is projected in the inertial CCS – [6x1].
- umdot – Manipulator joint accelerations – [n_qx1].
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- t0dot – Base-link twist-rate vector omegadot,rddot], projected in inertial frame – as a [6x1] matrix.
- tLdot – Manipulator twist-rate vector omegadot,rddot], projected in inertial frame – as a [6xn] matrix.

See also: `src.kinematics_dynamics.Jacobdot()`.

`Jacobdot`(*rp*, *tp*, *r0*, *t0*, *rL*, *tL*, *P0*, *pm*, *i*, *robot*)

Computes the geometric Jacobian time-derivative of a point *p*.

[J0dot, Jmdot]=Jacobdot(rp,tp,r0,t0,rL,tL,P0,pm,i,robot)

Parameters

- rp – Position of the point of interest, projected in the inertial CCS – [3x1].
- tp – Twist of the point of interest [omega,rdot], projected in the intertial CCS – [6x1].
- r0 – Position of the base-link center-of-mass with respect to the origin of the inertial frame, projected in the inertial CCS – [3x1].
- t0 – Base-link twist [omega,rdot], projected in the inertial CCS – as a [6x1] matrix.
- rL – Positions of the links, projected in the inertial CCS – as a [3xn] matrix.
- tL – Manipulator twist [omega,rdot], projected in the inertial CCS – as a [6xn] matrix.
- P0 – Base-link twist-propagation "vector" – as a [6x6] matrix.
- pm – Manipulator twist-propagation "vector" – as a [6xn] matrix.
- i – Link id where the point *p* is located – int 0 to n.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- J0dot – Base-link Jacobian time-derivative – as a [6x6] matrix.
- Jmdot – Manipulator Jacobian time-derivative – as a [6xn_q] matrix.

Examples:

To compute the acceleration of a point `p` on the ith link:

```
%Compute Jacobians
[J0, Jm]=Jacob(rp,r0,rL,P0,pm,i,robot);
Compute Jacobians time-derivatives
[J0dot, Jmdot]=Jacobdot(rp,tp,r0,t0,rL,tL,P0,pm,i,robot)
%Twist-rate of that point
tpdot=J0*u0dot+J0dot*u0+Jm*umdot+Jmdot*um;
```

See also: `src.kinematics_dynamics.Accelerations()` and `src.kinematics_dynamics.Jacob()`.

Center_of_Mass(*r0*, *rL*, *robot*)

Computes the center-of-mass (CoM) of the system.

r_com = Center_of_Mass(r0,rL,robot)

Parameters

- r0 – Position of the base-link, projected in the inertial CCS – [3x1].
- rL – Positions of the links, projected in the inertial CCS – as a [3xn] matrix.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- r_com – Location of the center-of-mass, projected in the inertial CCS – [3x1].

Use *src.kinematics_dynamics.Kinematics()* to compute `rL`.

This function can also be used to compute the velocity/acceleration of the center-of-mass. To do it use as paremeters the velocities `r0dot,rLdot` or acceleration `r0ddot,rLddot` and you will get the CoM velocity `rcomdot` or acceleration `rcomddot`.

See also: *src.kinematics_dynamics.Kinematics()*

NOC(*r0*, *rL*, *P0*, *pm*, *robot*)

Computes the Natural Orthogonal Complement (NOC) matrix (generalized Jacobian).

[N] = NOC(r0,rL,P0,pm,robot)

Parameters

- r0 – Position of the base-link, projected in the inertial CCS – [3x1].
- rL – Positions of the links, projected in the inertial CCS – as a [3xn] matrix.
- P0 – Base-link twist-propagation "vector" – as a [6x6] matrix.
- pm – Manipulator twist-propagation "vector" – as a [6xn] matrix.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- N – Natural Orthogonal Complement (NOC) matrix – a [(6+6*n)x(6+n_q)] matrix.

Examples:

To compute the velocities of all links:

```
%Compute NOC.
[N] = NOC(r0,rL,P0,pm,robot)
%Generalized twist (concatenation of the twist of all links).
t=N*[u0;um];
%Twist of the base-link
t0=t(1:6,1);
%Twist of the ith link
i=2;
ti=t(6*i:6+6*i,1);
```

See also: *src.kinematics_dynamics.Jacob()* and *src.kinematics_dynamics.NOCdot()*.

NOCdot(*r0*, *t0*, *rL*, *tL*, *P0*, *pm*, *robot*)

Computes the Natural Orthogonal Complement (NOC) matrix time-derivative.

[Ndot] = NOCdot(r0,t0,rL,tL,P0,pm,robot)

Parameters

- r0 – Position of the base-link center-of-mass with respect to the origin of the inertial frame, projected in the inertial CCS – [3x1].
- t0 – Base-link twist [omega,rdot], projected in the inertial CCS – as a [6x1] matrix.
- rL – Positions of the links, projected in the inertial CCS – as a [3xn] matrix.
- tL – Manipulator twist [omega,rdot], projected in the inertial CCS – as a [6xn] matrix.
- P0 – Base-link twist-propagation "vector" – as a [6x6] matrix.
- pm – Manipulator twist-propagation "vector" – as a [6xn] matrix.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- Ndot – Natural Orthogonal Complement (NOC) matrix time-derivative – as a [(6+6*n)x(6+n_q)] matrix.

Examples:

To compute the operational-space accelerations of all links:

```
%Compute NOC
[N] = NOC(r0,rL,P0,pm,robot)
%Compute NOC time-derivative
[Ndot] = NOCdot(r0,t0,rL,tL,P0,pm,robot)
%Twist time-derivatives of all the links
tdot=N*[u0dot;umdot]+Ndot*[u0;um];
%Twist time-derivative of the base-link
t0dot=tdot(1:6,1);
%Twist time-derivative of the ith link
i=2;
tidot=tdot(6*i:6+6*i,1);
```

See also: `src.kinematics_dynamics.Jacobdot()` and `src.kinematics_dynamics.NOC()`.

1.6.2 Dynamics

FD (*tau0, taum, wF0, wFm, t0, tm, P0, pm, I0, Im, Bij, Bi0, u0, um, robot*)

This function solves the forward dynamics (FD) problem (it obtains the acceleration from forces).

[u0dot,umdot] = FD(tau0,taum,wF0,wFm,t0,tm,P0,pm,I0,Im,Bij,Bi0,u0,um,robot)

Parameters

- tau0 – Base-link forces [n,f]. The torque n is projected in the body-fixed CCS, while the force f is projected in the inertial CCS – [6x1].
- taum – Joint forces/torques – as a [n_qx1] matrix.
- wF0 – Wrench acting on the base-link center-of-mass [n,f], projected in the inertial CCS – as a [6x1] matrix.
- wFm – Wrench acting on the links center-of-mass [n,f], projected in the inertial CCS – as a [6xn] matrix.
- t0 – Base-link twist [omega,rdot], projected in the inertial CCS – as a [6x1] matrix.
- tL – Manipulator twist [omega,rdot], projected in the inertial CCS – as a [6xn] matrix.

- P0 – Base-link twist-propagation "vector" – as a [6x6] matrix.
- pm – Manipulator twist-propagation "vector" – as a [6xn] matrix.
- I0 – Base-link inertia matrix, projected in the inertial CCS – as a [3x3] matrix.
- Im – Links inertia matrices, projected in the inertial CCS – as a [3x3xn] matrix.
- Bij – Twist-propagation matrix (for manipulator i>0 and j>0) – as a [6x6xn] matrix.
- Bi0 – Twist-propagation matrix (for i>0 and j=0) – as a [6x6xn] matrix.
- u0 – Base-link velocities [omega,rdot]. The angular velocity is projected in the body-fixed CCS, while the linear velocity is projected in the inertial CCS – [6x1].
- um – Joint velocities – [n_qx1].
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- u0dot – Base-link accelerations [omegadot,rddot]. The angular acceleration is projected in a body-fixed CCS, while the linear acceleration is projected in the inertial CCS – [6x1].
- umdot – Manipulator joint accelerations – [n_qx1].

See also: `src.kinematics_dynamics.ID()` and `src.kinematics_dynamics.I_I()`.

ID(*wF0*, *wFm*, *t0*, *tL*, *t0dot*, *tLdot*, *P0*, *pm*, *I0*, *Im*, *Bij*, *Bi0*, *robot*)

This function solves the inverse dynamics (ID) problem (it obtains the generalized forces from the accelerations) for a manipulator.

[tau0,taum] = ID(wF0,wFm,t0,tL,t0dot,tLdot,P0,pm,I0,Im,Bij,Bi0,robot)

Parameters

- wF0 – Wrench acting on the base-link center-of-mass [n,f], projected in the inertial CCS – as a [6x1] matrix.
- wFm – Wrench acting on the links center-of-mass [n,f], projected in the inertial CCS – as a [6xn] matrix.
- t0 – Base-link twist [omega,rdot], projected in the inertial CCS – as a [6x1] matrix.
- tL – Manipulator twist [omega,rdot], projected in the inertial CCS – as a [6xn] matrix.
- t0dot – Base-link twist-rate vector omegadot,rddot], projected in inertial frame – as a [6x1] matrix.
- tLdot – Manipulator twist-rate vector omegadot,rddot], projected in inertial frame – as a [6xn] matrix.
- P0 – Base-link twist-propagation "vector" – as a [6x6] matrix.
- pm – Manipulator twist-propagation "vector" – as a [6xn] matrix.
- I0 – Base-link inertia matrix, projected in the inertial CCS – as a [3x3] matrix.
- Im – Links inertia matrices, projected in the inertial CCS – as a [3x3xn] matrix.
- Bij – Twist-propagation matrix (for manipulator i>0 and j>0) – as a [6x6xn] matrix.
- Bi0 – Twist-propagation matrix (for i>0 and j=0) – as a [6x6xn] matrix.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- tau0 – Base-link forces [n,f]. The torque n is projected in the body-fixed CCS, while the force f is projected in the inertial CCS – [6x1].
- taum – Joint forces/torques – as a [n_qx1] matrix.

See also: *src.kinematics_dynamics.Floating_ID()* and *src.kinematics_dynamics.FD()*.

Floating_ID(*wF0*, *wFm*, *Mm_tilde*, *H0*, *t0*, *tm*, *P0*, *pm*, *I0*, *Im*, *Bij*, *Bi0*, *u0*, *um*, *umdot*, *robot*)

This function solves the inverse dynamics problem (it obtains the generalized forces from the accelerations) for a manipulator with a floating base.

[taum,u0dot] = Floating_ID(wF0,wFm,Mm_tilde,H0,t0,tm,P0,pm,I0,Im,Bij,Bi0,u0,um,umdot,robot)

Parameters

- wF0 – Wrench acting on the base-link center-of-mass [n,f], projected in the inertial CCS – as a [6x1] matrix.
- wFm – Wrench acting on the links center-of-mass [n,f], projected in the inertial CCS – as a [6xn] matrix.
- M0_tilde – Base-link mass composite body matrix – as a [6x6] matrix .
- Mm_tilde – Manipulator mass composite body matrix – as a [6x6xn] matrix.
- t0 – Base-link twist [omega,rdot], projected in the inertial CCS – as a [6x1] matrix.
- tL – Manipulator twist [omega,rdot], projected in the inertial CCS – as a [6xn] matrix.
- P0 – Base-link twist-propagation "vector" – as a [6x6] matrix.
- pm – Manipulator twist-propagation "vector" – as a [6xn] matrix.
- I0 – Base-link inertia matrix, projected in the inertial CCS – as a [3x3] matrix.
- Im – Links inertia matrices, projected in the inertial CCS – as a [3x3xn] matrix.
- Bij – Twist-propagation matrix (for manipulator i>0 and j>0) – as a [6x6xn] matrix.
- Bi0 – Twist-propagation matrix (for i>0 and j=0) – as a [6x6xn] matrix.
- u0 – Base-link velocities [omega,rdot]. The angular velocity is projected in the body-fixed CCS, while the linear velocity is projected in the inertial CCS – [6x1].
- um – Joint velocities – [n_qx1].
- umdot – Manipulator joint accelerations – [n_qx1].
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- tau0 – Base-link forces [n,f]. The torque n is projected in the body-fixed CCS, while the force f is projected in the inertial CCS – [6x1].
- taum – Joint forces/torques – as a [n_qx1] matrix.

See also: `src.kinematics_dynamics.sID()` and *src.kinematics_dynamics.FD()*.

I_I(*R0*, *RL*, *robot*)

Projects the link inertias in the inertial CCS.

[I0,Im]=I_I(R0,RL,robot)

Parameters

- R0 – Rotation matrix from the base-link CCS to the inertial CCS – [3x3].

- RL – Links CCS 3x3 rotation matrices with respect to the inertial CCS – as a [3x3xn] matrix.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- I0 – Base-link inertia matrix, projected in the inertial CCS – as a [3x3] matrix.
- Im – Links inertia matrices, projected in the inertial CCS – as a [3x3xn] matrix.

See also: `src.kinematics_dynamics.MCB()`.

MCB(*I0*, *Im*, *Bij*, *Bi0*, *robot*)

Computes the Mass Composite Body Matrix (MCB) of the multibody system.

[M0_tilde,Mm_tilde]=MCB(I0,Im,Bij,Bi0,robot)

Parameters

- I0 – Base-link inertia matrix, projected in the inertial CCS – as a [3x3] matrix.
- Im – Links inertia matrices, projected in the inertial CCS – as a [3x3xn] matrix.
- Bij – Twist-propagation matrix (for manipulator i>0 and j>0) – as a [6x6xn] matrix.
- Bi0 – Twist-propagation matrix (for i>0 and j=0) – as a [6x6xn] matrix.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- M0_tilde – Base-link mass composite body matrix – as a [6x6] matrix .
- Mm_tilde – Manipulator mass composite body matrix – as a [6x6xn] matrix.

See also: `src.kinematics_dynamics.I_I()`.

GIM(*M0_tilde*, *Mm_tilde*, *Bij*, *Bi0*, *P0*, *pm*, *robot*)

Computes the Generalized Inertia Matrix (GIM) H of the multibody vehicle.

This function uses a recursive algorithm.

[H0, H0m, Hm] = GIM(M0_tilde,Mm_tilde,Bij,Bi0,P0,pm,robot)

Parameters

- M0_tilde – Base-link mass composite body matrix – as a [6x6] matrix .
- Mm_tilde – Manipulator mass composite body matrix – as a [6x6xn] matrix.
- Bij – Twist-propagation matrix (for manipulator i>0 and j>0) – as a [6x6xn] matrix.
- Bi0 – Twist-propagation matrix (for i>0 and j=0) – as a [6x6xn] matrix.
- P0 – Base-link twist-propagation "vector" – as a [6x6] matrix.
- pm – Manipulator twist-propagation "vector" – as a [6xn] matrix.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- H0 – Base-link inertia matrix – as a [6x6] matrix.
- H0m – Base-link – manipulator coupling inertia matrix – as a [6xn_q] matrix.
- Hm – Manipulator inertia matrix – as a [n_qxn_q] matrix.

To obtain the full generalized inertia matrix H:

```
%Compute H
[H0, H0m, Hm] = GIM(M0_tilde,Mm_tilde,Bij,Bi0,P0,pm,robot);
H=[H0,H0m;H0m';Hm];
```

See also: *src.kinematics_dynamics.CIM()*.

CIM(*t0*, *tL*, *I0*, *Im*, *M0_tilde*, *Mm_tilde*, *Bij*, *Bi0*, *P0*, *pm*, *robot*)

Computes the Generalized Convective Inertia Matrix C of the multibody system.

Parameters

- t0 – Base-link twist [omega,rdot], projected in the inertial CCS – as a [6x1] matrix.
- tL – Manipulator twist [omega,rdot], projected in the inertial CCS – as a [6xn] matrix.
- I0 – Base-link inertia matrix, projected in the inertial CCS – as a [3x3] matrix.
- Im – Links inertia matrices, projected in the inertial CCS – as a [3x3xn] matrix.
- M0_tilde – Base-link mass composite body matrix – as a [6x6] matrix .
- Mm_tilde – Manipulator mass composite body matrix – as a [6x6xn] matrix.
- Bij – Twist-propagation matrix (for manipulator i>0 and j>0) – as a [6x6xn] matrix.
- Bi0 – Twist-propagation matrix (for i>0 and j=0) – as a [6x6xn] matrix.
- P0 – Base-link twist-propagation “vector” – as a [6x6] matrix.
- pm – Manipulator twist-propagation “vector” – as a [6xn] matrix.
- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- C0 -> Base-link convective inertia matrix – as a [6x6] matrix.
- C0m -> Base-link - manipulator coupling convective inertia matrix – as a [6xn_q] matrix.
- Cm0 -> Manipulator - base-link coupling convective inertia matrix – as a [n_qx6] matrix.
- Cm -> Manipulator convective inertia matrix – as a [n_qxn_q] matrix.

To obtain the full convective inertia matrix C:

```
%Compute the Convective Inertia Matrix C
[C0, C0m, Cm0, Cm] = CIM(t0,tL,I0,Im,M0_tilde,Mm_tilde,Bij,Bi0,P0,pm,robot)
C=[C0,C0m;Cm0,Cm];
```

See also: *src.kinematics_dynamics.GIM()*.

1.6.3 Robot Model

urdf2robot(*filename*, *verbose_flag*)

Creates a SPART robot model from a URDF file.

[robot,robot_keys] = urdf2robot(filename,verbose_flag)

Parameters

- filename – Path to the URDF file.
- verbose_flag – True for verbose output (default False).

Returns

- robot – Robot model (see *SPART Tutorial – Robot Model*).
- robot_keys – Links/Joints name map (see *SPART Tutorial – Robot Model*).

This function was inspired by: https://github.com/jhu-lcsr/matlab_urdf/blob/master/load_ne_id.m

`DH_Serial2robot`(*DH_data*)

Transforms a description of a multibody system, provided in Denavit-Hartenberg parameters, into the SPART robot model.

[robot,T_Ln_EE] = DH_Serial2robot(DH_data)

Parameters

- DH_data – Structure containing the DH parameters. (see /DH).

Returns

- robot – Robot model (see *SPART Tutorial – Robot Model*).
- T_Ln_EE – Homogeneous transformation matrix from last link to end-effector –[4x4].

DH descriptions are only supported for serial configurations.

`ConnectivityMap`(*robot*)

Produces the connectivity map for a robot model.

[branch,child,child_base]=ConnectivityMap(robot)

Parameters

- robot – Robot model (see *SPART Tutorial – Robot Model*).

Returns

- branch – Branch connectivity map. This is a [nxn] lower triangular matrix. If the i,j element is 1 it means that the ith and jth link are on the same branch.
- child – A [nxn] matrix. If the i,j element is 1, then the ith link is a child of the jth link.
- child_base – A [nx1] matrix. If the ith element is 1, the ith link is connected to the base-link.

See also: `src.robot_model.urdf2robot()` and `src.robot_model.DH_Serial2robot()`.

1.6.4 Attitude Transformations

`Angles321_DCM`(*Angles*)

Convert the Euler angles (321 sequence), x-phi, y-theta, z-psi to its DCM equivalent.

DCM = Angles321_DCM(Angles)

Parameters

- Angles – Euler angles [x-phi, y-theta, z-psi] – [3x1].

Returns

- DCM – Direction Cosine Matrix – [3x3].

See also: `src.attitude_transformations.Angles123_DCM()` and `src.attitude_transformations.DCM_Angles321()`.

`Angles123_DCM`(*Angles*)

#codegen Convert the Euler angles (123 sequence), x-phi, y-theta, z-psi to DCM.

DCM = Angles123_DCM(Angles)

Parameters

- Angles – Euler angles [x-phi, y-theta, z-psi] – [3x1].

Returns

- DCM – Direction Cosine Matrix – [3x3].

See also: *src.attitude_transformations.Angles321_DCM()* and `src.attitude_transformations.DCM_Angles321()`.

1.6.5 Utilities

SkewSym(*x*)

Computes the skew-symmetric matrix of a vector, which is also the left-hand-side matricial equivalent of the vector cross product

[x_skew] = SkewSym(x)

Parameters

- x – [3x1] column matrix (the vector).

Returns

- x_skew – [3x3] skew-symmetric matrix of x.

1.7 Examples of URDF Models

SPART comes with a few URDF models ready to use. These models are located in `/URDF_models`, which should be already included in your path (see *Installing SPART*).

1.7.1 Spacecraft with robotic manipulators

- `SC_3DoF.urdf` a simple spacecraft with a 3 DoF manipulator.

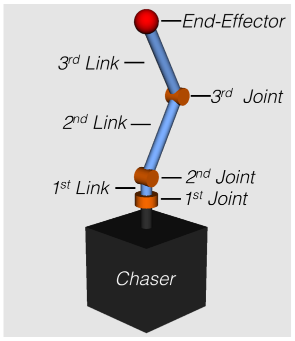

Fig. 1.7: Simple spacecraft with a 3 DoF manipulator.

1.7.2 Spacecraft

- `Simple_Spacecraft.urdf` a simple spacecraft model.

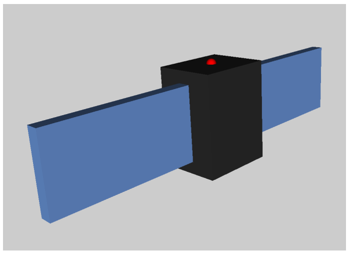

Fig. 1.8: Simple spacecraft with 2 solar panels.

1.7.3 Fixed-base manipulators

- `kuka_iiwa.urdf` Kuka iiwa manipulator.
- `kuka_lwr.urdf` Kuka lwr manipulator.

1.7.4 Acknowledgments

The `kuka_iiwa.urdf` and `kuka_lwr.urdf` URDF models have been obtained from the Bullet Physics SDK.

1.8 Citing SPART

Are you using SPART in research work to be published? If so, please include explicit mention of SPART. We suggest to use the following language:

> To derive the kinematic and dynamic properties of the system we used SPART, a modeling and control software package for mobile-base robotic multibody systems [1].

with the following corresponding entry in your references:

> [1] J. Virgili-Llop et al., "SPART: an open-source modeling and control toolkit for mobile-base robotic multibody systems with kinematic tree topologies," https://github.com/NPS-SRL/SPART.

The corresponding BiBTeX citation is given below:

```
@misc{SPART,
Author = {Josep Virgili-Llop and others},
Howpublished = {\url{https://github.com/NPS-SRL/SPART}},
Title = {{SPART}: an open-source modeling and control toolkit for mobile-base robotic␣
↪multibody systems with kinematic tree topologies}}
```

1.9 Getting in touch

If you can't figure how to do something with SPART, have found a bug in the code, or want to suggest an improvement or new feature, you can always open an issue or send an email to jvirgili@nps.edu. We will try our best to help you, resolve the problem, or implement the suggested features.

CHAPTER 2

License

SPART is released under the LGPLv3 license.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index