# Function Reference

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


- Kinematics
- Dynamics
- Robot Model
- Attitude Transformations
- Utilities


## 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()`.

# 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()` .

# 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 Using the Denavit–Hartenberg convention with SPART).

**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()`.

# 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()`.

# 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.