# Recursive Dynamic Simulator (ReDySim): Floating-base Module Instruction Manual for Forward Dynamics

## Getting started

1. Require MATLAB 2009a or higher version in order to use this module.

## Run Demo of a 3-link robot mounted on a satellite

1. Run function file *run_me.m*. This will simulate the system and generate the plots for joint motions, total momentum and energy.
2. The simulation data can be animated using function *animate*(*tskip*), where *tskip* is the input variable representing the sampling time.
3. User may also copy all the files from folder '7-link' or 'Dual Arm' in the main folder containing the file *run_me.m* in order to simulate either of them.

## Simulation of a system using *run_me.m*

1. Function prototype: `run_me()`
2. Enter the input parameters such as type modified-DH parameters (See appendix A), inertia tensors, masses, etc. in the file *inputs.m* and save the file.
3. Enter the initial conditions and integration parameters such as initial state variable, integration tolerances, etc. in the file *initials.m* and save the file.
4. Enter the joint trajectories to be tracked, i.e., desired trajectories, in the file *trajectory.m*.
5. Enter joint torques with/without control law in the function file *torque.m* and save the file
6. Ensure that the parameters are properly entered in the files *inputs.m*, *initials.m*, trajectory.m and *torque.m*.
7. Run file *run_me.m* to simulate the system and to generate the plots for joint motions, momentum and energy. It actually runs *runfor.p* (for simulation), *plot_motion.m* (for motion plot), *energy.p* (for energy calculation), *ploten.m* (for energy plot), *momentum.p* (for momentum calculation) and *plot_mtum.m* (for momentum plot).
8. Ensure that both the *timevar.dat* and *statevar.dat* files are in the program folder containing file *animate.m*. Run file *animate.m* in order to animate system using the simulation data. This file is specific to the system under study and hence need to be modified depending on the system or mechanism.

## Details of the functions used in the simulation above

### 1. Input functions

These function files are required as the input.

#### *inputs.m*

Function prototype: `[n nq alp a b bt dx dy dz al alt m g  Icxx Icyy Iczz Icxy Icyz Iczx]=inputs()`

- The number of links (n), type of motion (nq), i.e., 0 for planar and 1 for spatial, model parameters such as modified DH parameters (alp, a, b), parent of each link (bt), the X, Y and Z coordinates of position of Center-of-Mass (COM) form the origin (dx, dy, dz), link lengths (al), distance from origin to link tip (alt), masses (m), vector of gravitational acceleration (g), and elements of inertia tensors about COM (Icxx, Icyy, Iczz, Icxy, Icyz, Iczx)..

***initials.m***

- Function prototype: `[y0, ti, tf, incr, rtol, atol]=initials()`
- The vector of initial base position and ZXY-Euler angles and joint angles and their initial rates, and actuator energy (y0), initial and final time of simulation (ti, tf), sampling time (incr) and relative and absolute integration tolerances (rtol and atol).

***trajectory.m***

- Function prototype: `[th_d dth_d ddth_d]= trajectory(t, n, tf)`
- The joint-level trajectories to be followed are provided in this function. The outputs of this function are vectors of desired joint angles (th_d)., velocities (dth_d) and accelerations (ddth_d).

***torque.m***

- Function prototype: `[tu_q tu_th] = torque(t, n, tf, q, th, dq, dth)`
- The forces associated with base (tu_q) and joints torques (tu_th) are computed in this files. It is worth noting that the integrator passes current time (t), number of links (n), simulation time (tf), vectors of base position and ZXY euler angles (q) joint angels (th), base linear velocity and rates of euler angles (dq) and joint velocities (dth) as input to this function, and hence use can write any control algorithms such as P, PD, model-based, etc, using them.

## 2. Output function

This file generates output in the forms of either data files or plots.

***runfor.p***

- Function prototype: `runfor()`
- The file *runfor.m* performs simulation and generate output data files *timevar.dat* and *statevar.da*t containing time and state variable for given simulation time period.
- First 6 columns in the file *statevar.dat* contains base positions (q), next *n* columns contain joint angles, next 6 columns contains base velocities (dq), next *n* columns contain joint rates, and the last column contains actuator energy.

***plot_motion.m***

- Function prototype: `plot_motion()`
- The file *plot_motion.m* plots base and joint position and velocity using data files *timevar.dat* and *statevar.dat*.

***energy.p***

- Function prototype: `energy()`
- The file *energy.m* calculates potential energy, kinetic energy, actuator energy and total energy.

- It also generates data file *envar.dat* containing energy values. In the file *envar.dat* the first, second, third and fourth columns contain values of potential energy, kinetic energy, actuator energy and total energy, respectively.

***plot_en.m***

- Function prototype: `plot_en()`
- The file *plot_en.m* show the energy distribution over the simulation time period using data files *timevar.dat* and *envar.dat*.

***momentum.p***

- Function prototype: `momentum()`
- The file *momentum.m* calculates linear and angular momentums.
- It also generates data file *mtvar.dat* containing momentum values. In the file *mtvar.dat* the first three columns contain X, Y, and Z components of linear momentum, and the final three columns contain X, Y, And Z components of angular momentum.

***plot_mt.m***

- Function prototype: `plot_mt()`
- The file *plot_mt.m* plot the linear and angular momentum distribution over the simulation time period using data files *timevar.dat* and *mtvar.dat*.

***animate.m***

- Function prototype: `animate()`
- This function file animate the system using the simulation data. This file is specific to the system under study and hence need to be modified depending on the system or mechanism. It uses function *for_kine.m* to obtained Cartesian co-ordinate of the point on a link from joint angles saved in data file statevar.dat using forward kinematics relationships. User has full access to both the *animate.m* and *for_kine.m*.

**Note**: All the above files can be run by using *run_me.m* or independently in the order shown above.

## 3 Some other important functions

Some other important functions are outline below:

***sys_ode.m***

- This file contains differential equations of the system under study.

***fordyn_float.p***

- Function prototype: `[ddy]=fordyn_float(q,dq,th,dth, n,alp,a,b,bt,dx,dy,dz, al,alt,m,g,Icxx,Icyy,Iczz,Icxy,Icyz,Iczx, tu_q,tu_th)`
- The function calculates base and joint accelerations by using **UDU**$^T$ decomposition of the Generalized Inertia Matrix (GIM).

### *for_kine.m*

- Function prototype: `[tt vc  scf vcf sof stf sbf vtf ttf]=for_kine(q,th, dq, dth, n, alp, a, b, bt, dx, dy, dz, al, alt)`
- This function calculates angular velocity (tt), velocity of the COM (vc) in body fixed frame and position (scf) and velocity of the link COM (vcf), positions of origin (sof) COM, link tip (st) and base coordinate (sbf), link's tip velocity (vtf) and link's angular velocity (ttf), all in the inertial frame. As users have full access to this function, other points on the link can also be taken as output depending on the system.

**Note**:

Users have full access to functions *inputs.m*, *initials.m*, *trajectory.m*, *torque.m*, *plot_motion.m*, *plot_en.m*, *plot_mt.m*, *for_kine.m*, *sys_ode.m* and *animate.m* whereas the functions *run_for.p*, *fordyn_float.p*, *energy.p*, *momentum.p* are protected codes (pcodes) which however can be used as the function.