Recursive Dynamic Simulator (ReDySim): Floating-base (Legged robots) Instruction Manual for Forward Dynamics

Getting started

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

Run Demo of a Quadruped robot

1. Run function file *run_me.m*. This will simulate the quadruped robot and generates the plot for joint motions 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 the folder '*biped'* or '*hexapod'* in the main folder containing the file *run_me.m* in order to simulate either of them.

Simulation 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 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 and energy and momentum. It actually runs *runfor.p* (for simulation), *plot_motion.m* (for plotting joint and base motions), *energy.p* (for energy calculation), and *plot_en.m* (for energy 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 dof type alp a b bt dx dy dz m g Icxx Icyy Iczz Icxy Icyz Iczx aj]=inputs()`

- The number of links (n), Degrees-of-Freddom of the system (dof), type of system (type), i.e., closed or open , 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), masses (m), vector of gravitational acceleration (g), elements of inertia tensors about COM (Icxx, Icyy, Iczz, Icxy, Icyz, Iczx) and details of actuated joints (aj) are provided in this file.

initials.m

- Function prototype: `[n, y0, len_sum, ti, tf, incr, rtol, atol, int_type] = initials()`
- The number of link (n), vector of initial joint angles, velocities and actuator energy (y0), initial and final time of simulation (ti, tf), sampling time (incr), integration tolerances (atol, rtol), and type of integrator (int_type) are provided in this file.

trajectory.m

- Function prototype: `[thi dthi ddthi]= trajectory(tim)`
- The trajectories to be controlled can be provided in this function.

torque.m

- Function prototype: `[tau_d] = torque(t,n,th,dth)`
- The joint torques (tau_d) are entered in this files. Default values of torques are zeros.
- It is worth noting that the integrator passes current time (t), number of links (n), vectors of joint angels (th) 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. One can also integrate any user defined function in this function as well. The output of this function is joint torques.

jacobian.m

- Function prototype: `[J, dJ]=jacobian(th,dth)`
- The jacobian and its time derivative are entered in this function.
- It is worth noting that the integrator passes vectors of current joint angels (th) and joint velocities (dth) as inputs to this function and output is jacobian [J] and its time derivative [dJ].

2. Output function

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

runfor.m

- 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 *n*, *n* being number of link in the open-type system, columns in file *statevar.dat* contain positions associated with joint variables, and next *n*, i.e., (*n*+1) to (2*n*), columns contain time rates, i.e., velocities, associated with joint variables.

plot_motion.m

- Function prototype: `plot_motion()`

- The file *plot_motion.m* show the plots containing joint position and joint velocities using data files *timevar.dat* and *statevar.dat*.

energy.m

- 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 *energyvar.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*.

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.

ddth_tree_eff.m

- Function prototype:
```
[ddth]=ddth_tree_eff(th,n,alp,a,b,bt,dx,dy,dz,m,Icxx,Icyy,Iczz,Icxy,Icy
z,Iczx,phi)
```
- The function calculates joint acceleration by factorizing generalized inertia matrix.

invdyn_tree_eff.m

- Function prototype:
```
[tu] = invdyn_tree_eff(th,dth,ddth,n,alp,a,b,bt,dx,dy,dz,
m,g,Icxx,Icyy,Iczz,Icxy,Icyz,Iczx)
```
- The function is used to calculates non-inertial forces by substituting ddth=0.

GIM_tree.m

- Function prototype:
```
[I]=GIM_tree(th,n,alp,a,b,bt,dx,dy,dz,m,Icxx,Icyy,Iczz,Icxy,Icyz,Iczx)
```

- This function formulates the generalized inertia matrix of the system.

for_kine.m

- Function prototype: `[so sc vc w st]=for_kine(th,dth,n,alp,a,b,bt,dx,dy,dz)`
- This function calculates position of the link origin (so) and COM (sc), velocity of the COM (vc) of the link and angular velocity (w) of the link and the tip position (st).
- As users have access to this function, other points on the link can also be taken as output depending on the system.

ode5.m

- Function prototype: `Y = ode5(odefun,tspan,y0,varargin)`
- It is fixed step solver.

Note:

Users have full access to functions *inputs.m*, *initials.m*, *torque.m*, *jacobian.m*, *plot_en.m*, *plot_motion.m*, *for_kine.m* and *animate.m* whereas the rest are protected codes (pcodes) which can only be used as function.