Hopping Leg Simulator Instructions:

Yanran Ding and João Ramos

1. The Generation Function:

First you need to navigate to the folder "gen" and generate the kinematics and dynamic model by running the function "*gen_dyn_boom_leg_v1.m*". It takes a while for the function to generate the files which will be located in the "gen" folder. In the following test, the "*gen_dyn_boom_leg_v1.m*" function will be referred to as the generation function. Further details are provided below:

The generation function uses the MALTAB symbolic Toolbox. The "define symbol" section defines the symbols that are used in the derivation. *m_list_params* is a list of all parameters. *m_list_q* and *m_list_dq* are the joint angle and joint velocities, respectively. *m_list_dq* is for the foot position at impact.

After the we calculate the Homogeneous Transformation Matrices (HTM), Jacobian matrices, Equation of Motion (EOM) matrices, etc., the *write_fcn_m* function writes a .m file that returns the instantaneous value of that matrix/vector. For instance, the function:

```
write_fcn_m('fcn_p_toe.m',{'q','p'},[m_list_q;m_list_params],{p_toe,'p_toe'});
```

writes a function *fcn_p_toe.m* that takes the joint angles q and the system parameters p to compute the x, y, and z spatial position of the foot. The files

```
write_fcn_m('fcn_De.m',{'q', 'p'},[m_list_q;m_list_params],{De,'De'});
```

```
write_fcn_m('fcn_Ce.m',{'q', 'dq', 'p'},[m_list_q; m_list_dq;m_list_params], {Ce,'Ce'});
```

are the function that return the inertia matrix De and Coriolis vector Ce of the manipulator with instantaneous configuration q and joint velocities dq. The actuation selection matrix Be maps the 2x1 control input vector you have access to (hip and knee torques) to the 4x1 input torque in the EOM. Remember that the first two joints are passive.

The generation function generates the .m files for all the required matrices that are going to be used for the computation of kinematics and dynamics of the manipulator in the main simulator.

2. The MAIN Simulation Function:

The main function for the simulation is named "*MAIN.m*" and is located at the root directory. There are several options that could be set in the MAIN function:

- p.Nstep: number of simulated hops
- p.isMotorDynamics: if the simulation includes the motor dynamics
- p.isControlSaturate: if torque saturation is considered

2.1. Parameters

The function "*get_params.m*" defines the parameters of the system, including:

- p.Tst: Nominal stance time for force parametrization during the stance phase
- p.Rboom: Boom radius
- p.g: gravity constant
- p. Fx_co_bz: Bezier polynomial coefficients for the ground reaction force (GRF)
- p.Kp & p.Kd: proportional & derivative gains for a PD controller
- p.Krh: constant for raibert hopper foot position heuristics
- p.N_animation: parameter for adjusting animation speeds
- p.M_r & p.B_EMF: motor dynamics
- p.uHip & p.uKnee: for control constraints
- Link lengths: HB,LB,DB,LH,LK,DK
- Link masses: M1,M2,M3,M4
- CoM position in link i frame: rxi, ryi, rzi
- Components of inertia tensor of link i: Jxxi, Jyyi, Jzzi, Jxyi, Jxzi, Jyzi

2.2. Dynamics and Transition Events

The hopping leg is a system with hybrid dynamics. This means that the dynamic are interrupted by discrete events (impacts with the grounds) that may cause discontinuous transition of the states (joint velocities). The system has different dynamic equations of motion during flight (foot not touching the ground) and during stance (foot in contact with the ground). The transitions between different dynamics are determined by the triggering of events, specifically, touch down (flight to stance) and lift off (stance to flight).

2.2.1.Flight Phase

The EoM for the flight phase is:

$$D_e(q)\ddot{q} + C_e(q,\dot{q})\dot{q} + G_e(q) = B_e u$$

The actuation selection matrix B_e is defined as

$$B_e = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

To make sure that joints 1 are 2 are passive (you can't apply torque to them). A workspace PD controller is implemented to maintain the foot at a desired location.

$$F_{sw} = \begin{bmatrix} K_{p1} & 0 \\ 0 & K_{p2} \end{bmatrix}(p_d - p) - \begin{bmatrix} K_{d1} & 0 \\ 0 & K_{d2} \end{bmatrix}\dot{p}$$

Then joint torque is calculated through the Jacobian mapping

$$u = J^T(q) \cdot F_{sw}$$

2.2.2.Stance Phase

The EoM for the stance phase is:

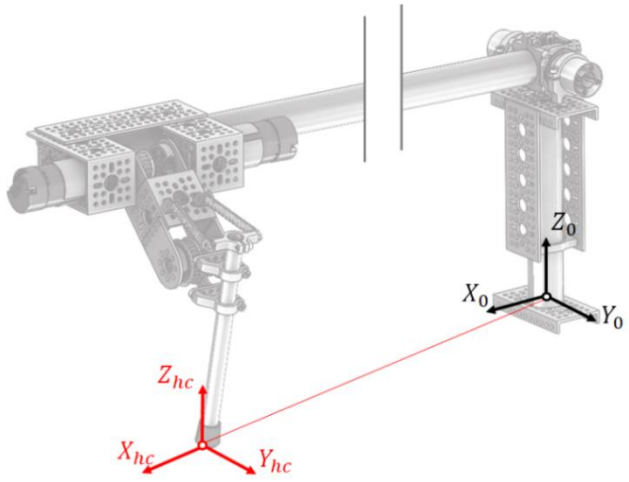
$$D_e(q)\ddot{q} + C_e(q,\dot{q})\dot{q} + G_e(q) = J_{hc}^T F_{GRF} + B_e u$$

$$J_{hc}\ddot{q} + \dot{J}_{hc}\dot{q} = 0$$

The second equation defines the holonomic constraint that the foot is in contact with the ground and cannot move (velocity of the foot is zero):

$$J_{hc}\dot{q} = \begin{bmatrix} J_{Yhc} \\ J_{Zhc} \end{bmatrix} \dot{q} = 0$$

The Jacobian J_{hc} maps the joint velocities to the foot velocity in the directions $Y_{hc}Z_{hc}$. This frame does not necessarily coincide with the foot fixed frame. It changes at every stepping location. The direction X_{hc} is the radial direction between the foot and the boom base. Direction Z_{hc} is vertical and always parallel with Z_0. When the robot steps on the ground, the foot is not allowed to move vertically Z_{hc} or tangentially to the boom rotation Y_{hc}. However, it can move radially towards the boom in the X_{hc} direction. In readl life, the foot always slides a little bit in the X_{hc} direction.



This means that the Jacobian J_{hc} imposes a constraint to the foot motion. Taking the time derivative, we have:

$$J_{hc}\ddot{q} + \dot{J}_{hc}\dot{q} = 0$$

Now we not only have to solve for the joint acceleration $\ddot{q}$ at all the ode45 iterations, but we also need to compute the foot contact forces that make sure that the foot does not move. This model is called “Hard contact” model:

$$\begin{bmatrix} D_e & -J_{hc}^T \\ J_{hc} & 0_{2x2} \end{bmatrix} \begin{bmatrix} \ddot{q} \\ F_{GRF} \end{bmatrix} = \begin{bmatrix} B_e u - C_e\dot{q} - G_e \\ -\dot{J}_{hc}\dot{q} \end{bmatrix}$$

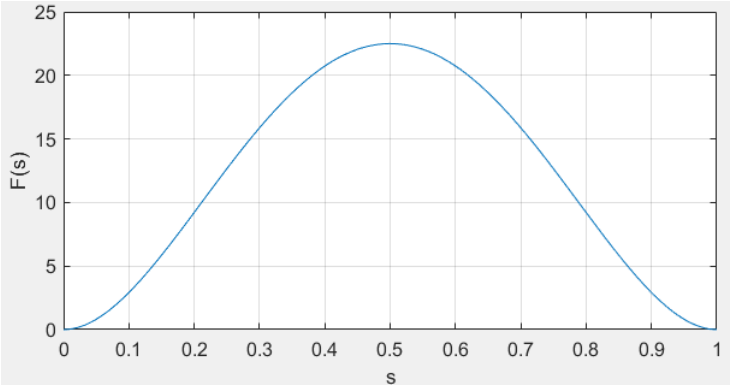
$$\begin{bmatrix} \ddot{q} \\ F_{GRF} \end{bmatrix} = \begin{bmatrix} D_e & -J_{hc}^T \\ J_{hc} & 0_{2x2} \end{bmatrix}^{-1} \begin{bmatrix} B_e u - C_e \dot{q} - G_e \\ -\dot{J}_{hc} \dot{q} \end{bmatrix}$$

The stance controller regulates the contact force between the foot and the ground. You can implement any force profile you want, but currently the for profile follows a Bezier polynomial parametrized in $s \in [0,1]$. The function *polyval_bz(alpha, s)* evaluates the force at instant *s* using Bezier coefficients alpha. Value of s is 0 at the beginning of stance and 1 at the end. For instance, for

```
s = 0:0.01:1;

plot(s,polyval_bz([0 0 60 0 0],s)
```

gives:



The stance controller now simply applies a force upwards to maintain hopping by using the foot Jacobian in respect to the hip J_{HIP}. This Jacobian maps the relative velocity of the foot in respect to the hip. It's the same Jacobian you developed for item 1.f) of milestone #1.

$$u = J_{HIP}^T \begin{bmatrix} F_{Yd} \\ F_{Zd} \end{bmatrix} = J_{HIP}^T \begin{bmatrix} 0 \\ F_{Bezier} \end{bmatrix}$$

You can change it to regulate hopping height, speed or the boom angle θ_1. In addition, a soft joint PD controller is super-imposed to increase the robustness of the stance controller.

2.2.3.Transition Events

The touchdown event occurs when the foot height decreases to 0. This is defined in the function "*event_touchDown(t,X,p)*". The current version of simulation only support flat ground. However, the users are encouraged to modify the touchdown event function to enable the hopping leg to traverse irregular terrains.

Because the foot impacts the ground and doesn't bounce, we need to calculate what is the impact force applied to the robot that changes the velocity of the foot instantaneously. From, the joint velocities before

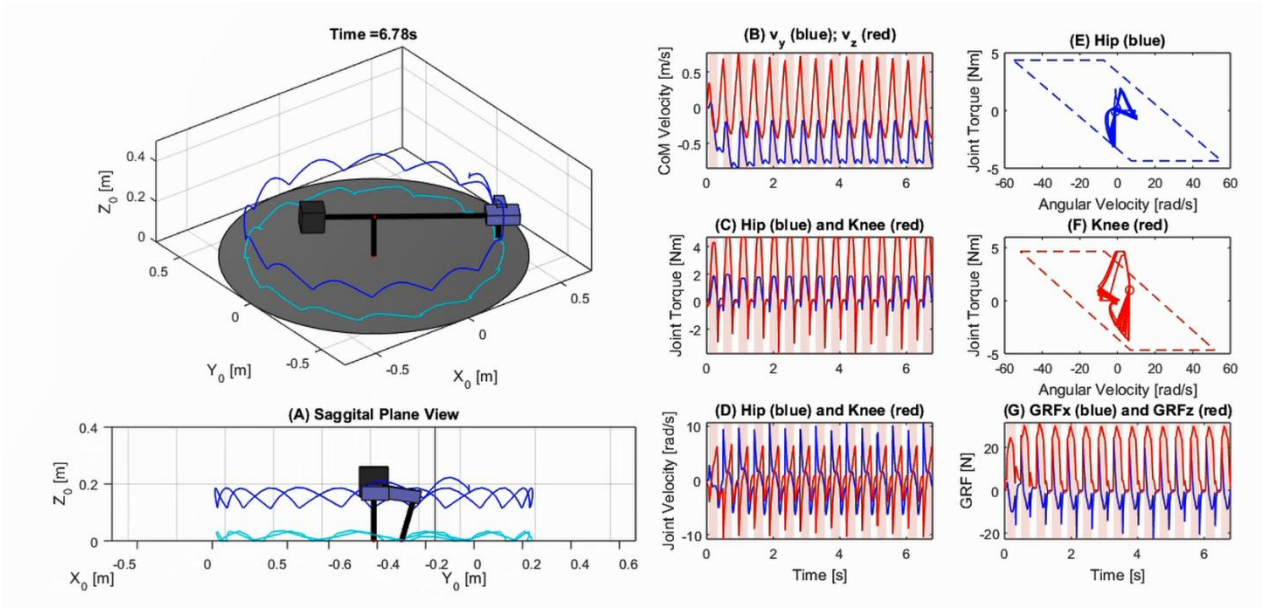
impact $\dot{q}^-$ we need to compute the joint velocities after impact $\dot{q}^+$ and the impact F_{imp}. For that we us the function *X_post = fcn_impactMap(X_prev,p)* to solve the system:

$$M(\dot{q}^+ - \dot{q}^-) = J_{hc}^T F_{imp}$$
$$J_{hc}\dot{q}^+ = 0$$

The liftoff event occurs when the ground reaction force in the vertical direction decreases to zero. For that we must monitor the z-component of F_{GRF} obtained from solving the stance dynamics. In practice, a small threshold is used to declare liftoff event, which helps to increase the robustness of the simulation.

2.3. Animation

The *animateRobot* function takes int the logged data to produce an animation. If the bool value p.boolAnimate was set to 1, than a video file is going to be generated in the current directory, by the default name "video.mp4". The following figure shows a snapshot of a hopping simulation.



3. Summary

The simulation runs for a number of Nstep hops. It starts at flight (foot not touching the ground) and transitions to stance when touch down happens. The pseudocode of what happens is:

```
define q0, q0_dot

for i_steps = 1:N_steps
do:
        air_phase_dynamics (ode45)
                wait for touch_down event
        stance_phase_dynamics (contact dynamics in ode45)
                wait for lift_off event
end

animateRobot
```