Implement Raibert-control of a 3D Hopper in Pybullet

Hang Gao, Wenran Tian

***Abstract*— The spring-mass model has ample profound usages in developing the basic models for understanding the running gait in animals and robots. In this paper, we concern the behavior of a 3D hopping robot by introducing the Raibert-control. Specifically, we separate the research contents into three targets: 1, construct the essential steady forward speed control using the original Raibert method in 2D and 3D dynamic model on even ground. 2, control method for the SLIP model on uneven ground, test the result of Raibert method. 3, based on the result from Raibert-control method, adjust and improve the control method. From of simulation, our final controller provides good results.**

Index Terms—Spring-mass model, Raibert-control, Pybullet, nonlinear dynamical systems.

I. Introduction

Recent years, the spring-mass model has ample profound usages in developing the basic models for understanding the running gait in animals and robots[1]. Specifically, there are extensive studies about developing approximate solutions to conclude the relationship between stability and hybrid dynamics with Poincare maps [2][3]. In the earlier time, there are also studies aim at basic and pragmatic tasks like developing the leg model among humans and animals [4][5]. With the hopper systems become more accurate and robust, we believe there will not be a long time until the hopper systems can have the potential of wide-scale applications like quadruped robots. So, the analysis and simulation of hopper system will be a reasonable project topic nowadays.

In this paper, we concern the behavior of a 3D hopping robot by introducing the Raibert-control. This system separates the forward velocity, hopping height, and body altitude into three decoupling control systems. Like many other studies [6], we restrict attention to the forward speed control rather than the other two—specifically, the feet position of the event at end flight domain (collision moment). The core concept of Raiber-control method to control the forward speed is by changing the position of the feet at the collision moment [7]. From experience, it can provide a pretty stable control output. We will simulate this method, see the result, and concentrate on how to amend the controller to better performance. Besides, we will define the dynamics both in 2D and 3D to help us simulate the controller.

II. Research Question

As implied above, the overarching research question is: simulating the SLIP model by applying Raibert-control in Pybullet, and make the hopper robot on uneven ground and maintain a desired velocity. Based on this final target, we mainly separate the research contents into three targets: 1, construct the essential steady forward speed control by using the original Raibert method in 2D and 3D dynamic model on even ground. 2, control method for the SLIP model on uneven ground, test the result of Raibert method. 3, based on the result from Raibert-control method, adjust and improve the control method.

III. Hopper Model Establishment and Problem Formation

A. Dynamics

We will use the 3D spring-mass model as the previous work [1] like Fig.1. We separate the dynamics into flight phase and stance phase as usual as usual [6]. Then we assume that the leg is massless and the body is a unit point mass.

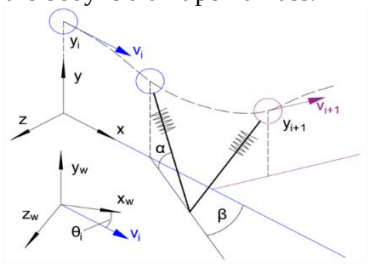


Fig1. 3D spring-mass model. We use the cyan color to denote the ith hop and the purple color to denote the $i+1$th hop. Frame (x_w, y_w, z_w) denotes the fixed world frame. Frame (x, y, z) denote the apex frame, which has its original point on $x_w y_w$ plan, y axis parallel to the y axis of world frame, and its x-axis parallel to hopper's velocity at the apex point. α is the angle between the leg and plan xy at the touch-down moment, and β is the angle between the projection of leg on plane xy and axis-x.

The moving gait of the hopper in flight is the parabola, so we can use formula (1) describe the dynamics in flight phase in apex frame:

$$m\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = \begin{bmatrix}0\\ -mg\\ 0\end{bmatrix} \tag{1}$$

Then we use $[x_p \quad 0 \quad z_p]^T$ to represent the location of the foot point in apex frame during the stance phase. We need to take it for granted that the foot point will not move during the stance phase. We use μ to denote the static friction coefficient, and we have $\alpha \geq \text{atan}\,(\mu^{-1})$.

After we keep the foot point fixed, the vector of the leg $\boldsymbol{l}$ and the force from the leg $\boldsymbol{f_p}$ will be:

$$\boldsymbol{l} = \begin{bmatrix} x - x_p \\ y \\ z - x_p \end{bmatrix}, \boldsymbol{f_p} = k(\frac{l_0}{|\boldsymbol{l}|} - 1)\boldsymbol{l}$$

Then with the gravity item, we can provide the dynamics of the stance phase:

$$m\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = k\left(\frac{l_0}{|\boldsymbol{l}|} - 1\right)\boldsymbol{l} + \begin{bmatrix} 0 \\ -mg \\ 0 \end{bmatrix} \quad (2)$$

B. Behavior function

We will still use the return map at the apex point to describe the behavior of the model.

C. Control

Here we will also decouple handling hopping apex and forward velocity as different problems. Our control target will be [7]: 1. By given the thrust of the leg at the stance phase, compensate the robot to has a certain apex height; 2. By choosing the foot's position at collision moment, compensate the robot to has a certain forward velocity.

We can only control the forward velocity in the stance phase because in our model, because there is no external force during the flight phase. From the Raibert's observation, there is a unique touchdown posture of the leg which can maintain the forward velocity from this flight stance to the next flight stance. This angel is called neutral point. Besides, if and only if the foot touchdown at collision point, can cause the robot has a symmetric resulting in zero net forward acceleration.

Firstly, we assume the movement only occurs in xy plane, i.e., $v_i = \dot{x}$, from the Raibert's approach, we have the control law:

$$\alpha = cos^{-1}\left(\frac{x_f}{|l|}\right) \quad (3)$$

$$x_f = \frac{\dot{v}T_s}{2} + k_{\dot{v}}(\dot{v} - \dot{v}_d) \quad (4)$$

where x_f denotes the forward displacement of the foot with respect to the center of mass, and T_s represents the duration of stance phase. $\dot{v}_d$ is considered as the desired forward speed, $k_{\dot{v}}$ denotes a feedback gain.

Specifically, after the stance phase, we can get the duration time T_s and the forward velocity $\dot{v}$. By using formula (3) and (4), we can get the desired contact angle α of next collusion. We assume that the collusion will only happen in descent period in flight phase, so our control goal is finishing adjust the posture of the fool during the ascent period in flight phase. In addition, in the descent period, the posture of the foot will remain to prepare for the collision. Here we can have the geometry analysis like Fig2.:

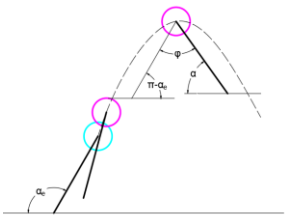


Fig2. The hopper model from the end of the stance phase to the apex in the 2D hopper model. α_e is the angle between foot and ground at the end of the stance phase.

Once we know the initial vertical velocity of flight phase v_{y0}, we will know the time of ascent period is $\tau_{as} = v_{y0}/g$. So, the angular velocity of foot in ascent period is:

$$\omega = \frac{\varphi}{\tau_{as}}$$

Since we restrict attention to the forward velocity control, we assume that there is no energy loss and no change of hopper's spring constant during stance.

In 3D situation, the control law will be, we use $\boldsymbol{q}$ to denote the forward displacement of the foot with respect to the center of mass, i.e., $\boldsymbol{q} = [x_f \quad 0 \quad z_f]^T$:

$$\alpha = cos^{-1}\left(\frac{|\boldsymbol{q}|}{|\boldsymbol{l}|}\right), \quad \beta = sin^{-1}\left(\frac{z_f}{|\boldsymbol{q}|}\right)$$

$$\boldsymbol{q}_f = \frac{\dot{\boldsymbol{v}}T_s}{2} + k_{\dot{v}}(\dot{\boldsymbol{v}} - \dot{\boldsymbol{v}}_d) \quad (5)$$

IV. TECHNICAL APPROACH

A. 2D Simulation

The hopper in 2D has more straightforward dynamics and required much less computational resources, so we will simulate the hopper in 2D firstly, test the controller and try to get some inspiration to implement the 3D controller.

In the simulation, we mainly use packages in python to help us execute a discrete-time state simulation. For computational convenience, our system state will be in cartesian coordinate ($[x \quad \dot{x} \quad y \quad \dot{y}]^T$) in the flight phase, and be in polar coordinate ($[r \quad \dot{r} \quad \theta \quad \dot{\theta}]^T$) in the stance phase. This will change the dynamics in the stance formula (2) to:

$$\ddot{r} = r\dot{\theta}^2 - \frac{k}{m}(r - |\boldsymbol{l}|) - gsin\theta \quad (6)$$

$$\ddot{\theta} = \frac{-2\dot{r}\dot{\theta} - gcos\theta}{r} \quad (7)$$

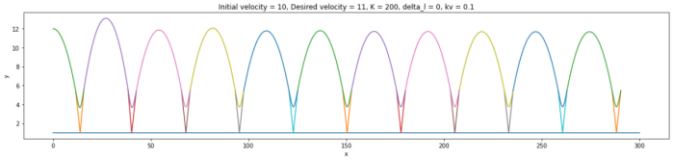
Where θ is the angle between leg and forward speed axis x, i.e., $\theta = \pi - \alpha$; r is the length of leg during the stance phase so that $r = |\boldsymbol{q}|$.

We define the hopper from starting flight phase to end of stance phase as an iteration. Then we will set a loop in program to iterate the iteration. Specifically, we will have the following steps in program:

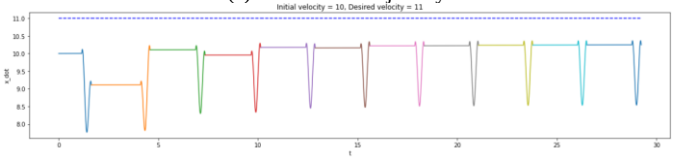
1. Implement the kinematics and dynamics functions (1), (6), (7) to the python function. And implement the forward velocity controller formula (5) to function.
2. Use the solve_ivp method from scipy.integrate package, integrate the dynamics function both in flight phase and stance phase in an iteration.
3. At the end of each iteration, use the function from formula (5) to update the desired x_f
4. Maintain a numpy array from numpy package to save the state series for each iteration.

After implementing the steps above, and several times of parameter adjustment, we can have the trajectory of mass center of hopper like Fig3(a), and a versus between desired velocity and real velocity like Fig3(b). From Fig3(a), we can see that the original Raibert-control can provide us a stable output on an even ground simulation, but Fig3(b) also shows the system has

an obvious steady-state error. That is not unreasonable because, from formula (5), the forward speed controller in Raibert-control is quite concise with only duration time item and error item. Furthermore, there is a lot of room for improvement.



(a) Mass center trajectory



(b) Time-velocity diagram

Fig3. The result of the 2D hopper with Raibert-control. The simulator iterates 11 times on even ground. In (a), the blue straight line represents the ground; the colored arcs represent the trajectory of the mass center of the hopper; and the v-shape elements which connect the ground and arcs are the poses of leg at start and end for each stance phase. In (b), the blue dotted line is the desired velocity; the colored lines are the real velocities in each iteration.

B. Improved Raibert-control

Here we will discuss how we can compensate for the steady-state error by amending the original Raibert-control.

The PID controller is an extremely common filter with a long history [8]. And there is a wealth of experience in the industry, so we will try to get inspired by it. In the experiment of the PID controller [9], the steady-state error of system will disappear if the integral term is added properly. This inspired us to add an integral item to our Raibert-control that:

$$x_f = \frac{\dot{v}T_s}{2} + k_{\dot{v}}(\dot{v} - \dot{v}_d) + k_I \int (\dot{v} - \dot{v}_d)dt \qquad (8)$$

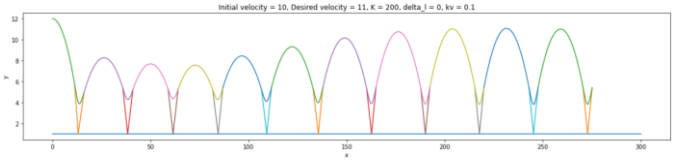
This is our new controller formation in continues-time. As we can see, the formula (8) will be formula (4) add an integral term $k_I \int (\dot{v} - \dot{v}_d)dt$. The controller is now pretty much like the classical PID controller for it only misses a differential term. In this paper, we will call this improved Raibert-control as IRC.

For the implementation of integral term in program, we need do the following steps:

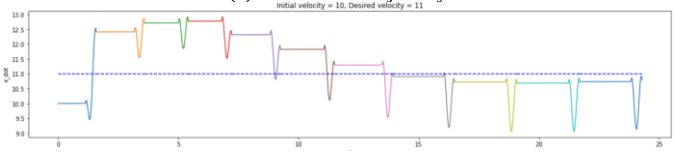
1. Maintain a queue to save the history of velocity in flight phase.
2. Subtract all the values in queue with the desired velocity, and sum them up.
3. Multiply the sum up value with feedback gain k_I
4. Simulation and turn the value of k_I and the length of a queue.

Because we do not know what is the result of system after adding this integral term, we decide to just use a queue to store the velocity, tune and see after the simulation. As we can see, if the length of queue equal to 1, the integral term will shrink to proportional term like $k_{\dot{v}}(\dot{v} - \dot{v}_d)$; if the length of queue equals to infinity, the integral term will become the original integral term that combine all the errors in the classical PID controller.

Then, after tuning and adjustment, we can get the result of the 2D hopper with IRC as Fig4. Firstly, the system has overshoot, then the curve will vibrate, eventually converge to near the desired velocity. This is a quite common system respond result. Although the system responding time has increased, the steady-state error of system is compensated reasonably. So, we believe this IRC is promising and decide to give further experiments.



(a) Mass center trejectory



(b) Time-velocity diagram

Fig4. The result of the 2D hopper with Raibert-control. The simulator iterates 11 times on even ground.

We tested IRC with the same initial velocity to different desired velocities that v_d={5, 6.5, 8}. Each velocity we will simulate six times. Furthermore, to add some randomness, we will run the hopper on uneven ground. Finally, we will get the results in Fig5.

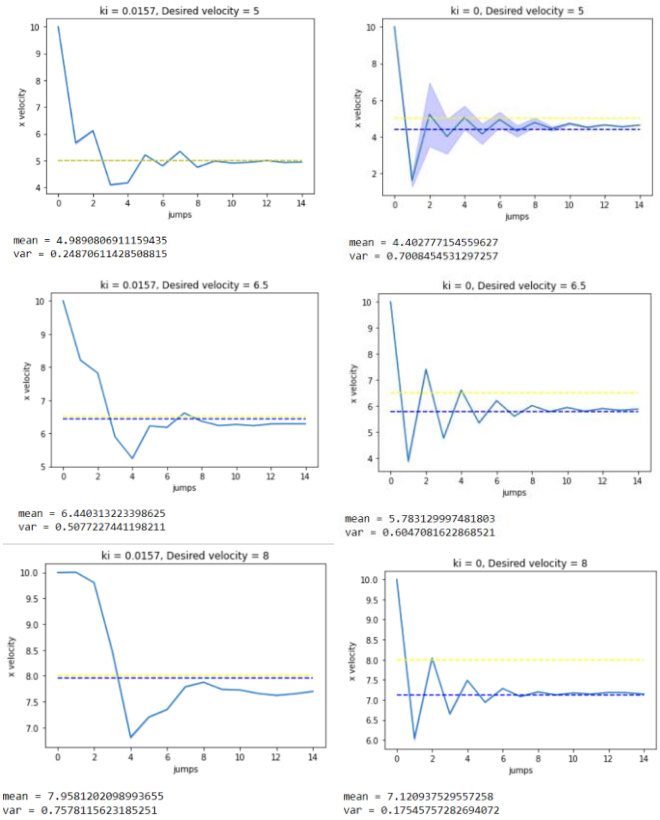


Fig5. The result of the 2D hopper with IPC in different velocities. The charts in the left are the results from IPC, and the charts in the right are the results from original Raibert-control (ki = 0). The blue line stands for velocity in flight phrase during each hop. The blue dash line stands for desired velocity, and the yellow dash line stands for mean value of velocity for all hops (except the first one).

From the Fig5, IPC can give a less error and even less variance in tracking the desired velocity in simulation. we can hold the opinion that IPC may give a better performance than

original Railert-control, and we decide to implement the IPC in 3D and execute further experiments.

C. 3D simulation in Pybullet

Compared with 2D simulation, there are more details in URDF model and specific control strategy.

As in URDF, to have two degrees of freedom, to be specific, we use two links on the hip structure to make the hopper has the freedom of roll and pitch. Having these two degrees of freedom, makes the hopper keep the balance of its mass body (mass disk) when the hopper moves in the x-direction and y-direction.

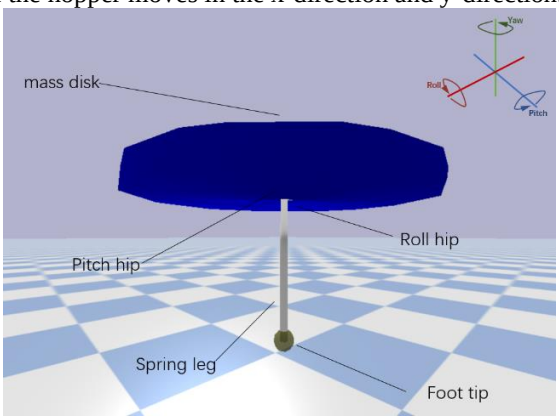


Fig6. The 3D hopper structure model. We construct this physical model in URDF, assuming the mass disk is the main mass body with 100 units, and the others' mass are all 1 unit.

Here, we introduce the moment of inertia, take cylinder for instance,

$$\rho = \frac{M}{\pi R^2 h} \tag{9}$$

where ρ denotes the cylinder's density; R denotes the radius; M represents mass, h is height. Therefore, we derive the moment of inertia tensor:

$$I = \int_V \rho(x,y,z) \begin{bmatrix} y^2+z^2 & -xy & -xz \\ -xy & x^2+z^2 & -yz \\ -xz & -yz & x^2+y^2 \end{bmatrix} dxdy\,dz$$
$$= \begin{bmatrix} \frac{1}{12}Mh^2 + \frac{1}{4}MR^2 & 0 & 0 \\ 0 & \frac{1}{12}Mh^2 + \frac{1}{4}MR^2 & 0 \\ 0 & 0 & \frac{1}{4}MR^2 \end{bmatrix} \tag{10}$$

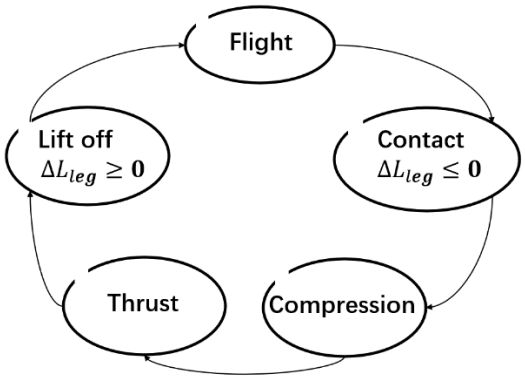


Fig7. The 3D hopper dynamic phase. We define the compression value as a judgment of contact event.

In the dynamic phase model, because the hopper is moving on uneven ground. Thus, we change the contact event algorithm by checking the value of leg position compared to the ground to the algorithm by checking the spring leg's relative compression value. Furthermore, to vanish the influence of the system perturbation on compression leg, we choose the largest noise value in the simulation to construct the checking strategy as a threshold value problem.

As for the controller, we mainly follow the idea from Raibert [7]. Owing to the time limitation and the uneven ground situation, we set the force to the hopper of force thrust constant. Thus, the hopping height controller is much simpler.

$$F_s = k_s(r_0 - r) + F_{thrust} \tag{11}$$
$$F_{thrust} = const \tag{12}$$

Then, for the forward speed controller, instead of using the torque on the hip to control the real-time roll and pitch hip joint position γ close to the desired roll and pitch hip joint position γ_d, we use the position controller function in Pybullet to make the real-time γ close to γ_d.

$$x_f = \frac{\dot{x}T_s}{2} + k_{\dot{v}}(\dot{x} - \dot{x}_d) \tag{13}$$
$$\tau = -k_p(\gamma - \gamma_d) - k_v\dot{\gamma} \tag{14}$$

where x_f is the forward displacement of the foot to the center of mass, and T_s represents last duration of stance phase. $\dot{x}$ is considered as estimate horizontal velocity, and $\dot{x}_d$ denotes desire velocity. γ is called as roll hip joint position and pitch hip joint position in B frame, γ_d is desire roll hip joint position and desire pitch hip joint position in B frame, $\dot{\gamma}$ denotes roll hip joint velocity and pitch hip joint velocity in B frame, k_p, k_v represent position and velocity feedback gains.

When we apply the controller of body attitude, similarly, instead of using the original model (13), we apply the straighter method (14) by velocity controller of Pybullet.

$$\tau = -(-k_p(\theta - \theta_d) - k_v\dot{\theta}) \tag{13}$$
$$\tau = (-k_p(\theta - \theta_d) - k_v\dot{\theta}) \tag{14}$$

where θ represents roll and pitch angular, θ_d denotes the desired roll and pitch angular. $\dot{\theta}$ is considered as roll and pitch angular velocity. k_p, k_v represent position and velocity feedback gains, which values are different with forward speed controller's.

D. Improved Raibert-control in 3D

From the IPC in 2D formular (8), we can give the forward velocity control in 3D similarly:

$$\boldsymbol{q}_f = \frac{\dot{\boldsymbol{v}}T_p}{2} + k_{\dot{v}}(\dot{\boldsymbol{v}} - \dot{\boldsymbol{v}}_d) + k_I \int (\dot{\boldsymbol{v}} - \dot{\boldsymbol{v}}_d)dt \tag{15}$$

The implement of this controller will be almost the same as we described in 2D IPC. The difference is the velocity will be a 2D vector that contains velocity in x-axis and y-axis. And because the structure of hopper is symmetric, we can just apply one ki value to both x-axis and y-axis.

By designing the hopper' trajectory, we can look at the speed controller more visually and intuitively. We set the hopper to move forward with a positive direction and then turn around to the negative direction with the desired speed; finally, the hopper

turns back to the positive direction with the positive desired speed.

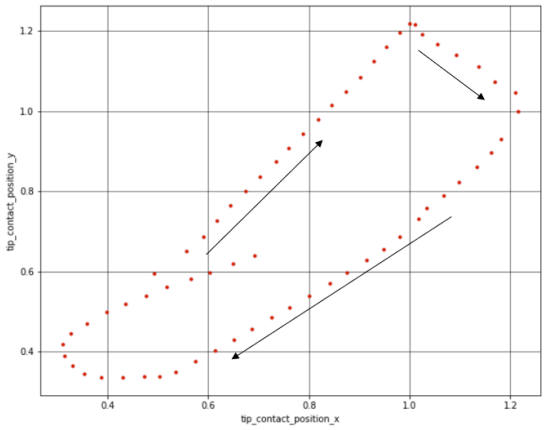


Fig8. The trajectory of the hopper. The red dot is the record of contact event.

By applying the IPC, we can first inspect the performance between the original speed controller and the newer speed controller on even ground.

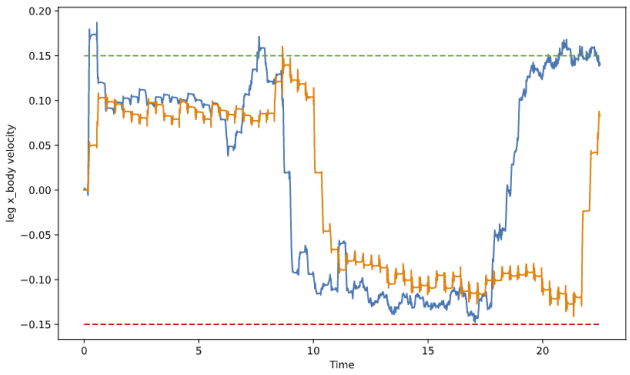


Fig9. The velocity map on even ground. The green dash line and red dash line are the desired velocity of the hopper. The solid yellow line is the result from original controller, and the solid blue line is the result from IPC.

Based on the velocity figure above, we can conclude, as in part B, the newer controller can kill off the error (between the desired speed and real-time speed for the mass body) better.

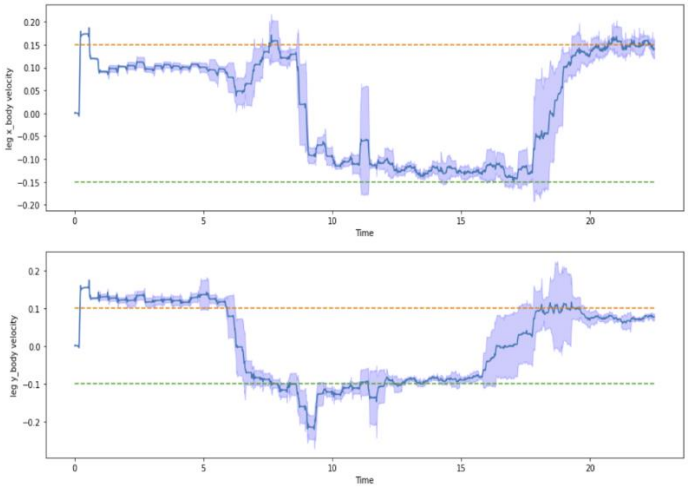


Fig10. The std velocity map with six sets. The solid blue line is the mean value, and the fill region is the std of them.

Fig10 shows that the newer speed controller has an excellent robust performance on closing the desired speed. However, the gap between desired velocity and real-time velocity on the top left of Fig10 will be killed off immediately after the hopper turned around and vanished when the hopper turned back to the positive direction. According to speculation, it is something because of the controller's over adjustment. Due to time and content constraints, we are prepared to discuss this issue in our future work.

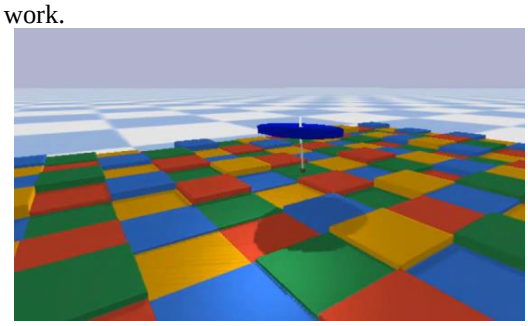


Fig11. The hopper is running on uneven ground with 26% leg height.

For the uneven ground environment, the hopper can move on rough ground with maximum height difference $\Delta H_{\max} = 0.04 - 0.001 = 0.039$ units and the original hopper leg length $L_{\text{leg}} = 0.15$ units.

$$\frac{\Delta H_{\max}}{L_{\text{leg}}} = 26\%$$

Therefore, this hopper can run steadily at 26% leg height.

V. Conclusion and Prospect

In this paper, we use solve_ivp function from scipy.integrate package to simulate the hopper in 2D, and pybullet package to simulate hopper in 3D. They all give reasonable results.

From the simulation, we can give the conclusion that the original Raibert-control can provide a stable output in forward speed control. And the IPC (Improved Raibert-control), which is the Raibert-control add with an integral term, can give a better performance in tracking the desired forward velocity. For it can give a result has a closer mean value to desired velocity. But the system responding time can be longer.

As for the increase of system responding time, this is also can be seen in PID controller. In the PID controller, with a larger feedback gain in integral term, the system corresponding time will also increase. For the system requires less responding time, still we can get inspiration from PID controller. That is, add a differential term to the IRC. We believe this is worth to be studied furtherly.

References

[1] A. Wu and H. Geyer, "The 3-D Spring–Mass Model Reveals a Time-Based Deadbeat Control for Highly Robust Running and Steering in Uncertain Environments," in IEEE Transactions on Robotics, vol. 29, no. 5, pp. 1114-1124, Oct. 2013, doi: 10.1109/TRO.2013.2263718. W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[2] Schwind, W., Koditschek, D. Approximating the Stance Map of a 2-DOF Monoped Runner. J. Nonlinear Sci. 10, 533–568 (2000). https://doi.org/10.1007/s004530010001

[3] U. Saranli, O¨ . Arslan, M. M. Ankaral, and O¨ . Morgu¨l, “Approximate analytic solutions to non-symmetric stance trajectories of the passive springloaded inverted pendulum with damping,” Nonlinear Dyn., vol. 62, no. 4, pp. 729–742, 2010.
[4] R.Alexander, “Three uses for springs in legged locomotion,” Int. J. Robot. Res., vol. 9, no. 2, pp. 53–61, 1990.
[5] M. Dickinson, C. Farley, R. Full, M. Koehl, R. Kram, and S. Lehman, “How animals move: An integrative view,” Science, vol. 288, no. 5463, pp. 100–106, 2000.
[6] W. J. Schwind and D. E. Koditschek, "Control of forward velocity for a simplified planar hopping robot," Proceedings of 1995 IEEE International Conference on Robotics and Automation, Nagoya, Japan, 1995, pp. 691-696 vol.1, doi: 10.1109/ROBOT.1995.525364.
[7] Raibert, M. H. 1986. Legged Robots That Balance. Cambridge, MA: MIT Press.
[8] Bennett, S. (1993). Development of the PID controller. IEEE Control Systems Magazine, 13(6), 58-62.
[9] Åström, K. J. Control system design lecture notes for me 155a. Department of Mechanical and Environmental Engineering University of California Santa Barbara, 333. pp216-220, 2002.