Report:

Dodo Alive!
Resurrecting the Dodo with Robotics and AI

Dynamic Locomotion of simplified hopping robots

Jingkun Feng, Matthias Lehner and Jean-Pascal Lutze

Supervision: Dennis Ossadnik

March 31, 2021

Contents

1 Introduction

1.1 Motivation

Bio-mechanical research has shown that the dynamics of the center of gravity (CoG) in running humans closely resemble the dynamics produced by the model of a spring loaded inverted pendulum (SLIP). The SLIP model combines a spring with a point mass on top. The motion of the CoG of a SLIP model does not only approximately represent the CoG motion in a large variety of animals and humans but also produces ground reaction forces that closely resemble the forces observed in experimental recordings. Due to it's simplicity yet powerful expressiveness the SLIP model quickly got accepted as the de-facto standard for gait-simulation tasks and found widespread application in gait research.[2] Unlike the SLIP model, all biologic and most robotic legs do not express prismatic

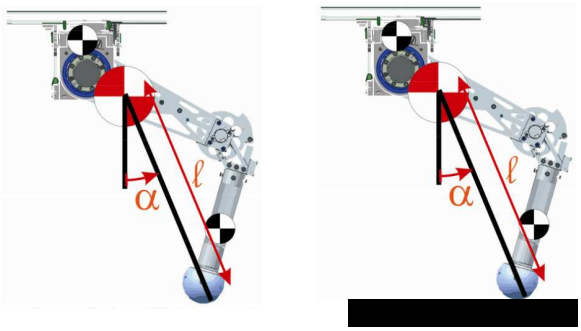

Figure 1: Left: A free floating robot leg. The red mass center is the overall center of mass of the robot, α is the angle of attack and l is the theoretical length of the spring between the foot and the CoG. Right: Robot leg in contact with the ground. Edit from: [2]

elasticity but instead have a segmented structure with elasticities that generates a torque in the leg's joints or through the control of the motors in the joints. Additionally, in a real robotic leg as well as in biological limbs, the mass and inertia of the leg's segments couple the motion of the segments, and lead to energetic losses at the impact-collisions at touchdown. These are two effects that have a large influence on the CoG motion but are not accounted for in the classic SLIP model. To close the bridge between real robotic legs and the theoretical foundation provided by the SLIP model, Marco Hutter et al.[2] propose a robot control and abstraction approach that manages to fully project the dynamics of the SLIP model onto actual segmented robotic legs. In this seminar, we implemented the results of that research in the simulation of a self-stabilizing, hopping robot.

1.2 Problem Statement

The challenge was to implement the simulation of an one-legged, continuously hopping robot. When the simulated model is subject to deflection from it's stationary hopping it should first be able to stabilize itself to the point of hopping stationary again and then move back to the position it inhabited before the deflection. Additionally, the implementation was supposed to be constrained to two-dimensional space and follow the robot control and abstraction approach proposed by Marco Hutter et al. This is described in more detail in the following sections.

2 Mathematical Notation in this Report

In the equations in the upcoming sections we are using the following notation.
Scalars are denoted using non-bold lowercase letters.

$$s \in \mathbb{R}$$

Vectors are denoted using bold lowercase letters.

$$\mathbf{v} \in \mathbb{R}^m = \begin{pmatrix} v_0 \\ \vdots \\ v_{n-1} \end{pmatrix}$$

Matrices are denoted using bold uppercase letters.

$$\mathbf{M} \in \mathbb{R}^{m \times n} = \begin{pmatrix} a_{0,0} & \cdots & a_{0,n-1} \\ \vdots & \ddots & \vdots \\ a_{m-1,0} & \cdots & a_{m,n-1} \end{pmatrix}$$

3 Theoretical Foundation

The theoretical foundation for our implementation is based on the paper "SLIP running with an Articulated Robotic leg" by Marco Hutter et al. [2]. In the paper, the authors propose an operational space controller that projects the behavior of the SLIP-model onto the dynamics of an actual segmented robotic leg. The dynamics of the center of gravity (CoG) in running gaits (which include bounding, trotting, or galloping)[1] can be described by the model of a spring loaded inverted pendulum, which combines a mass-less spring leg with a point mass on top. It has been proven in various experiments that the motion of the center of gravity closely resembles the motion of a large variety of gaited animals and humans and produces ground reaction forces that closely match the biological counter parts[1]. Since the SLIP-model does not have into account the energetic losses that occur for actual segmented robotic legs on ground contact, the authors also introduce methods to compensate theses losses. This in turn allows the application of existing dead-beat control strategies to arbitrary robotic legs.

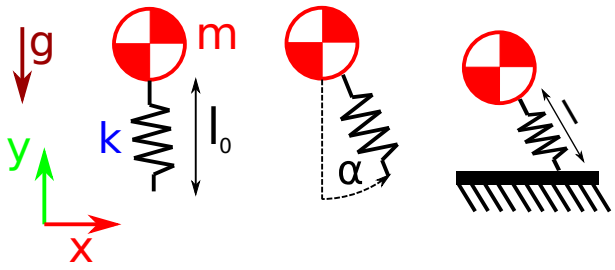

Figure 2: The SLIP model is composed of a mass less spring on which a point mass is fixed. The spring have the spring stiffens k and the initial spring length l_0. If the spring is compressed it actual length is written as l. The rotational angle of the spring around of the mass is called angle of attack α.

4 Implementation

4.1 Initial Model

The simulation for our hopper is implemented using the C++ programming language and the Rigid Body Dynamics Library (RBDL) [4]. RBDL is a library of rigid body dynamics algorithms such as the Articulated Body Algorithm for forward dynamics, Recursive Newton-Euler Algorithm for inverse dynamics and the Composite Rigid Body Algorithm for the computation of the joint space inertia matrix. Additionally, the library also provides tooling to calculate Jacobians, handling contact and collisions constraints and closed-loop models. The definition of the robot itself is done using the Lua-language and looks like the following:

```
linkProperties= {mass = 1., com = {0.,-1.,0.}, inertia = inertiaMatrix}
baseProperties= {mass = 10., com = {0.,0.,0.}, inertia = nullMatrix}
footProperties= {mass = 0.001, com = {0.,0.,0.}, inertia = nullMatrix}

model = {
    gravity = {0., -9.81, 0.},
    frames = {
        {   name = "floatingBase",
            parent  = "ROOT",
            visuals = {meshes.floatingBase},
            body    = bodies.floatingBase,
            joint   = joints.floating,
            joint_frame = {r = {0, 0, 0},}
        },
        {  name    = "link1",
            parent  = "floatingBase",
            visuals = {meshes.link},
            body    = bodies.link,
            joint   = joints.rotational_z,
            joint_frame = {r = {0,-1, 0},}
        },
        {   name    = "link2",
            parent  = "link1",
            visuals = {meshes.link},
            body    = bodies.link,
            joint   = joints.rotational_z,
            joint_frame = {r = {0,-1, 0},}
        },
        {   name    = "foot",
            parent  = "link1",
            joint_frame = {r = {0,-1, 0},}
        },
    }
}
```

The model is composed of two mass-afflicted links connected by a rotational joint. The upper link in the leg is connected to a mass simulating the torso of a robot with a floating joint. As the simulation is for the time being bound to two-dimensional space, the model also can only move in the

XY-plane. For visualizing the model we are using MeshUp [3], a visualization tool for RBDL-based rigid multi-body systems that can render the real time motion of the simulated model as well as draw the trajectories executed during the simulation. An example of that visualization can be seen in figure 3.

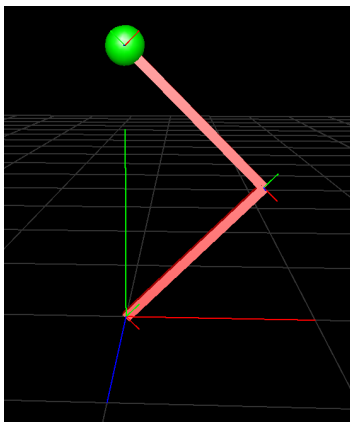

Figure 3: RBDL leg model consisting of two mass-afflicted links, two joints and a mass on top visualized in MeshUp [3]

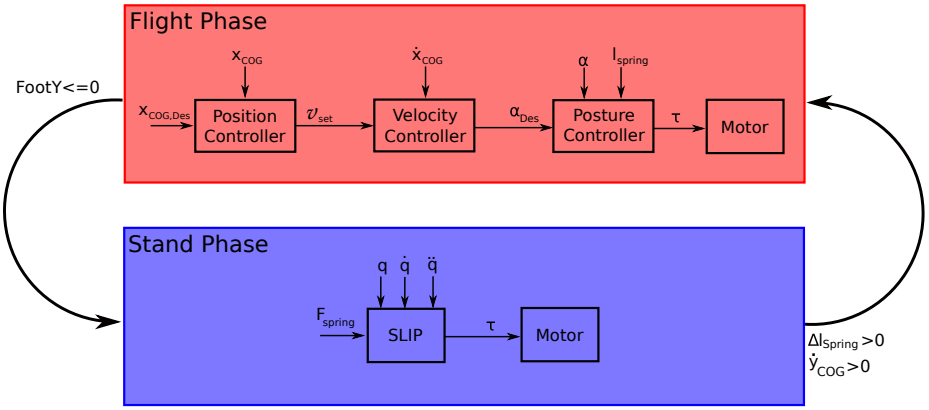

Figure 4: The life cycle of our model is split into two phases: The Stand Phase and the Flight Phase. In the Stand Phase, the model's dynamics are bound to the behavior of the SLIP model with a given position (q), velocity ($\dot{q}$) and acceleration $\ddot{q}$. In the Flight Phase, the model's position is controlled by the implemented position controller which determines a desired velocity for the velocity controller. This controller in turn determines which posture (angle of attack and leg length) the robot needs to obtain to accelerate our leg model to the desired velocity.

4.2 SLIP Behaviour in an Articulated Leg

As mentioned before, SLIP stands for **S**pring **L**oaded **I**nverted **P**endulum. A SLIP model consists of a mass mounted on a spring, see figure 2. If the spring is in contact with the ground the spring creates a force $\mathbf{F}_{leg} = k(l_0 - l)$ depending on the length of the spring in rest l_0 and the actual spring length l. The generated force depends also on the spring stiffness k. In addition the gravitational acceleration $\mathbf{g}_c$ acting on the mass m.

$$m\ddot{\mathbf{r}}_{SLIP} = \mathbf{F}_{leg} + m\mathbf{g}_c \tag{1}$$

For the stance phase equation (1) describes the forces acting on the center of gravity $\mathbf{r}$. For a hopping robot we have two phases the robot can be in, see figure 4. The first one is the flight phase in which the robot is airborne without contact to the ground. In this phase only the gravitational forces act on the robot. The second phase is the stand phase where the robot's foot is in contact with the ground. This is the relevant phase for the implementation of SLIP-behavior as it is the only phase in which the leg can propel itself through the force between the ground and the CoG. It is assumed that due to the ground contact the velocity as well as the acceleration of the foot are zero $\dot{\boldsymbol{x}}_{\boldsymbol{s}} = \boldsymbol{J}_s\dot{\boldsymbol{q}} = 0$ and $\ddot{\boldsymbol{x}}_s = \boldsymbol{J}_s\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_s\dot{\boldsymbol{q}} = 0$ where $\boldsymbol{x}_s$ is the position of the foot. With these conditions we can eliminate $\mathbf{F}_s$ from the dynamics equation

$$\mathbf{M}\ddot{\boldsymbol{q}} + \mathbf{b} + \mathbf{g} + \mathbf{J}_s^T\mathbf{F}_s = \mathbf{S}^T\boldsymbol{\tau} \tag{2}$$

as it is shown in [5]. Then we obtain

$$\mathbf{M}\ddot{\boldsymbol{q}} + \mathbf{N}_s^T(\mathbf{b} + \mathbf{g}) + \mathbf{J}_s^T\boldsymbol{\Lambda}_s\dot{\boldsymbol{J}}_s\dot{\boldsymbol{q}} = (\mathbf{S}\mathbf{N}_s)^T\boldsymbol{\tau} \tag{3}$$

The Coriolis and centrifugal torques are combined in the vector $\mathbf{b}$. The torques induced by the gravitation are represented by the vector $\mathbf{g}$. Here $\boldsymbol{\Lambda}_s = \left(\mathbf{J}_s\mathbf{M}^{-1}\mathbf{J}_s^T\right)^{-1}$ is the support space inertia matrix and $\mathbf{N}_s = \left[\mathbf{I} - \mathbf{M}^{-1}\mathbf{J}_s^T\boldsymbol{\Lambda}_s\mathbf{J}_s\right]$ is the nullspace which is used to calculate the change of the joint velocity $\dot{\boldsymbol{q}}^-$ before and $\dot{\boldsymbol{q}}^+$ after the ground impact.

$$\dot{\boldsymbol{q}}^+ = \boldsymbol{N}_s\dot{\boldsymbol{q}}^- \tag{4}$$

For a better understanding we imagine that the robot from figure 1 is in free fall with it's joints moving arbitrarily. In the moment the robot foot acquires ground contact the foot itself is stopped from moving freely while the rest of the robot is still in motion. Here the nullspace matrix give us the velocities of the joints after the impact. Since we want to control the robot's center of gravity in Cartesian coordinates (and not in joint space) we use an operational space controller. This operational space controller projects the robot's dynamics equation (2) into Cartesian space.

$$\boldsymbol{F} = \boldsymbol{\Lambda}^*\ddot{\mathbf{r}}_{cog} + \boldsymbol{\mu}^* + \boldsymbol{p}^* \tag{5}$$

The force $\boldsymbol{F}$ depends on the projected gravitational parts $\boldsymbol{p}^*$, the projected Coriolis and centrifugal parts $\boldsymbol{\mu}^*$ as well as the task inertia $\boldsymbol{\Lambda}^*$ given an acceleration of the center of gravity $\ddot{\mathbf{r}}_{cog}$. The definition of the terms $\boldsymbol{\mu}^*$, $\boldsymbol{\Lambda}^*$ and $\mathbf{J}^*$ can be found in [2] and appendix A.1. With the transpose Jacobian we can project these forces onto the torques $\boldsymbol{\tau}$ of our joints.

$$\boldsymbol{\tau} = \mathbf{J}^{*T}\mathbf{F} \tag{6}$$

If we combine this equation with equation:(5) we get a controller in task space given a Cartesian acceleration. Since we want to control the robot like a SLIP model we use equation (1) as our acceleration of the center of gravity.

$$\boldsymbol{\tau} = \mathbf{J}^{*T} \left(\boldsymbol{\Lambda}^* \frac{1}{m} \left(\mathbf{F}_{leg} + m\mathbf{g} \right) + \boldsymbol{\mu}^* + \boldsymbol{p}^* \right) \tag{7}$$

Figure 5 shows the implementation of equation (7) where the robot model falls down from an initial elevated position. On ground contact, the model behaves like a spring.

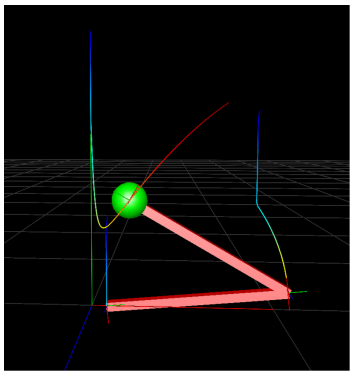

Figure 5: RBDL leg model consisting of two mass-afflicted links, two joints and a mass on top visualized in MeshUp [3], behaving like the SLIP model. In the image, the model is dropped onto the ground, expressing the behavior of a mass on a spring. The movement of the floating base is visualized by the path.

Since an articulated robot leg, unlike the SLIP model, will lose energy at the impact we have to compensate this lost energy to achieve continuous hopping. The authors of the paper solved this problem with pre-compressing the virtual leg spring at touch down with a different spring leg length before l_0^- and after l_0^+ impact. The equation $l_0^+ = l_0^- + \Delta l$ holds. The lost kinetic energy is compensated with the difference of the spring leg length Δl. This difference together with the stiffness k yields the energy ΔE.

$$\Delta E = 0.5 k \Delta l^2 \tag{8}$$

This energy has to match the lost kinetic energy of the center of gravity, which is the difference of the kinetic energy before and after the impact.

$$\Delta E = 0.5 m_{cog} \left(|\dot{\boldsymbol{r}}_{cog}^-|^2 - |\dot{\boldsymbol{r}}_{cog}^+|^2 \right) \tag{9}$$

With (4) we can calculate the difference of energy in the joint space.

$$\Delta E = 0.5 m_{cog} \dot{\boldsymbol{q}}^{(-)T} \left(\mathbf{J}_{cog}^T \mathbf{J}_{cog} - \mathbf{N}_s^T \mathbf{J}_{cog}^T \mathbf{J}_{cog} \mathbf{N}_s \right) \dot{\boldsymbol{q}}^- \tag{10}$$

The $\mathbf{J}_{cog}$ in (10) is the Jacobian of the center of mass. It is weighted sum of Jacobian of all n bodies in the robot (11). The weight is calculated by dividing the mass of the body m_i by the total mass M of the robot (12).

$$\mathbf{J}_{cog} = \sum_{i=0}^{n} w_i \mathbf{J}_{cog,i} \tag{11}$$

$$w_i = \frac{m_i}{M}, \quad i = 1, ..., n \tag{12}$$

This energy will be added with the pre-compressed spring to the model. With this method the system does not lose energy over time.

4.3 Posture Controller

After implementing the SLIP behavior and the energy compensation, our robot behaves like a spring mass system. To have our robot generate stable hopping patterns, we also need to control it's foot position so our robot is able to produce accelerations allowing it to change the actual velocities. Therefore, we implemented a controller to control the angle of attack and the spring length between the CoG and the foot. Let's get some insight into how that controller works. First we assume that our torques:

$$\boldsymbol{\tau} = \mathbf{M}\ddot{\mathbf{q}} \tag{13}$$

and our forces in Cartesian space are:

$$\mathbf{F} = \mathbf{M}_{ee}\ddot{\mathbf{x}} \tag{14}$$

where $\ddot{\mathbf{x}}$ is the acceleration of our foot in Cartesian coordinates and $\mathbf{M}_{ee}$ the mass matrix mapped into Cartesian space. For our force we use an PID controller where K_p, K_d, K_i are the gain values.

$$\mathbf{F} = K_p\mathbf{e}_p + K_d\dot{\mathbf{e}}_p + K_i\Delta t\sum \mathbf{e}_p \tag{15}$$

Here $\mathbf{e}_p = \mathbf{x}_{des} - \mathbf{x}_{foot}$ is our error between the position of our foot $\mathbf{x}_{foot}$ and the desired position of the foot. Next we have to calculate the connection between $\ddot{\mathbf{q}}$ and $\ddot{\mathbf{x}}$. For that we use the derivative of the equation $\dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}}$.

$$\ddot{\mathbf{x}} = \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{J}\ddot{\mathbf{q}} \tag{16}$$

For small changes of the robot configuration, we have $\dot{\mathbf{J}} \approx 0$ so we get $\ddot{\mathbf{x}} = \mathbf{J}\ddot{\mathbf{q}}$. To get our joint acceleration we reformulated the equation to:

$$\ddot{\mathbf{q}} = \mathbf{J}^{\#}\ddot{\mathbf{x}} \tag{17}$$

Where $\mathbf{J}^{\#}$ is the pseudo inverse of the Jacobian. We can now combine our equations (13) and (17) to calculate the torque

$$\boldsymbol{\tau} = \mathbf{M}\mathbf{J}^{\#}\ddot{\mathbf{x}} \tag{18}$$

Next we replace $\ddot{\mathbf{x}}$ with the result from our PID controller (15) and the reformulated equation for the occurring forces (14):

$$\boldsymbol{\tau} = \mathbf{M}\mathbf{J}^{\#}\mathbf{M}_{ee}^{-1}(K_p\mathbf{e}_p + K_d\dot{\mathbf{e}}_p + K_i\Delta t\sum \mathbf{e}_p) \tag{19}$$

To get a good control-behavior we set the K_p, K_i, K_d values after each other and analysed the behaviour. Figure 6 shows the robot falling down from an initial elevated position. The goal angle of attack is zero. As you can see the foot is moving from its starting position to the desired position while falling down, see figure 7.

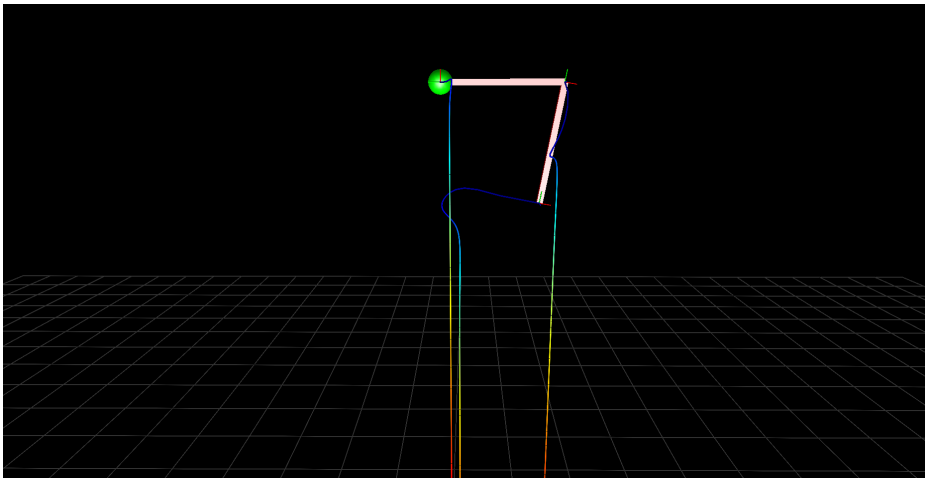

Figure 6: Posture control of our robot. The robot falls down from a initial position and changes the angle of attack to zero degrees and a spring length of 0.9 meters.

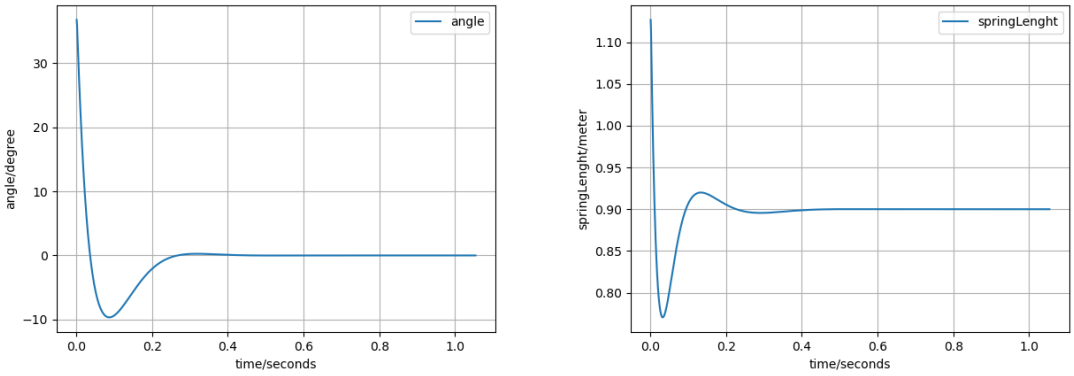

Figure 7: Left: Change of the angle of attack of the foot. With desired angle of attack of zero degrees. The system overshoots the zero position but returns to the stable end parameter. Right: Change of the spring length with a desired spring length of 0.9 meters. The system overshoots more then in the first image but also converges to the target parameter.

4.4 Velocity Controller

Since we can control the foot of our robot we next have to control the angle of attack depending on the actual velocity v_{cog}. For that we use an adapted PI controller.

$$\alpha = K_{c,v} v_{cog} + K_{p,v} e_v + K_{i,v} \sum e_v \tag{20}$$

Here $e_v = v_{set} - v_{cog}$ where $v_{cog} = \dot{\mathbf{x}}_{cog}[x]$ is the error in between the desired and the actual CoG velocity in x direction. To get to this controller we performed several tests trying to combine stable hopping with being able to move the overall model with a specific velocity and acceleration. We can only change our velocity with the angle of attack in the next impact, as this is the only point where we can apply a force. The velocity in flight phase is constant. Depending on the actual v_{cog} direction velocity in x direction of the center of gravity $\dot{x}_{cog}$ we tested different angles of attack and figured out a good constant connection $K_{c,v}$. To change or correct our actual velocity we are using a proportional controller $K_{p,v}e$ where $K_{p,v}$ is our constant gain. Since a single proportional controller can not compensate the error we added an integral part $K_{i,v} \sum e_v$ where $K_{i,v}$ is our integral gain. Figure 8 shows that the robot is hopping stably with $1.5\frac{m}{s}$ from the left to the right side. Over

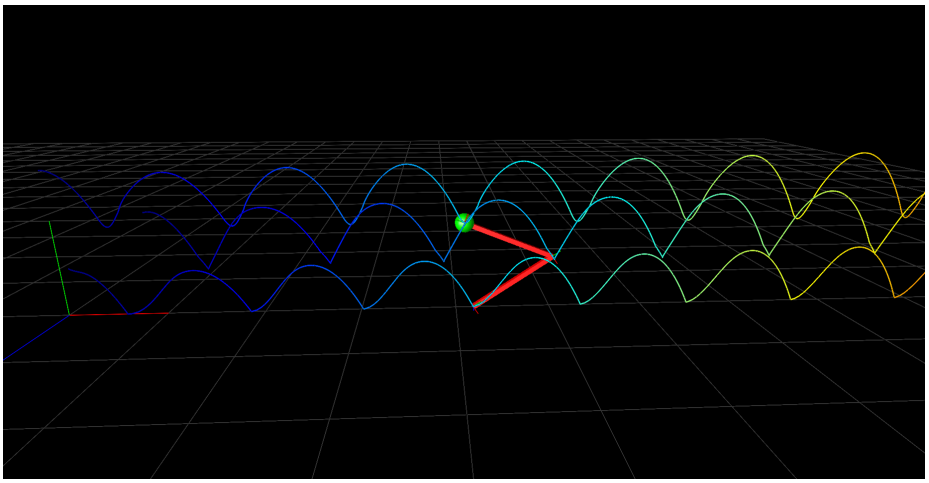

Figure 8: Robot trajectory: initial velocity of center of gravity TODO(v is not the name for the velocity of cog DONE) $\dot{\mathbf{r}}_{cog} = \begin{pmatrix} 1.5 & 0 \end{pmatrix}^T$ m/s, starting position of center of gravity $\mathbf{r}_{cog,\,start} = \begin{pmatrix} 0 & 1.5 \end{pmatrix}^T$ m, desired velocity of center of gravity in x-direction $\dot{x}_{cog,\,des} = 1.5\,\mathrm{m/s}$.

time the controller optimizes the angle of attack α to achieve stable hopping with $1.5\frac{m}{s}$ in the flight phase, see figure 9.

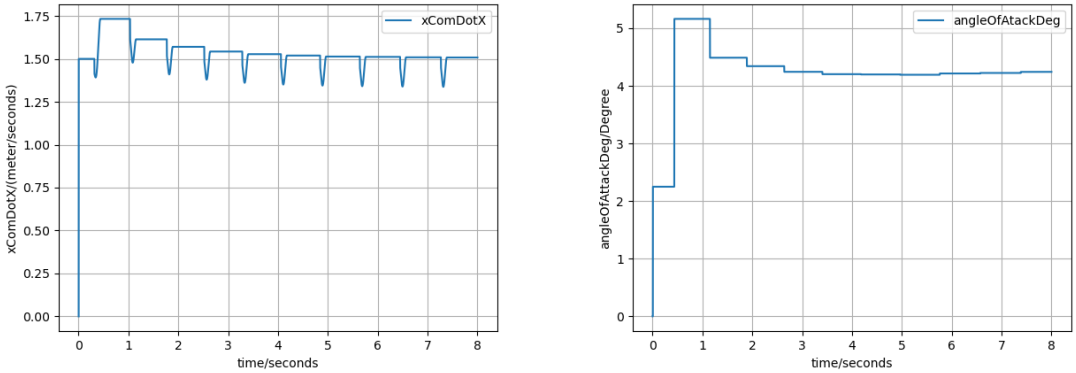

Figure 9: Left: Change of velocity in x-direction of the CoG $\dot{x}_{cog}$ over time t. Right: Change of the angle of attack α over time t

4.5 Position Controller

After successfully controlling the end-effector and the center of gravity to move with a desired velocity, an additional controller is required in order to drive the robot to a target position. For this purpose, a PI controller based on the following equation is introduced:

$$\hat{v}_{set} = K_{p,vel}\, e_{pos} + K_{i,vel} \sum e_{pos} \tag{21}$$

The controller will adjust the velocity of the center of mass in x-direction, depending on the temporal horizontal distance between current position and the destination.

$$e_{pos} = x_{cog,\,des} - x_{cog} \tag{22}$$

The robot should hops continuously, therefore it is sufficient to only control the velocity in the x-direction. The desired position we discussed in the following refers also only the x-coordinate of the robot's center of mass. The result $\hat{v}_{set}$ of the equation (22) is then used as the set value for the velocity controller introduced in the last section (section 4.4), if its absolute value does not extend the threshold $v_{threshold}$, otherwise we use $v_{threshold}$ as the set value.

$$v_{set} = \begin{cases} \hat{v}_{set} & if \quad |\hat{v}_{set}| \leq v_{threshold} \\ v_{threshold}\, sign(\hat{v}_{set}) & otherwise \end{cases} \tag{23}$$

Based on this value v_{set}, the velocity controller sets a suitable angle of attack as well as a feasible leg length for the leg posture controller.

Figure 10 shows the trajectory of the robot under the effect of the controllers. Here the destination is set to 4 meters away from the starting position. At the beginning, due to the large difference between the desired position and the original, the robot hops with a high horizontal velocity. When it gets closer to the goal, it takes smaller steps and also lower velocities. This is presented by the closer landing points and the stacked curve at the end of the trajectory. Figure 11 shows that the robot reaches the target $y = 4\,\mathrm{m}$ at about $t = 5\,\mathrm{s}$. The small oscillation at the end of curve around $x = 4\,\mathrm{m}$ indicates that the robot is adjusting its position carefully jumping backwards and forwards with tiny step size around the target to minimize the positional error e_{pos}.

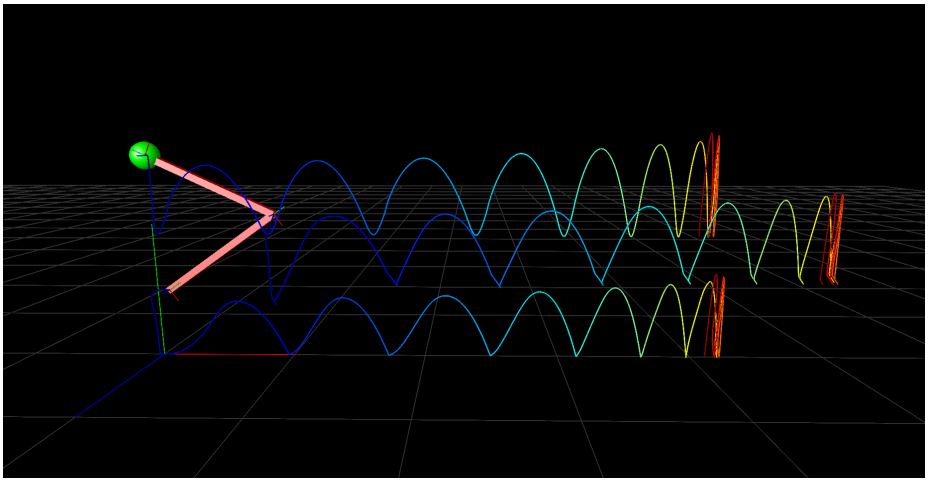

Figure 10: Robot trajectory: TODO(v is again not the velocitiy of the cog DONE) initial velocity of center of gravity $\dot{\mathbf{r}}_{cog} = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$ m/s, starting position of center of gravity $\mathbf{r}_{cog,\,start} = \begin{pmatrix} 0 & 1.5 \end{pmatrix}^T$ m, desired x-coordinate of center of gravity $x_{cog,\,des} = 4\,\text{m}$.

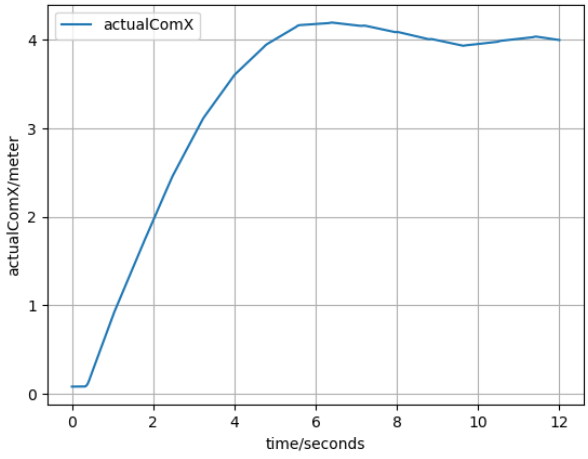

Figure 11: x-coordinate of center of gravity x_{cog} over time. With desired position of $x_{cog,\,des} = 4\,\text{m}$.

4.6 Stability to Disturbance

As the final step of our project, we create a simulation to validate, if the robot controlled with the introduced control system behaves stably after being disturbed.

In the experiment, the robot is disturbed by a constant impulse exerting on the center of mass in the positive x-direction at the starting position. The target horizontal position is set identical to the starting position for a clear illustration of the robot trajectory under influence of the impulse and the effect of the control system. Due to the disturbance, the robot hops forwards trying to damp the influence of the disturbance. After that, it jumps backwards towards the desired position, $x_{cog,\,des} = 0\,\mathrm{m}$. The change of the position of the CoG in x-direction is plotted in Figure 13. The trajectory in the Figure 12 illustrates that the introduced control system is capable of stabilising the robot from disturbance in flight phase. It also shows that the robot model has the ability to compensate the loss of kinetic energy at the impact at landing, as the trajectory apexes around the desired position have nearly the same height (apart from minimal numerical inconsistencies).

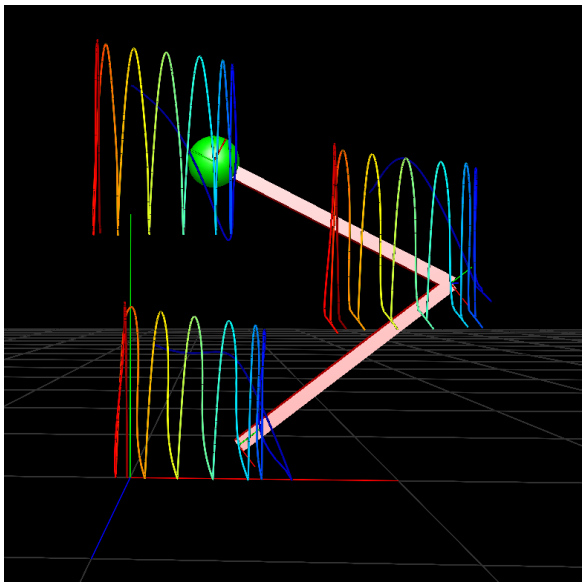

Figure 12: Robot trajectory: initial velocity of center of gravity $\dot{\mathbf{r}}_{cog} = \begin{pmatrix} 1.5 & 0 \end{pmatrix}^T\,\mathrm{m/s}$, starting position of center of gravity $\mathbf{r}_{cog,\,start} = \begin{pmatrix} 0 & 0 \end{pmatrix}^T\,\mathrm{m}$, desired x-coordinate of center of gravity $x_{cog,\,des} = 0\,\mathrm{m}$.

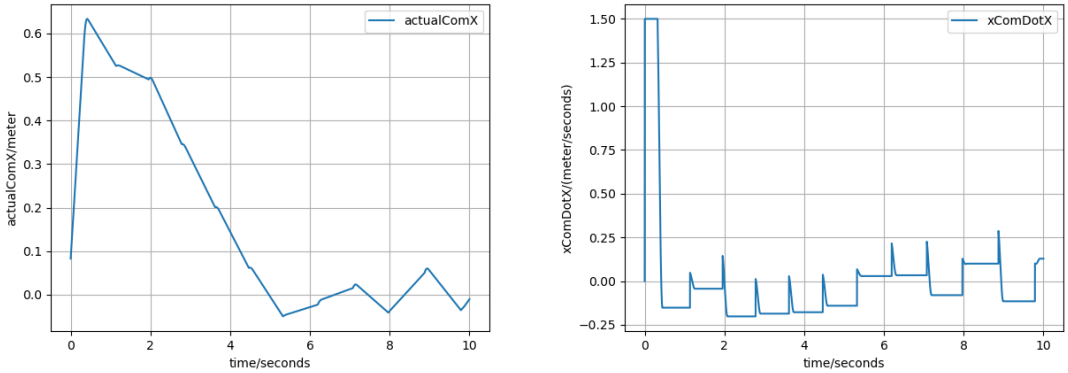

Figure 13: Left: Change of position of the CoG in x-direction x_{cog} over time t. The robot moves around 0.65 meters away from the original position due to the disturbance at the beginning. Under effect of the controller system, it hops backwards to approach the goal. Right: Change of the velocity in x-direction of the CoG $\dot{x}_{cog}$ over time t. The velocity at the beginning is 1.5 m/s due to the disturbance. It becomes lower than 0m/s under the adjustment made by the control system. At the end it oscillates around 0m/s, which indicates that the robot is adjusting its position in order to minimize its distance to the target.

5 How to Run

5.1 RBDL Simulation

There are several ways to run our simulation. For a start we want to show how to run a predefined scenario with the "articulatedLeg.lua" model which falls down from 1.5 meter above the ground and hops to its aim 4 meter in x direction.

1. To run our code, first clone our git repository from the LRZ-Gitlab-Instance. This can be done by running "git clone https://gitlab.lrz.de/00000000014A5DC6/dodo-alive" in your terminal emulation of choice.

2. Once the cloning has finished, change to the folder dodo-alive/examples/Slip/ by running "cd dodo-alive/examples/Slip/".

3. Execute "./start.sh 8000 4" in a terminal in the Slip folder to launch the simulation

The last command will create a new build folder and change the current directory to this folder. Then it will build the project. Start the program with two parameters. The first one determines the number of iterations executed for the simulation, in this case, 8000. The second one is the desired x-coordinate of the CoG's position in meters. Once the simulation has been calculated, the program "MeshUp" will open a new window showing the visualization of the simulation results we just calculated. If you change some parts in the code you can run "./start.sh" to rebuild the project.

Furthermore, the user can also change some quantities in the program to obtain a desired configuration for the simulation. If you want to change the initial position and the initial velocity of the robot you have to open `src/dynamics.cc` and change the parameter "q" for the model's initial position and "qd" for the model's initial velocity to your requirements as it can be seen in figure 14.

```
64      /*
65       * q : The model's initial position (x_cog, y_cog) and angles between links (J1, J2)
66       * qd: The model's initial velocity.
67       * These two parameters should be changed according to the desired initial configuration.
68       */
69      q << 0,1.5,j1,j2;//1.5
70      qd << 0.0,0,0,0;
```

Figure 14: Changeable parameters in `src/dynamics.cc`. “q” and “qd” encode the robot's initial position and initial velocity respectively. To simulate certain initial configuration, these two parameters should be adjusted accordingly.

5.2 Visualization of Output Data

To get an easy access to the simulation data, we created a python interface which enables the convenient generation of plots. For that we used the Python library 'matplotlib' for the visualization as well as'pandas' for processing csv data. To start the visualization change to the folder `/dodo-alive/examples/Slip/` and execute `python3 viso.py` . You will be prompted with the console interface (Figure 15) which shows the variables you can select for the plot. Next it will ask

```
Dodo Alive
Easy Viso Interface
-----------------------------------------------------------
Available variables:
 ['time', 'actualComX', 'actualComY', 'xComDotX', 'xComDotY', 'angleOfAttackDeg', 'xComDifPha
se', 'springLegDelta', 'Phase', 'footInBaseX', 'footInBaseY', 'deltaLegLength']
Please select your x value: time
you selected for x: time
Please enter the axis label: time/seconds
Please select your y value: actualComX
you selected for y: actualComX
Please enter the axis label: actualComX/meter
-----------------------------------------------------------
Generated plot: time/seconds ( time ) over actualComX/meter ( actualComX ).
```

Figure 15: Console interface of `viso.py`. Available parameters for visualisation are: time of the simulation (”time”), x- and y-coordinate of the actual position of cog in world coordinate system (“actualComX”, “actualComY”), x- and y-coordinate of the velocity of end-effector(“xComDotX”, “xComDotY”) in foot fixed coordinate system, angle of attack (“angleOfAttack”), etc.

you to select a value for the x-axis, here we use ”time”. You also need to select your y-axis value, like ”actualComX”. Then it opens the generated plot, see Figure 16. If you want to save the figure you have can saved it to your hard drive from the matplotlib interface.

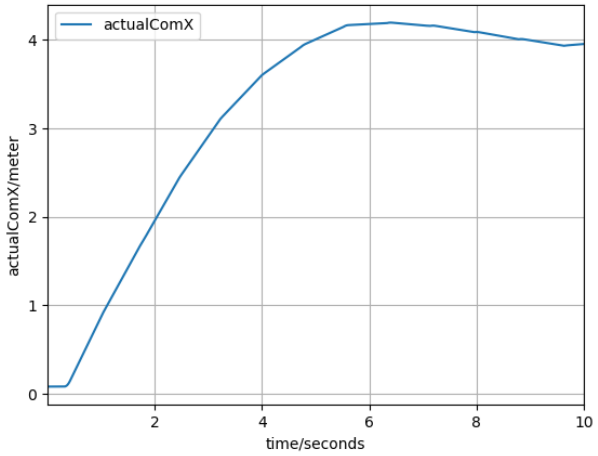

Figure 16: Generated plot of the actual x-coordinate of the CoG position for a simulation run. Labels of x- and y-axis are user-defined values “time/seconds” and “actualComX/meter” respectively.

6 Conclusion

6.1 Result

In our project, we simulated an articulated leg model based on the theoretical foundation from [2] using the RBDL library. The model behaves similar to a spring-mass model. However, in contrast to an ideal SLIP model with a mass-less prismatic spring, a real robotic leg suffers from energetic losses on ground impact. With the methods proposed in [2], we pre-compress the virtual spring of our model before establishing ground contact. The energy stored in the pre-compressed virtual spring compensates the loss of kinetic energy of the CoG in the collision. Our model is therefore energetically conservative even in the presence of impact losses.

Additionally, we control the behavior of the robot with a three level cascade control system consisting of a position controller (section 4.5), a velocity controller (section 4.4) as well as a task space posture controller (section 4.3). With this control system, our model is capable of stabilizing itself from disturbance in the flight phase, hop to the target position and hop continuously maintaining the apex height (apart from minimal numerical inconsistencies)

Furthermore, for our project, we implemented a simulation program in C++ using the RBDL library as well as a convenient visualisation tool in python, which is introduced in section 5. To obtain an intuitive representation of the result, the user can not only generate an animation of the simulated movement of the robot with Meshup, but also visualise the change of selected parameters over time using the visualisation tool. These tools can reduce the time consumed in modifying the simulation and in evaluating the simulation results. Changing a few parameters before running the program, simulations in different configurations can be generated easily.

6.2 Future Work

So far, we only simulated the behavior of the model in a 2D space. For future extension of the model, one could evaluate behavior of the model in 3D space and optimize the corresponding control mechanisms accordingly. Additionally, we only validated the stability of our model to disturbance exerted during the flight phase and did not examine whether our model still behaves stably when it is disturbed during the stance phase. In reality, the robot should be able to hop on an uneven ground. Hence, the remaining problem, how to damp the disturbance in stance phase, plays an important role when we want to ensure a stable behavior of the model in real life.

A Appendix

A.1 Note to chapter 4.2 equation:5

In the paper 'Slip running with an articulated robotic leg' by Marco Hutter et al. [2] in equation:[9] for the task inertia, we found an inconsistency. In the paper, the following equation is given:

$$\mathbf{\Lambda}^* = (\mathbf{J}\mathbf{M}^{-1}\mathbf{S}\mathbf{N}_s\mathbf{J}^*)^{-1} \tag{24}$$

The dimensions of the matrices multiplied here do not match, making the evaluation of the equation in the form impossible. After tracking the calculations of each element of the equation, we derived the correct formulation of the inertia task $\mathbf{\Lambda}^*$ as follows:

With Eq. (2.80) in [5]

$$\mathbf{\Lambda}^* = (\mathbf{J}^*\mathbf{\Phi}^*\mathbf{J}^{*-1})^{-1} \tag{25}$$

and Eq. (2.50) in [5]

$$\mathbf{J}^* = \mathbf{J}_{t|s}(\mathbf{S}\mathbf{N}_s)^{\#} \tag{26}$$

as well as Eq. (2.42) in [5]

$$\mathbf{\Phi}^* = \mathbf{S}\mathbf{N}_s\mathbf{M}^{-1}(\mathbf{S}\mathbf{N}_s)^T \tag{27}$$

we obtain the formulation of $\mathbf{\Lambda}^*$ as

$$\mathbf{\Lambda}^* = (\mathbf{J}_s(\mathbf{S}\mathbf{N}_s)^{\#}\mathbf{S}\mathbf{N}_s\mathbf{M}^{-1}(\mathbf{S}\mathbf{N}_s)^T\mathbf{J}^{*T})^{-1} \tag{28}$$

which we can combine to the following final equation:

$$\mathbf{\Lambda}^* = (\mathbf{J}\mathbf{M}^{-1}(\mathbf{S}\mathbf{N}_s)^T\mathbf{J}^{*-1})^{-1} \tag{29}$$

References

[1] Alexander. "Three Uses for Springs in Legged Locomotion". In: *The International Journal of Robotics Research* 9 (1990), pp. 54–61.

[2] Marco Hutter et al. "Slip running with an articulated robotic leg". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 4934–4939.

[3] *MeshUp*. `https://github.com/ORB-HD/MeshUp`. Accessed: 2020-18-05.

[4] *RBDL - Rigid Body Dynamics Library*. `https://github.com/ORB-HD/rbdl-orb`. Accessed: 2020-18-05.

[5] Luis Sentis. *Synthesis and control of whole-body behaviors in humanoid systems*. Citeseer, 2007.