Exploration of control strategies for a three-link 2D biped

Teguh Santoso Lembono

December 13, 2020

1 Introduction

The purpose of this report is to present the modelling, simulation, and control of a simplified humanoid robot walking model. The report is divided into two parts. The first presents the kinematics and dynamics analysis of the walking model, the impact map, the numerical integration for simulation, and the standard controller. The second part then presents the proposed controller using an optimal control approach. The first part uses matlab, while python + CasADi are used for the second part.

2 Theory

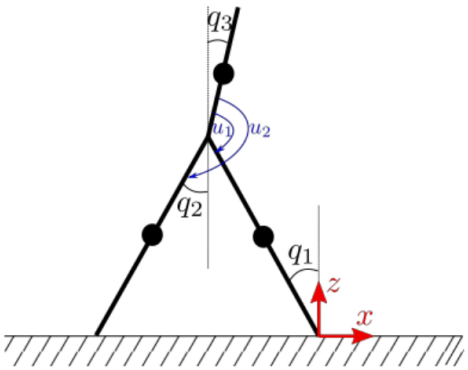


Figure 1: The simplified humanoid model used in this report.

The simplified model of the humanoid is shown in Fig. **??**. It is represented by three-links model, two refer to the legs and one for the torso. Each link has a point mass at its center. At each instant, one leg acts as the stance leg that stays stationary, and the other as the swing leg.

2.1 Kinematics

The kinematics can be easily calculated by considering the stance foot location as the origin of the coordinate frame, as shown in Fig. **??**. The points of interests are then the location of each mass $(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3)$, the torso $(\boldsymbol{r}_t)$, the hip $(\boldsymbol{r}_h)$, and the swing foot $(\boldsymbol{r}_{swf})$. Each point $\boldsymbol{r}$ consists of the x and

z coordinates. By simple geometry, we obtain

$$\begin{aligned}
x_1 =& 0.5 * l_1 * sin(q_1) && (1)\\
z_1 =& 0.5 * l_1 * cos(q_1) && (2)\\
x_2 =& l_1 * sin(q_1) - 0.5 * l_2 * sin(q_2) && (3)\\
z_2 =& l_1 * cos(q_1) - 0.5 * l_2 * cos(q_2) && (4)\\
x_3 =& l_1 * sin(q_1) + 0.5 * l_3 * sin(q_3) && (5)\\
z_3 =& l_1 * cos(q_1) + 0.5 * l_3 * cos(q_3) && (6)\\
x_h =& l_1 * sin(q_1) && (7)\\
z_h =& l_1 * cos(q_1) && (8)\\
x_{swf} =& l_1 * sin(q_1) - l_2 * sin(q_2) && (9)\\
z_{swf} =& l_1 * cos(q_1) - l_2 * cos(q_2) && (10)\\
x_t =& l_1 * sin(q_1) + l_3 * sin(q_3) && (11)\\
z_t =& l_1 * cos(q_1) + l_3 * cos(q_3). && (12)
\end{aligned}$$

The corresponding velocities can then be computed using chain rule, i.e.

$$\frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t} = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{q}} \frac{\partial \boldsymbol{q}}{\partial t}. \tag{13}$$

Using the symbolic programming in matlab or python, the velocity calculation can be done automatically.

2.2 Dynamics

To obtain the dynamic equation of the locomotion model, the dynamics is divided into two phases: the swing phase and the impact phase. In the swing phase, we assume that the stance foot is not moving (there is sufficient contact force to keep it stationary), while the swing foot is moving to the next contact location. At the point of impact, we assume an instantaneous impact that preserves the momentum, as described in the next section.

To describe the dynamics of the swing phase, we can rely on Lagrangian dynamics. First, we found the expression of the total kinetic and potential energy (T and V),

$$T = \sum_{i=1}^{3} \frac{1}{2} m_i \frac{\mathrm{d}\boldsymbol{r}_i}{\mathrm{d}t}^2 \tag{14}$$

$$V = \sum_{i=1}^{3} m_i g z_i. \tag{15}$$

The Lagrangian is then calculated as

$$L = T - V. \tag{16}$$

With this, the Lagrangian equation of motion can be calculated as

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \tau_i, \tag{17}$$

where τ_i is the generalized torque. The equations can be computed using the symbolic programming.

After obtaining the equations for $i = 1, 2, 3$, we can rearrange them in the following form,

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau}. \tag{18}$$

The terms $\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, and $\boldsymbol{G}(\boldsymbol{q})$ can be computed from (**??**) as follows. $\boldsymbol{G}(\boldsymbol{q})$ is computed by setting $(\dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{\tau})$ in (**??**) to zero. $\boldsymbol{M}(\boldsymbol{q})$ is computed by setting $(\dot{\boldsymbol{q}}, \boldsymbol{\tau})$ in (**??**) to zero, substracting the term $\boldsymbol{G}(\boldsymbol{q})$ from the equation. Finally, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is computed by setting $(\ddot{\boldsymbol{q}}, \boldsymbol{\tau})$ to zero, substracting the term $\boldsymbol{G}(\boldsymbol{q})$ from the equation, and dividing by the corresponding $\dot{\boldsymbol{q}}$.

2.3 Impact Map

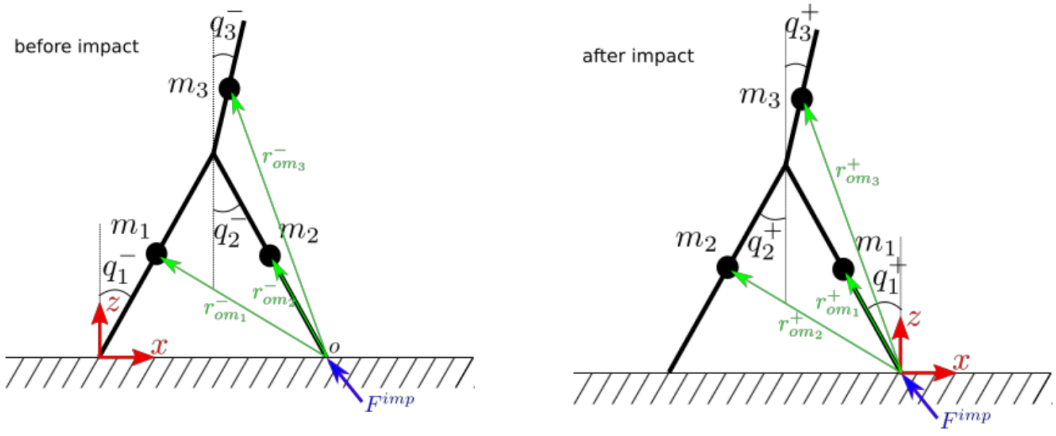


Figure 2: Diagram for angular momentum conservation at the impact point. *Left*: before impact, *right*: after impact.

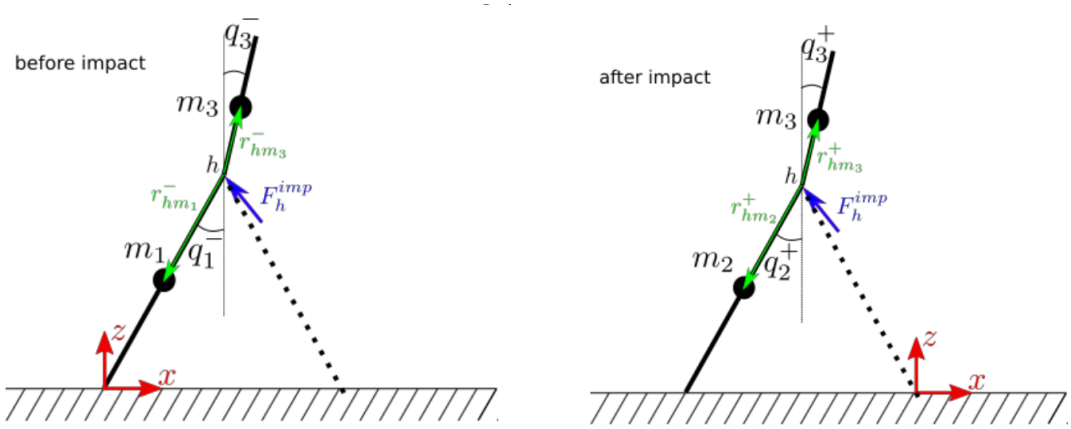


Figure 3: Diagram for angular momentum conservation at the hip. *Left*: before impact, *right*: after impact.

In our model, after the impact we switch the swing foot to become the stance foot, as shown in Fig. **??** (left: before impact, right: after impact). The angles are therefore changed as follows,

$$q_1^+ = q_2^- \tag{19}$$
$$q_2^+ = q_1^- \tag{20}$$
$$q_3^+ = q_3^-, \tag{21}$$

the superscripts $(^-,^+)$ refer to the variables before and after impact, respectively.

The impact map for the angular velocity can be computed by considering the conservation of angular momentum at the impact at two locations: the impact point (the location of the swing foot at the impact), and at the hip.

Looking at Fig. **??** and **??**, the momentum before the impact can be written as:

$$H_a^- = \sum_{i=1}^{3} m_i \boldsymbol{r}_{om_i}^- \times \dot{\boldsymbol{r}}_i{}^- = \sum_{i=1}^{3} m_i (\boldsymbol{r}_i^- - \boldsymbol{r}_{swf}^-) \times \dot{\boldsymbol{r}}_i{}^- \tag{22}$$

$$H_b^- = m_1 \boldsymbol{r}_{hm_1}^- \times \dot{\boldsymbol{r}}_1^- = m_1 (\boldsymbol{r}_1^- - \boldsymbol{r}_h^-) \times \dot{\boldsymbol{r}}_1^- \tag{23}$$

$$H_c^- = m_3 \boldsymbol{r}_{hm_3}^- \times \dot{\boldsymbol{r}}_3^- = m_3 (\boldsymbol{r}_3^- - \boldsymbol{r}_h^-) \times \dot{\boldsymbol{r}}_3^-, \tag{24}$$

and after impact as

$$H_a^+ = \sum_{i=1}^{3} m_i \boldsymbol{r}_{om_i}^+ \times \dot{\boldsymbol{r}}_i{}^+ = \sum_{i=1}^{3} m_i \boldsymbol{r}_i^+ \times \dot{\boldsymbol{r}}_i{}^+ \tag{25}$$

$$H_b^+ = m_1 \boldsymbol{r}_{hm_2}^+ \times \dot{\boldsymbol{r}}_2^+ = m_1 (\boldsymbol{r}_2^+ - \boldsymbol{r}_h^+) \times \dot{\boldsymbol{r}}_2^+ \tag{26}$$

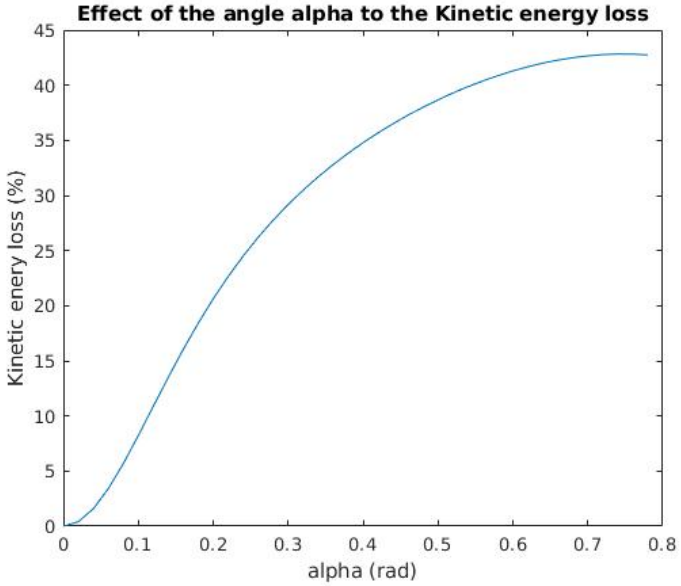
$$H_c^+ = m_3 \boldsymbol{r}_{hm_3}^+ \times \dot{\boldsymbol{r}}_3^+ = m_3 (\boldsymbol{r}_3^+ - \boldsymbol{r}_h^+) \times \dot{\boldsymbol{r}}_3^+. \tag{27}$$

We can rearrange the equations to obtain $\boldsymbol{H}^- = \boldsymbol{A}_- \dot{\boldsymbol{q}}^-$ and $\boldsymbol{H}^+ = \boldsymbol{A}_+ \dot{\boldsymbol{q}}^+$. Then we can obtain the velocities after the impact by solving the linear equation $\boldsymbol{H}^- = \boldsymbol{H}^+$,

$$\dot{\boldsymbol{q}}^+ = \boldsymbol{A}_+^{-1} \boldsymbol{A}_- \dot{\boldsymbol{q}}^-. \tag{28}$$

2.3.1 Exercise Answers

- Q: *What can you say about the potential energy before and after impact?* The potential energy before and after impact do not differ, because the position of each mass remains the same. What changes after the impact is only the kinetic energy.
- Q: *Try* $q_m = [\pi/6, -\pi/6, \pi/10], dq_m = a[1, 0.2, 0]$. *What percentage of the kinetic energy of the biped is lost due to the impact?* There is 27.92% of the kinetic energy lost due to the impact.
- Q: *Plot the percentage of the kinetic energy loss due to impact as a function of angle* α *where* $q^- = [\alpha, -\alpha, 0]$ *and* α *varies from 0 to* $\pi/4$. *Assume that* $\dot{q}^- = [1, 0.2, 0]$. *Include your plot.*



- Q: *The bigger* α *is the bigger is the step length. Based on your answer to question 3, what is the relation between step length and energy loss at impact given a fixed* $\dot{q}^-$*?* As the step length grows bigger, the energy loss increases.

2.4 Model Validation and Animation

Using the kinematics, dynamics, and impact map functions, we can run a simulation of the locomotion model, given a particular controller. The simulation is run using numerical integration, which is described in the next section.

2.4.1 Exercise Answers

- *In the animations, what does a real-time factor of 1 mean? How about a real-time factor less than 1?* The real time factor describes the ratio between the actual time taken by the motion and the animation time. Real time factor of 1 means that the duration of the animation that appears to us is exactly the same as the duration of the actual step according to the model. Real time factor less than 1 means that the animation appears slower than the actual motion.
- *How does "skip" in animate.m effect the real-time factor and the speed of the animation?* The variable "skip" determines how many data points that we do not show in the animation. The larger this value is, the sparser the animation is, and the faster it is. Hence, as "skip" increases, the real time factor will increase and the animation speed becomes faster.
- *What is the role of r0 in animate.m?* r0 is the location of the supporting foot. This is necessary because the coordinates of the other points are described with respect to this point. As the supporting foot changes at the impact, the value of r0 is updated correspondingly.

2.5 Numerical Integration

At each step, we run the numerical integration of the swing phase dynamics, while checking if the swing foot crosses the ground. This can be defined as an event function that is checked in the numerical integrator at each function evaluation. For the integrator, I am using *solve_ivp* from the scipy library in python.

2.5.1 Exercise Answer for Explicit Euler

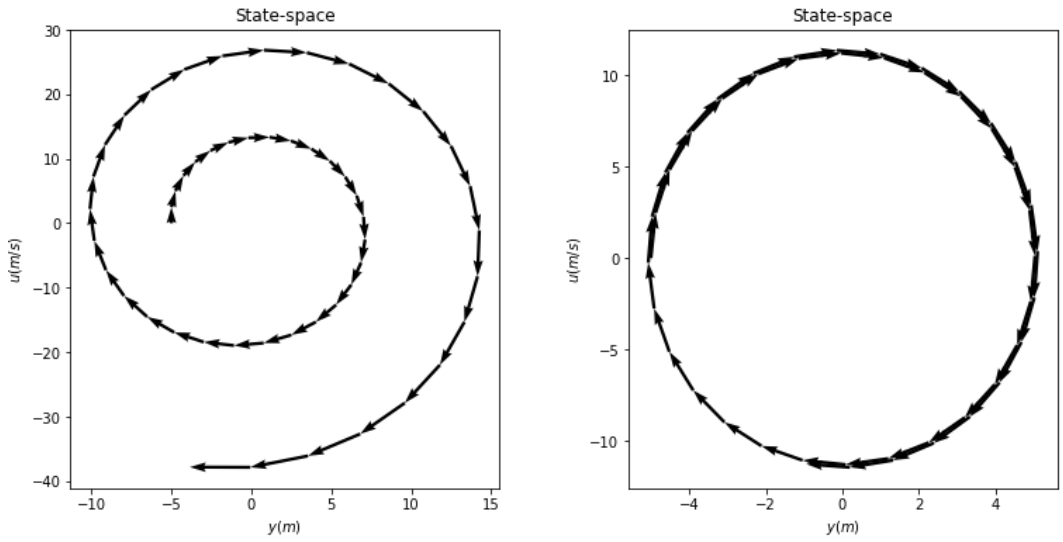


Figure 4: State space plot for explicit euler. *Left*: $dt = 0.1$, *right*: $dt = 0.002$.

- *Explain the meaning of the state-space plot (right subfigure). What should be the expected state-space plot (hint: change visualize function)?* The state space plot shows the evolution of the joint angle $\boldsymbol{q}$ and its velocity $\dot{\boldsymbol{q}}$ over time. Since the damping coefficient here is zero, the system should have a stable oscillation, i.e. the movement should be periodical and after one period we should go back to the same point on the phase plot. What is shown here, though, is a spiralling movement whose radius is increasing, showing that the energy of the system increases as time goes on, due to the inaccuracy of Explicit Euler method.

- *Why does the numerical solution diverges from the true solution (hint: Taylor expansion)?* Explicit Euler uses a first order Taylor expansion to approximate the integration, this results in a quite significant error, especially when the time interval is large (0.1 in this case). This results in energy increase as the integration is performed, resulting in the divergence.

- *What can you do to better approximate the actual solution with this method?* We can decrease the time interval to reduce the error. Fig. **??** (right) shows the phase plot for dt = 0.002, which has an almost circular shape (stable limit cycle).

2.5.2 Exercise Answer for Semi-implicit Euler

- *Is this method stable and when?* By looking at the phase plot, the method seems to be quite stable, although the shape does not fit a perfect circle (meaning a non-negligible error, although the error is not accumulating).

- *What about the accuracy of the obtained solution?* The accuracy can be computed as the mean squared error $e = \frac{1}{N}\sum_{n=1} N(\boldsymbol{y} - \boldsymbol{y}_{\text{true}})^2$. For $dt = 0.1$, the error is $e = 0.202$.

2.5.3 Exercise Answer for Runge-Kutta

- *What is the error of this method?* The mean squared error is $e = 2.58e - 7$ for $dt = 0.1$, which is much lower than Semi-implicit euler method.

- *Do you expect the solution to diverge?* Looking at the phase plot, it does not seem to diverge. The error is also very small.

2.5.4 Exercise Answer for Adaptive Time Stepping

- *Change the density of the time points in np.linspace. What do you observe?* With different density, the solution seems to remain quite stable. Even with very low density, the solution is still stable.

- *Do you expect the solution to diverge?* Looking at the phase plot, it seems to make a stable limit cycle and does not seem to diverge.

- *Which algorithms are supported by odeint?* Odeint supports 'LSODA' from FORTRAN library. The newer algorithm for numerical integration seems to be *solve_ivp*, which supports 'RK45', 'RK23', 'DOP853', 'Radau', 'BDF', and 'LSODA'.

2.6 Control and Optimization

The system during the swing phase is underactuated, i.e. there are three degrees of freedom to control but only two control inputs, as shown in Fig. **??**. In the standard controller, we first choose the first control input u_1 to control the torso angle to be at a desired value α, whereas the second control input

u_2 is used to control the swing foot angle q_2. Expressing these as virtual constraints in terms of new variables y_1 and y_2, we set

$$y_1 = \quad q_3 - \alpha \tag{29}$$
$$y_2 = \quad -q_2 - q_1. \tag{30}$$

Differentiating this constraint, we obtain

$$\dot{y}_1 = \quad \dot{q}_3 \tag{31}$$
$$\dot{y}_2 = \quad -\dot{q}_2 - \dot{q}_1. \tag{32}$$

We then set the controller as PD control to drive y_1, y_2 to zeros,

$$u_1 = \quad k_1^P y_1 + k_1^D \dot{y}_1 \tag{33}$$
$$u_2 = \quad k_2^P y_2 + k_2^D \dot{y}_2. \tag{34}$$

Now what is left is to choose the parameters ($k_1^P, k_1^D, k_2^P, k_2^D$, and α), as well as the initial state ($\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0$). I use the function *fminsearch* in Matlab to optimize these parameters, given an objective function.

2.6.1 Objective function for optimizing the standard controller

For this part, I use the standard objective function given in the exercise,

$$F(\boldsymbol{\Phi}) = w_1 * |\epsilon - \epsilon_d| + w_2 * CoT, \tag{35}$$

where w_1 and w_2 are the weights and CoT is the cost of transport (as defined in the exercise). ϵ refers to the criteria of interest (e.g. velocity, step length, frequency), and ϵ_d is the desired value. Since CoT typically has high value, w_2 is set to be quite small, typically 3 order lower than w_1 (e.g. $w_1 = 3, w_2 = 0.001$). The CoT term is useful to help the motion to be stable, even when we do not want to minimize CoT. Finally, to avoid bad cases such as moving backwards or falling down, such event is detected and a large objective function value is assigned (= 1000).

2.6.2 Result

Table **??** shows the parameters obtained for various criteria. The controller, although simple, is able to reach quite a large range of criterias.

Table 1: Parameters for standard controller

Criteria	**Value**	$\boldsymbol{q}_0$	$\dot{\boldsymbol{q}}_0$	$(k_1^P, k_2^P, k_1^D, k_2^D, \alpha)$
Velocity	0.2	(-0.284, 0.345, 0.005)	(0.759, -0.323, 2.11)	(476, 160, 72, 7.45, 0.0016)
Velocity	0.9	(-0.371, 0.382, 0.129)	(0.607, -1.162, 3.857)	(352, 179, 80, 4.65, 0.194)
Frequency	0.7	(-0.284, 0.345, 0.005)	(0.759, -0.323, 2.11)	(476, 160, 72, 7.45, 0.0016)
Frequency	3.7	(-0.168, 0.260, -0.014)	(1.85, 1.356, 0.034)	(304, 189, 134, -0.05, -0.183)
Step Length	0.21	(-0.168, 0.260, -0.014)	(1.85, 1.356, 0.034)	(304, 189, 134, -0.05, -0.183)
Step Length	0.4	(-0.037, -0.610, 0.00)	(0.00, 0.156, 5.79)	(244, 269, 53.4, 6.51, 0.156)

3 Control

3.1 Overview of Control Strategy

In this work, I rely on *optimal control* formulation to design an efficient locomotion controller. The resulting motion is optimal w.r.t. the desired criteria, which can be expressed as an objective function and constraint(s). A high level specification (such as minimizing energy, tracking a desired foot trajectory, tracking a desired torso angle) can be translated quite easily to the objective function.

However, there are several challenges in implementing the optimal controller to this problem. Firstly, most optimal control strategy in robotics is time-based. That is, the time horizon has to be specified (e.g. 1 second), and solving the optimal control problem will give us a specific control and state trajectory for this time horizon that minimizes the objective function while satisfying the constraints. Some variants are time independent, e.g. when the time horizon is infinite, but it is not suitable for this locomotion problem (where the desired behavior highly depends on time and event, and not stationary). Since the impact timing is difficult to predict precisely (especially given some disturbance), a purely time-based controller is difficult to be used here. Secondly, the dynamics model of our locomotion is discontinuous due to the impact map. This is a big problem for optimal control, as most methods would require the gradient of the dynamics function.

To overcome these issues, I propose to use the following approach. First, the optimal control is solved for only one step until impact, so that the discontinous event at the impact is not considered. Considering that the locomotion is a periodical motion, designing an optimal trajectory for one step should be sufficient, provided that the state after the impact returns to the initial state. Secondly, we need a controller that is not purely time based or feed forward only. One option is to use a feedback controller (e.g. PD) to track the desired optimal trajectory. However, there is a specific optimal control formulation that provides this automatically, i.e. Iterative Linear Quadratic Regulator (ILQR) [**Tassa2012**]. Solving ILQR gives us both the optimal trajectories and the optimal feedback controller (note that 'optimal' here refers to local optimum, not the global one). Finally, to choose the optimum parameters for the optimal control, we again rely on an optimization solver (fmin) to ensure that the controller is stable.

3.2 Iterative Linear Quadratic Regulator (ILQR)

A general discrete Optimal Control Problem consists of a cost function

$$C(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + c_T(\boldsymbol{x}_T, \boldsymbol{u}_T), \tag{36}$$

subject to the dynamics

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t). \tag{37}$$

$(\boldsymbol{x}, \boldsymbol{u})$ are the state and control trajectories for the time horizon T. $\boldsymbol{x}_t$, the state at time t, consists of the joint angles and velocities, i.e. $\boldsymbol{x}_t = [\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t]$. ILQR solves this optimal control problem by iteratively formulating a sub problem as a Linear Quadratic Problem (LQR) problem and solve it to obtain a better solution at each iteration.

Let us consider the current guess $(\boldsymbol{x}^k, \boldsymbol{u}^k)$, where k is the iteration index. Given the general cost function in (**??**), we approximate it as a quadratic function around $(\boldsymbol{x}^k, \boldsymbol{u}^k)$,

$$c_t(\delta\boldsymbol{x}_t, \delta\boldsymbol{u}_t) = \frac{1}{2}\begin{bmatrix}\delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t\end{bmatrix}^\top \begin{bmatrix}\boldsymbol{c}_{\boldsymbol{xx},t} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{c}_{\boldsymbol{uu},t}\end{bmatrix}\begin{bmatrix}\delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t\end{bmatrix} + \begin{bmatrix}\boldsymbol{c}_{\boldsymbol{x},t} & \boldsymbol{c}_{\boldsymbol{u},t}\end{bmatrix}\begin{bmatrix}\delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t\end{bmatrix}, \tag{38}$$

where $\boldsymbol{c}_{\boldsymbol{x},t}, \boldsymbol{c}_{\boldsymbol{u},t}, \boldsymbol{c}_{\boldsymbol{xx},t}$, and $\boldsymbol{c}_{\boldsymbol{uu},t}$ are the cost function's first and second order derivatives with respect to $\boldsymbol{x}$ and $\boldsymbol{u}$. Similarly, we can approximate the dynamics in (**??**) using the linear approximation

$$\delta\boldsymbol{x}_t = \boldsymbol{A}_t\delta\boldsymbol{x}_t + \boldsymbol{B}_t\delta\boldsymbol{u}_t, \tag{39}$$

where $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ are the derivatives of the dynamics $\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ with respect to $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$, respectively. The derivatives are evaluated at the current guess $(\boldsymbol{x}^k, \boldsymbol{u}^k)$. With this, we have quadratic costs and linear dynamics as functions of $\delta\boldsymbol{x}$ and $\delta\boldsymbol{u}$. This is therefore a time-varying LQR problem, of which the variables of interest are $\delta\boldsymbol{x}$ and $\delta\boldsymbol{u}$, and the solution can be computed analytically.

By the end of ILQR iteration, we will have a locally optimal solution as state and control trajectories $(\boldsymbol{x}^*, \boldsymbol{u}^*)$, as well as the time-varying optimal feedback term $\boldsymbol{K}_t$. To use this feedback term, at online execution we compute the control command by

$$\boldsymbol{u}_t = \boldsymbol{u}_t^* + \boldsymbol{K}_t(\boldsymbol{x}_t - \boldsymbol{x}_t^*). \tag{40}$$

This optimal feedback term serves as a locally optimal PD controler to track the optimal trajectories $(\boldsymbol{x}^*, \boldsymbol{u}^*)$.

More details about ILQR can be found in [**Tassa2012**].

3.3 ILQR formulation for the locomotion

For the locomotion problem, we consider the initial state $\boldsymbol{x}_0$ to be the state right after an impact, and the end state $\boldsymbol{x}_T$ to be at the next predicted impact. It turns out that formulating the objective function for this problem is quite complicated, especially due to the fact that the impact is detected based on event, and cannot be predicted too precisely by the timing. I previously try using an objective function to only reach the desired end state $\boldsymbol{x}_T$, but this often results in failure because the swing foot touches the ground before reaching $\boldsymbol{x}_T$. In this event based detection, the swing foot trajectory is very important. Another variable that is also important is the torso angle ($\boldsymbol{q}_3$). This needs to be regulated carefully, otherwise the motion is often unstable.

After a lot of trials, the objective function that is found to be working is to track a desired foot swing trajectory, while keeping the torso angle to be around a desired angle. The running cost, i.e. the cost for $t = [0, T-1]$ is set to be

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = (\boldsymbol{r}_{swf,t} - \boldsymbol{r}^*_{swf,t})^\top \boldsymbol{W}(\boldsymbol{r}_{swf,t} - \boldsymbol{r}^*_{swf,t}) + \lambda(q_{3,t} - \alpha)^2 + \boldsymbol{u}_t^T \boldsymbol{R}\boldsymbol{u}_t, \tag{41}$$

while the terminal cost at $t = T$ is set to be

$$c(\boldsymbol{x}_T, \boldsymbol{u}_T) = (\boldsymbol{r}_{swf,T} - \boldsymbol{r}^*_{swf,T})^\top \boldsymbol{W}_T(\boldsymbol{r}_{swf,T} - \boldsymbol{r}^*_{swf,T}) + \lambda(q_{3,T} - \alpha)^2. \tag{42}$$

The constraint on the control input is a box constraint,

$$-30 \leq \boldsymbol{u}_i \leq 30 \quad \text{for} \quad i = 1, 2. \tag{43}$$

For time $t < T$, we only care about its vertical coordinate z_{swf}, to make sure that it does not touch the ground before time T. I use a simple linear interpolation between the initial height $z_{swf,0}$ and final height $z_{swf,T}$. At time $t = T$, we want the swing foot to reach a specific foot location, so both the horizontal and vertical coordinates are important. The horizontal coordinate corresponds to the desired step length, while the vertical coordinate corresponds to the ground (in this case, around -0.01). The weight parameters are

$$\boldsymbol{W}_t = \begin{bmatrix} 0 & 0 \\ 0 & 10^4 \end{bmatrix}, \boldsymbol{W}_T = \begin{bmatrix} 10^4 & 0 \\ 0 & 10^5 \end{bmatrix}, \boldsymbol{R} = \begin{bmatrix} 10^{-2} & 0 \\ 0 & 10^{-2} \end{bmatrix}, \lambda = 100. \tag{44}$$

To solve this constrained ILQR problem, I use the open source solver Crocoddyl [**Mastalli**]. It uses the box-FDDP algorithm [**mastalli2020direct**] that accepts infeasible initial guess $(\boldsymbol{x}^0, \boldsymbol{u}^0)$. *Infeasible* here means that $(\boldsymbol{x}^0, \boldsymbol{u}^0)$ may not be dynamically consistent. Allowing this infeasibility makes the initialisation easy.

3.4 Optimization Strategy

To use the optimal control formulation, we have to choose the following parameters $\mathbf{\Phi}$:

- The initial state $(\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0)$
- The desired torso angle α
- The desired goal position for the swing foot $x^*_{swf,T}$

In addition, we also have to specify the time horizon T. Although optimal control formulation is powerful, finding the optimal parameters for a desired objective of this particular locomotion problem turns out to be challenging. Instead of starting from random initial guess, I rely on the standard controller discussed in Section **??**. For a given desired criteria (e.g. a desired speed, step length etc.), first I optimize this standard controller to achieve the criteria, and observe the corresponding parameters $\mathbf{\Phi}$ as well as the time interval T. Using this with the ILQR often results in failure, i.e. unstable controller, as the ILQR is still time based and depend a lot on the impact event. So we need to further optimize $\mathbf{\Phi}$ to obtain a good controller.

To make the optimization tractable, I do it step by step. First, I optimize for only one foot step. Since a stable controller will have a stable limit cycle, the next state after the impact should be very close to the initial state. So I design the objective function for the optimization to be

$$f(\mathbf{\Phi}) = ||\boldsymbol{x}_0 - \boldsymbol{x}_T^+||_2, \tag{45}$$

where $\boldsymbol{x}_0$ is the initial state, and $\boldsymbol{x}_T^+$ is the final state after the impact (i.e. the initial state for the next foot step). By ensuring that $\boldsymbol{x}_0$ is as close as possible to $\boldsymbol{x}_T^+$, we can have a stable limit cycle.

In practice, optimizing for only one foot step sometimes makes a stable controller for only a few foot steps, as the error diverges. When this happens, the optimization is run again, but this time with a longer foot steps, and initialized using the previously obtained parameters. Doing this iteratively often results in a controller that is stable for a large number of foot steps, even to infinity.

3.5 Results and Discussion

Table **??** shows the parameters for the ILQR controller for various criteria. The parameters for the extreme value of one criteria turns out to yield another extreme value of another criteria, hence the same parameters $\mathbf{\Phi}$. For example, the controller for maximum velocity here also yield the largest step length in our experiment.

Here we note that the step length predicted by the ILQR controller may not correspond to that happen during the simulation, due to the feet touching the ground before that. This is the case for the minimum step length (0.117 m). The desired step length input to the controller is 0.25, but the interaction between the controller and the actual system results in a different motion, which looks quite interesting (it can be seen in the supplementary video). Fig. **??** shows the plots of various variables of interest for the controller with velocity = 0.877 m/s.

3.5.1 Internal and External Perturbation

I do not have time to run the optimization for handling the perturbation, so I just try the controller with the parameters that I find most stable qualitatively. While playing with the optimization, I often observe that the motion is quite stable when the velocity is around 0.8m/s, so I optimize the ILQR controller around this speed. I obtain the parameters as $\boldsymbol{q}_0 = (-0.275, 0.319, 0.192), \dot{\boldsymbol{q}}_0 = (0.774, -0.874, 4.05), \alpha = 0.189, \text{step length} = 0.302, T = 0.382$.

With these parameters, the maximum internal perturbation is $\sigma_q = [0.1, 0.15, 0.1], \sigma_{qd} = [3, 2, 1.2]$", where σ_q and σqd are the noise standard deviation for the joint angles and velocities, respectively.

Note that the controller is more sensitive towards the error in the joint angle estimation as compared to the velocity. The noises are also limited at $\pm 3\sigma$. The allowable external perturbation, in the form of horizontal force at the hip, is $F = [-51, 18]N$.

We note here that ILQR controller is not particularly robust towards large perturbation. The feedback gain $\boldsymbol{K}_t$ is only optimal and valid near the optimal trajectory $(\boldsymbol{x}^*, \boldsymbol{u}^*)$. To perform more stably, one possible idea is to track the optimal trajectory using another controller, e.g. PD controller, or Quadratic Programming (QP) controller as in [**park2017high**]. I have tried implementing this, but I did not have time to find the good parameters for the QP.

Another possible approach is to recompute the ILQR at each iteration in Model Predictive Control (MPC) fashion. This will give good result and possibly the most stable. However, this requires a lot of computational time, and I cannot make the computation time below 1ms as would be required here. In our implementation, the ILQR requires 100ms for 1 iteration. A hierarchical controller may be an option, i.e. when the MPC is computed at slower frequency (e.g. 100 Hz).

Table 2: Parameters for ILQR controller

Criteria	**Value**	$\boldsymbol{q}_0$	$\dot{\boldsymbol{q}}_0$	α	**Step**	T
Vel	0.31	(-0.17, 0.27, 0.05)	(1.8 , 0.99, 0.02)	0.05	0.25	0.268
Vel	0.877	(-0.34, 0.39, 0.35)	(0.78, -1.45, 5.44)	0.34	0.36	0.421
Freq	2.2	(-0.34, 0.39, 0.35)	(0.79, -1.45, 5.44)	0.34	0.36	0.421
Freq	3.65	(-0.17 , 0.27 , 0.05)	(1.84, 0.99, 0.02)	0.05	0.25	0.268
Step	0.117	(-0.17, 0.27, 0.05)	(1.84 , 0.99, 0.02)	0.05	0.25	0.268
Step	0.372	(-0.34, 0.39, 0.35)	(0.79, -1.45, 5.44)	0.34	0.36	0.421

4 Conclusion

In this report, I have presented an optimal control formulation for a simplified 2D humanoid robot walking model. The parameters are optimized for various criteria (speed, step length, frequency), and robustness towards perturbation has been characterized. The optimal control problem is solved using ILQR, which is quite popular in robotics due to its fast computation. In addition, ILQR provides us not only with the optimal state and control trajectory, but also the feedback controller.

However, the formulation poses several difficulties. Firstly, choosing the objective function is not too easy. I set the objective function to mainly track the desired foot trajectory while keeping the torso at a specified angle. Secondly, the impact map results in discontinuity of the dynamics, which is a big problem for optimal control in general. I solve this by computing the optimal control for one foot step, and optimize the parameters such that the motion reaches a stable limit cycle.

While the resulting controller performs well, it can be made even more stable by considering several possible approaches. If the computation time can be made faster, we can recompute the ILQR at each iteration, i.e. doing Model Predictive Control (MPC). We can also not use the feedback gain of ILQR and instead use another controller, such as QP-based controller, or a shorter horizon MPC controller that can be computed faster.

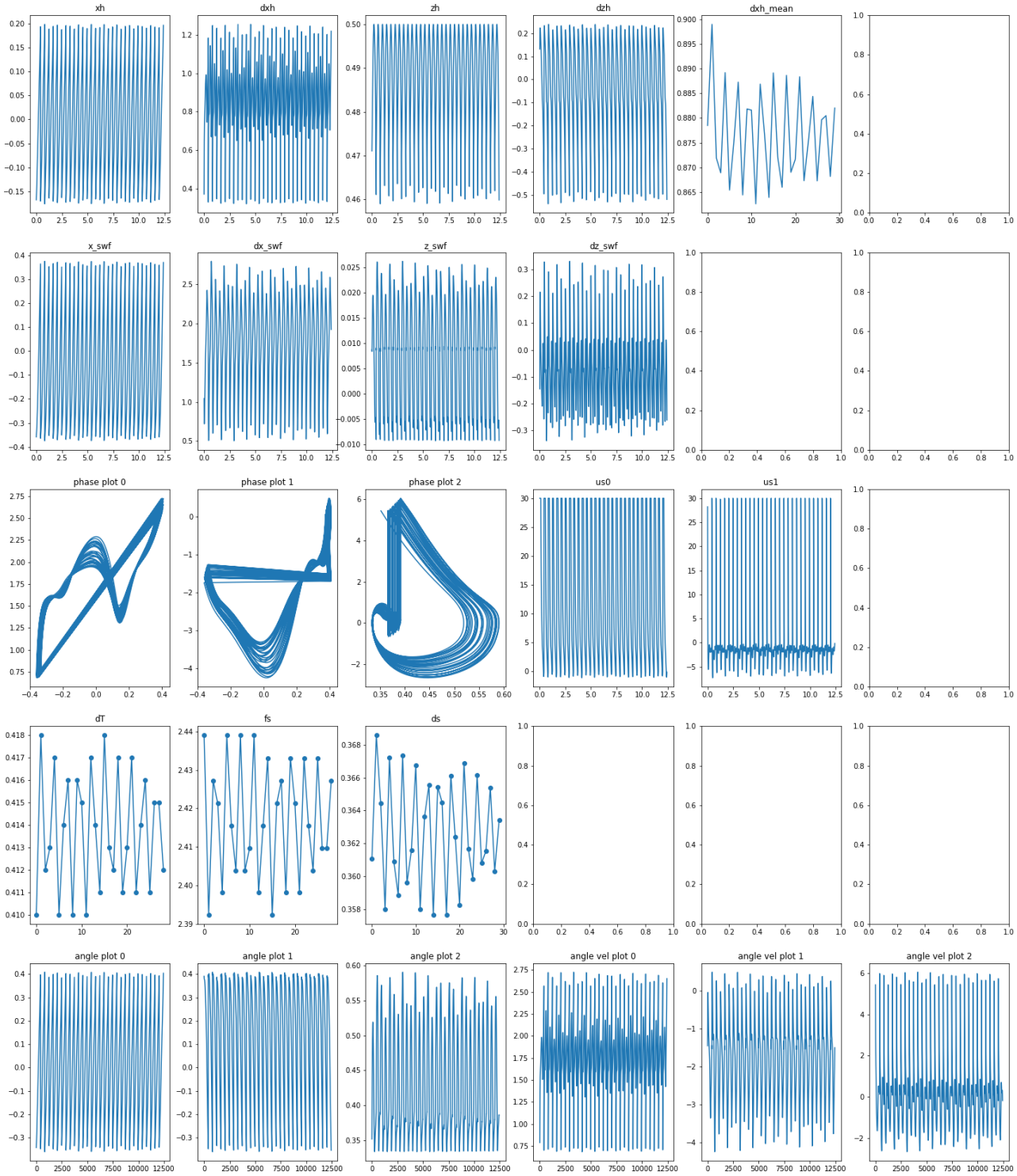


Figure 5: Various plots for the ILQR controller simulation with velocity = 0.877