Design and Control Co-Optimization of Pendular Robots

Yunsheng Tian
Computer Science & Artificial Intelligence Lab
Massachusetts Institute of Technology
Cambridge, MA 02139
yunsheng@csail.mit.edu

Jian Qian
Laboratory for Information & Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139
jianqian@mit.edu

***Abstract*—Joint optimization of both the design and control can significantly improve the performance of a robot, as indicated by the evolution of natural creatures. However, while optimal control is well studied in the robotics community, less attention is paid to finding the optimal robot design. In this paper, we present a general framework for co-optimizing the design and control of robots. Design parameters include but are not limited to mass, length, etc., and the controller is parameterized by a sequence of actuation. The co-optimization alternates between two stages in our framework: the control optimization stage and the design optimization stage. The control optimization stage can be done by any existing algorithm developed in the space of trajectory optimization, such as iterative linear quadratic regulator (iLQR) or model predictive path integral (MPPI). At the same time, we use gradient descent method (Adam) on design parameters for the design optimization stage. We apply our framework to simple pendular robots, including pendulum and acrobot, although it is extendable to other types of robots as well. The results demonstrate that the co-optimization of design and control improves the performance of the robots compared to fixed design, indicating that increasingly intelligent robot designs can be explored as an exciting future research direction.**

***Index Terms*—Co-optimization, trajectory optimization, design optimization, Acrobot, Pendulum**

I. Introduction

For a robot to achieve certain tasks successfully, it is well known that the control is important, but the design (i.e., structure) of the robot is also important. So we want to simultaneously optimize for the control and design of robots to figure out the optimal design of the robot for a given task and also the corresponding optimal policy for that design. The design of a robot, along with its control, determines the optimal performance for the robot to achieve certain tasks. But most of the previous works focus on improving robot control while the design optimization of robots is relatively less explored. Therefore, given certain tasks, we want to co-optimize both the design and the control of robots to achieve better performance than hand-designed robots. It would also be interesting to see how the optimal designs of robots get evolved and what insights we can obtain from the optimization that may help future robot design.

II. Related work

[1] co-optimizes the design parameters of articulated robots and their trajectories, which is very similar to the scope of this paper. [2] explores a similar idea about co-optimizing robot design and control, but uses reinforcement learning for both design and control optimization, which is much more computationally expensive for the overall co-optimization but has the potential to work on more complex tasks. [3] provides a differentiable way to do system identification, which has the potential of applying to co-optimization as well.

III. Problem formulation

Given the hard nature of the control problems, we don't hope to get elegant analytical solutions for the continuous to-go cost functions, so we formulate our problem in the following discretized optimization problem:

$$x^*_{1:T+1}, u^*_{1:T}, \theta^*_{1:T} = \underset{x_{1:T}, u_{1:T}, \theta_{1:T}}{\arg\min} \sum_{t=1}^{T} C_{\theta_t, t}(x_t, u_t)$$

subject to $x_{t+1} = f_{\theta_t}(x_t, u_t),\ x_1 = x_{\text{init}},\ \theta_{1:T} = \theta_c$

Here, $x_{1:T}$ are the states at time step t, $u_{1:T}$ are the control inputs at time step t, $\theta_{1:T}$ are the design paramteres, which are fixed to be θ_c through all time step t. For each time step t, there is a running cost $C_{\theta_t,t}(x_t, u_t)$. And our goal is to find the optimal $(\theta_c, u_{1:T})$ for a given x_1 that achieves the minimum cost. This formulation naturally hints at an iterative methods for co-optimizing $u_{1:T-1}$ and $\theta_{1:T-1}$. But obstacles lie at how to compute the gradient for θ. Thus we tried the following first algorithm.

A. *Robot Dynamics*

We adopt the derivations from the class textbook.

1) Pendulum: We consider a pendulum (Figure 1) with damping term constant being $b = 1$:

$$ml^2\ddot{\theta}(t) + mgl\sin\theta(t) = -\dot{\theta}(t) + u(t)$$

Thus, $\mathbf{x} = (\theta, \dot{\theta})^\top$ has,

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{\theta} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} \dot{\theta} \\ \frac{1}{ml^2}(-\dot{\theta} - mgl\sin\theta + u) \end{pmatrix}$$

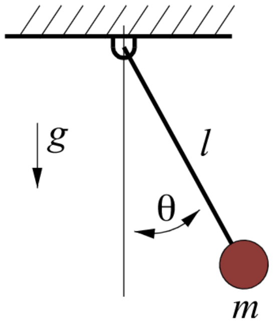


Figure 1: The pendulum robot

2) Acrobot: The acrobot dynamics uses the coordinate system as in the textbook and the state variable $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}})$, where $\mathbf{q} = (\theta_1, \theta_2)$, as shown in Figure 2. The design parameters under consideration are the mass and lengths of the two links, m_1, m_2, l_1, l_2. We assume the mass to be evenly distributed so that the center of mass is at the center of each link. Using the following lumped parameters:

$$I_1 = \frac{1}{2}m_1 l_1^2, I_2 = \frac{1}{2}m_2 l_2^2,$$
$$L_1 = m_2 l_1^2, L_2 = \frac{1}{2}m_2 l_1 l_2$$

and also,

$$s_1 = \sin(\theta_1), c_1 = \cos(\theta_1), s_2 = \sin(\theta_2), c_2 = \cos(\theta_2),$$
$$s_{1+2} = \sin(\theta_1 + \theta_2), c_{1+2} = \cos(\theta_1 + \theta_2)$$

Thus the coefficients are,

$$\mathbf{M}(\mathbf{q}) = \begin{pmatrix} I_1 + I_2 + L_1 + 2L_2c_2 & I_2 + L_2c_2 \\ I_2 + L_2c_2 & I_2 \end{pmatrix},$$
$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{pmatrix} -2L_2s_2\dot{q}_2 & -L_2s_2\dot{q}_2 \\ L_2s_2\dot{q}_1 & 0 \end{pmatrix},$$
$$\tau_g(\mathbf{q}) = \begin{pmatrix} -m_1gl_{c1}s_1 - m_2g(l_1s_1 + l_{c2}s_{1+2}) \\ -m_2gl_{c2}s_{1+2} \end{pmatrix},$$
$$\mathbf{B} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

In all, we have,

$$\ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1}(-\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}\tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u}),$$

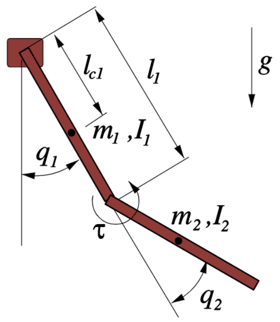


Figure 2: The acrobot robot

B. Time Integration

1) Forward Euler: Forward Euler is the most simple time integration method for approximating solutions of ordinary differential equations by temporal discretization. More specifically, suppose we are given the following dynamics,

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0$$

If we want to approximate the solution y, choose an appropriate time step h, then we have

$$\begin{aligned} y_{n+1} &= y_n + hf(t_n, y_n), \\ t_{n+1} &= t_n + h. \end{aligned}$$

Even though it is very simple to be implemented and computes fast, its accuracy is only of the first order, which means it hardly approximates the true continuous dynamics.

2) The Runge–Kutta method (rk4): The Runge-Kutta method provides a more accurate fourth-order approximation of the continuous dynamics, while being more computationally expensive. The approximation is given in the following form:

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right), \\ t_{n+1} &= t_n + h\,, \end{aligned}$$

for $n = 1, 2, ...$, where

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right), \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right), \\ k_4 &= f\left(t_n + h, y_n + hk_3\right). \end{aligned}$$

There is certainly a trade-off between speed and accuracy for choosing the time integration method. In our experiments, given time step $h = 0.1$, we find that forward Euler works well enough for the pendulum while using rk4 makes the acrobot significantly more solvable.

IV. Control Optimization

A. Controller

1) iLQR: The iLQR algorithm is a method that iteratively updates the control sequence to solve the following optimization problem:

$$\begin{aligned} \min_{u_{1:T}} \quad & \ell_f(x_{T+1}) + \sum_{t=1}^{T} \ell(x_t, u_t) \\ \text{subject to} \quad & x_{t+1} = \mathbf{f}_\theta(x_t, u_t), x_1 = x_{\text{init}} \end{aligned}$$

Here $\ell_f(x_{T+1})$ is the final loss at the final time step, $\ell(x_t, u_t)$ is the running loss, θ is the design parameters, and $x_{t+1} = \mathbf{f}_\theta(x_t, u_t)$ is the dynamics of the environment under design parameter θ. In each iteration, iLQR tries to approximate the solution of the following Bellman equation subject to the constraints:

$$V(x_t) = \min_{u_t} \ell(x_t, u_t) + V(x_{t+1})\,.$$

where the Q-value formulation with the constraints goes:

$$\begin{aligned} & \min_{u_t} Q(x_t, u_t), \quad \forall t \in [1, T] \\ \text{subject to } & Q(x_t, u_t) = \ell(x_t, u_t) + V(x_{t+1}) \\ & V(x_{T+1}) = \ell_f(x_{T+1}) \\ & x_{t+1} = x_t + \mathbf{f}_\theta(x_t, u_t) \\ & x_1 = x_{\text{init}}\,. \end{aligned}$$

Notice here, each term is differentiable, thus using linear approximations to each term, and verifying if the a loss reduction is going to occur, we have the iLQR algorithm as in Algorithm 1.

2) MPPI: The MPPI algorithm tries to address the control optimization problem in a probabilistic way. It assumes that the control sequence $u_{1:T}$ any algorithm decides to take is hit by a Guassian noise $\epsilon_{1:T} \sim \mathcal{N}(0, \Sigma)$, thus resulting in the final control input the system receives $v_{1:T}$. So the target of the this algorithm is to get the distribution of $v_{1:T}$ as close to optimal as possible. Namely the optimal distribution Q^* is defined as:

$$q^*(V) = \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}J(V)\right)p(V)\,,$$

where λ is a fixed parameters, η is the normalization constant, J is the total loss incurred by applying V,

Algorithm 1 iLQR

```
Input: $\mathbf{f}$: dynamics;
    $\ell$: running cost function;
    $\ell_f$: finial cost function;
    $x_{\text{init}}$: initial state;
    $u_{1:T}$: initial control sequence;
    $\lambda$: regularization term
1: $x_1 \leftarrow x_{\text{init}}$
2: Compute $x_{t+1} \leftarrow \mathbf{f}(x_t, u_t)$ for all $t$
3: $J \leftarrow \sum_{t=1}^{T} \ell(x_t, u_t) + \ell_f(x_{T+1})$
4: $\Delta J \leftarrow 0$
5: while $\Delta J >$1e-6 do
6:     $V_x, V_{xx} \leftarrow \nabla_x \ell_f(x_{T+1}), \nabla_{xx} \ell_f(x_{T+1})$
7:     $\Delta J \leftarrow 0$
8:     for $t = T...1$ do
9:         $\ell_x \leftarrow \frac{\partial \ell}{\partial x}|_{x_t,u_t}, \ell_u \leftarrow \frac{\partial \ell}{\partial u}|_{x_t,u_t}$
10:        $\mathbf{f}_x \leftarrow \frac{\partial \mathbf{f}}{\partial x}|_{x_t,u_t}, \mathbf{f}_u \leftarrow \frac{\partial \mathbf{f}}{\partial u}|_{x_t,u_t}$
11:        $Q_x \leftarrow \ell_x + \mathbf{f}_x^\top V_x$
12:        $Q_u \leftarrow \ell_u + \mathbf{f}_u^\top V_x$
13:        $Q_{xx} \leftarrow \ell_{xx} + \mathbf{f}_x^\top V_{xx} \mathbf{f}_x$
14:        $Q_{uu} \leftarrow \ell_{uu} + \mathbf{f}_u^\top V_{xx} \mathbf{f}_u$
15:        $Q_{ux} \leftarrow \ell_{ux} + \mathbf{f}_u^\top V_{xx} \mathbf{f}_x$
16:        $Q_{uu} \leftarrow Q_{uu} + \lambda I$
17:        $k_t \leftarrow -Q_{uu}^{-1} Q_u$
18:        $K_t \leftarrow -Q_{uu}^{-1} Q_{ux}$
19:        $V_x \leftarrow Q_x + K_t^\top Q_{uu} k_t + K_t^\top Q_u + Q_{ux}^\top k_t$
20:        $V_{xx} \leftarrow Q_{xx} + K_t^\top Q_{uu} K_t + K_t^\top Q_{ux} + Q_{ux}^\top K_t$
21:        $\Delta J -= Q_u^\top k_t + \frac{1}{2} k_t^\top Q_{uu} k_t$
22:    end for
23:    $x_1' = x_1$
24:    for $t = 1...T$ do
25:        $u_t' \leftarrow u_t + k_t + K_t(x_t' - x_t)$
26:        $x_{t+1}' \leftarrow \mathbf{f}(x_t', u_t')$
27:    end for
28:    $J' \leftarrow \sum_{t=1}^{T} \ell(x_t', u_t') + \ell_f(x_{T+1}')$
29:    if $J - J' > 0$ then
30:        $J \leftarrow J'$, $x_{1:T+1} \leftarrow x_{1:T+1}'$, $u_{1:T} \leftarrow u_{1:T}'$
31:        $\lambda *= 0.7$
32:    else
33:        $\lambda *= 2$
34:        $J \leftarrow J'$
35:    end if
36: end while
```

and $p(V)$ is the distribution of V when $u_{1:T} = 0$. The associated optimization is

$$U^* = \arg\min_U \mathrm{KL}(Q^* \| Q(U))$$

After some derivation, we have that the optimal u_t^* is retrieved by,

$$u_t^* = \int q^*(V) v_t \mathrm{d}V \,,$$

Bu since the optimal distribution is not available, thus the algorithm replaces it with important sampling. Suppose we have K noise trajectories $\epsilon_{1:T}^{1:K}$, we have the following update

$$u_t^{i+1} = u_t^i + \sum_{k=1}^{K} w(\epsilon_{1:T}^k) \epsilon_t^k \,,$$

where w is a weight assign to each noise trajectory as,

$$w(\epsilon_{1:T}) = \frac{1}{\eta} \exp\Bigg(-\frac{1}{\lambda}\Bigg(J(u_{1:T} + \epsilon_{1:T}) + \lambda \sum_{t=1}^{T} \frac{1}{2} u_t^\top \Sigma^{-1} (u_t + 2\epsilon_t)\Bigg)\Bigg) \,.$$

In all, we have the algorithm as in Algorithm 2.

V. Design Optimization

For fixed design parameter θ, once we have an optimal control trajectory $u_{1:T}^*$, we will be able to apply gradient descent along the following compute graph as in Figure 3. More specifically, we define,

$$\ell(\theta; u_{1:T}^*) = \ell_f(x_{T+1}) + \sum_{t=1}^{T} \ell(x_{t+1}, u_{t+1}^*) \,,$$

where $x_{t+1} = \mathbf{f}_\theta(x_t, u_t^*)$. Thus the partial derivative is,

$$\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell_f(x_{T+1})}{\partial x_{T+1}} \frac{\partial x_{T+1}}{\partial \theta} + \sum_{t=1}^{T} \frac{\partial \ell(x_{t+1}, u_{t+1}^*)}{\partial x_{t+1}} \frac{\partial x_t}{\partial \theta} \,.$$

In practice, we use PyTorch [4] to write a forward pass compute graph, then use Adam [5] optimizer to carry out the gradient descent across the compute graph to optimize the design parameters. In all, we have the general framework as in Algorithm 3.

Algorithm 2 MPPI

```
Input: $\mathbf{f}$: dynamics;
    $K$: number of samples;
    $T$: number of time-steps;
    $x_{\text{init}}$: initial state;
    $u_{1:T}$ initial control inputs;
    $\ell$: running cost;
    $\ell_f$: final cost;
    $\Sigma, \lambda$: hyper-parameters
1: while task not completed do
2:     $x_0 \leftarrow x_{\text{init}}$
3:     for $k = 1...K$ do
4:         $x \leftarrow x_0$;
5:         Sample $\epsilon^k_{1:T}$
6:         for $t = 1, ..., T$ do
7:             $x_{t+1} \leftarrow \mathbf{f}(x_t, u_t + \epsilon^k_t)$
8:             $J_k + = \ell(x_{t+1}, u_{t+1}) + \lambda u_t^\top \Sigma^{-1} \epsilon^k_t$
9:         end for
10:        $J_k + = \ell_f(x_{T+1})$
11:    end for
12:    $\beta \leftarrow \min_k J_k$
13:    $\eta \leftarrow \sum_{k=1}^{K} \exp(-\frac{1}{\lambda}(J_k - \beta))$
14:    for $k = 1, ..., K$ do
15:        $w(\epsilon^k_{1:T}) \leftarrow \frac{1}{\eta} \exp(-\frac{1}{\lambda}(J_k - \beta))$
16:    end for
17:    for $t = 1, ..., T$ do
18:        $u_t + = \sum_{k=1}^{K} w(\epsilon^k_{1:T}) \epsilon^k_t$
19:    end for
20: end while
```

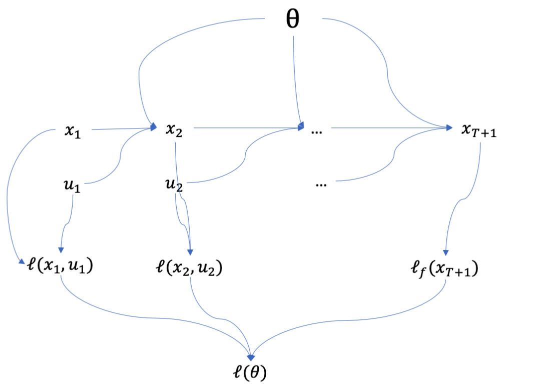


Figure 3: Compute graph for loss

Algorithm 3 General framework for co-optimization

```
Input: $\mathcal{A}(\cdot, \cdot; \cdot)$: control optimization algorithm
    (iLQR or MPPI) (taking an initial trajectory and
    the design parameters, then output the optimal
    trajectory);
    $\mathcal{B}$: one-step gradient descent algorithm
    (Adam);
    $\mathbf{f}(\cdot, \cdot; \cdot)$: dynamics;
    $\theta_{\text{init}}$: initial design;
    $x_{\text{init}}$: initial state;
1: $\theta \leftarrow \theta_{\text{init}}$
2: $x_1 \leftarrow x_{\text{init}}$, $x_{2:T+1} \leftarrow \mathbf{0}$, $u_{1:T} \leftarrow \mathbf{0}$
3: while task not completed do
4:     $u^*_{1:T} = \mathcal{A}(x_{1:T+1}, u_{1:T}; \theta)$
5:     $\theta \leftarrow \mathcal{B}(\ell(\cdot; u^*_{1:T}))$
6:     $u_{1:T} \leftarrow u^*_{1:T}$, $x_{2:T+1} \leftarrow \mathbf{f}(x_{1:T}, u_{1:T}; \theta)$
7: end while
```

VI. Experiment and analysis

A. Experiment Settings

The python implementation of all the algorithms mentioned above and all the experiment details can be found at https://github.com/yunshengtian/pendular-codesign, including all the hyperparameter settings for iLQR and MPPI controllers along with Adam optimizer.

In all of our experiments, we use the same parameters for physical simulation, where gravity $g = 9.81$, time step $h = 0.1$, and maximum horizon of the task $N = 50$. For time integration method, we use forward Euler for pendulum and rk4 for acrobot.

1) Pendulum: The design of a pendulum robot is paramaterized by mass m and length of the pole l. In our experiments, we use $(m, l) = (1, 2)$ as the initial design. We set the initial state of the pendulum as $\mathbf{x} = (\theta, \dot{\theta}) = (0, 0)$, and the target state as $\mathbf{x}^* = (\theta^*, \dot{\theta}^*) = (\pi, 0)$.

The cost functions of the pendulum robot is given in the following forms:

$$l(x,u) = 10(\theta - \theta^*)^2 + (\dot{\theta} - \dot{\theta}^*)^2 + 0.1u^2$$
$$l_f(x) = 10(\theta - \theta^*)^2 + (\dot{\theta} - \dot{\theta}^*)^2$$

2) Acrobot: The design of an acrobot robot is paramaterized by mass m_1 and length l_1 of the first pole connected to the origin, and also mass m_2 and length l_2 of the second pole that is connected to the first pole. In our experiments, we use $(m_1, m_2, l_1, l_2) = (1, 2, 1, 2)$ as the initial design. We set the initial state of the acrobot as $\mathbf{x} = (\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0.95\pi, 0, 0, 0)$, and the target state as $\mathbf{x} = (\theta_1^*, \theta_2^*, \dot{\theta}_1^*, \dot{\theta}_2^*) = (\pi, 0, 0, 0)$. The cost functions of the pendulum robot is given in the following forms:

$$\begin{aligned} l(x,u) &= 10((\theta_1 - \theta_1^*)^2 + (\theta_2 - \theta_2^*)^2) \\ &\quad + ((\dot{\theta}_1 - \dot{\theta}_1^*)^2 + (\dot{\theta}_2 - \dot{\theta}_2^*)^2) \\ &\quad + 0.1u^2 \\ l_f(x) &= 10((\theta_1 - \theta_1^*)^2 + (\theta_2 - \theta_2^*)^2) \\ &\quad + ((\dot{\theta}_1 - \dot{\theta}_1^*)^2 + (\dot{\theta}_2 - \dot{\theta}_2^*)^2) \end{aligned}$$

B. Control Optimization

Here we show the results of control optimization on the initial designs of robots.

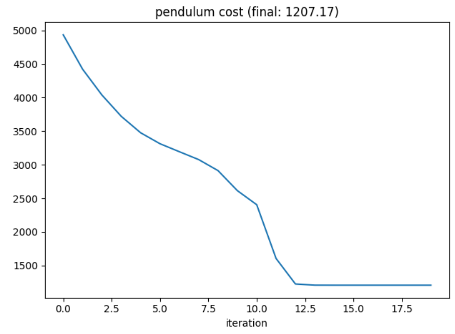


Figure 4: The cost change of pendulum trajectory during iLQR updates

1) Pendulum: Figure 4 shows the cost of pendulum trajectory during iLQR iterative updates, and it finally converges to the cost around 1207.17. Figure 5 shows the cost distribution of pendulum trajectory from 100 trials of MPPI controller. The mean of this cost distribution is around 1916.33, which is larger than the cost given by iLQR. The initial position of pendulum, along with the final positions of pendulum optimized by iLQR and MPPI are presented in Figure 6. We can see that iLQR successfully balances the pole at the top target position while MPPI fails.

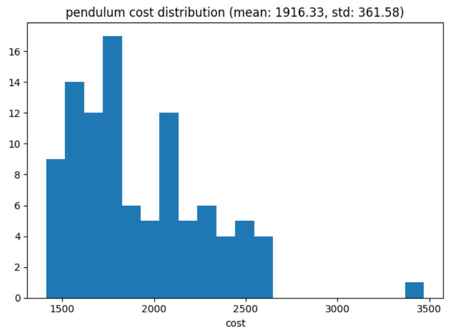
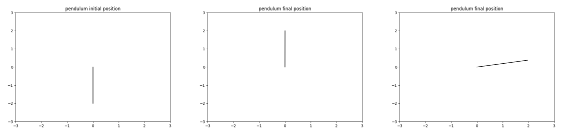


Figure 5: The cost distribution of pendulum trajectory from 100 trials of MPPI controller



(a) Initial (b) Final (iLQR) (c) Final (MPPI)

Figure 6: Initial and final positions of initial pendulum optimized by different controllers

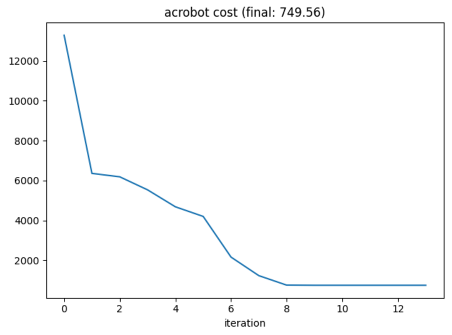


Figure 7: The cost change of acrobot trajectory during iLQR updates

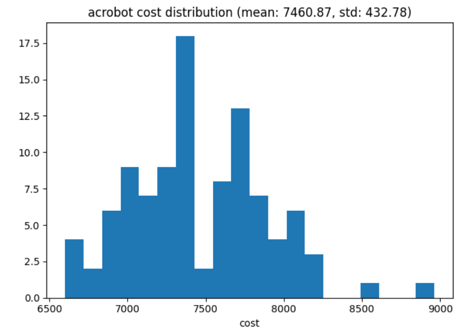
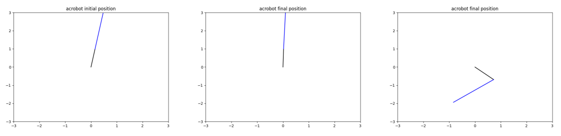


Figure 8: The cost distribution of acrobot trajectory from 100 trials of MPPI controller



(a) Initial (b) Final (iLQR) (c) Final (MPPI)

Figure 9: Initial and final positions of initial acrobot optimized by different controllers

2) Acrobot: Figure 7 shows the cost of acrobot trajectory during iLQR iterative updates, and it finally converges to the cost around 749.56. Figure 8 shows the cost distribution of acrobot trajectory from 100 trials of MPPI controller. The mean of this cost distribution is around 7460.87, which is significantly larger than the cost given by iLQR. The initial position of acrobot, along with the final positions of acrobot optimized by iLQR and MPPI are presented in Figure 9. We can see that iLQR successfully balances the poles at the top target position while MPPI fails.

C. Design Optimization

As mentioned in Section V, we use PyTorch library to construct a differentiable trajectory given the initial state and the optimal actuation sequence output by the controller. Since the design parameters of the robtos are also a part of the differentiable computation graph, we can run Adam optimizer to optimize the design parameters of robots. To further stabilize the co-optimization when design gets updated, we input the optimal trajectory solved from the previous design as a good initial guess to the controller optimizing for current design's trajectory.

1) Pendulum:

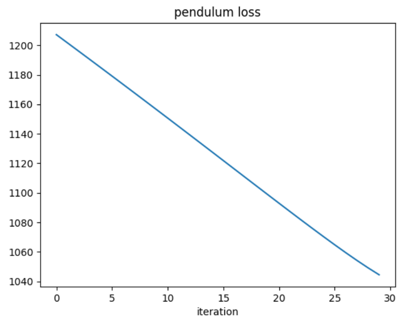
a) Cost: Figure 10 shows the cost change of pendulum trajectory during co-optimization using iLQR and MPPI controllers. We can conclude that first, the co-optimization works as the cost of pendulum keeps decreasing with better designs. Next, we find that with iLQR as controller, the co-optimization goes very smooth while MPPI controller induces higher variance during the co-optimization. More importantly, the cost of iLQR keeps strictly decreasing until some of the design parameters even become invalid (decrease to 0), as suggested by Figure 11. And finally, the cost of co-optimization using iLQR controller is constantly lower than that of MPPI controller.

b) Design: Figure 11 shows that the design change of pendulum during co-optimization using iLQR and MPPI controllers. It is interesting to see with the constraint of constant inertia, the mass of pendulum gradually decreases and correspondingly the length of the pole increases. We also find that if letting the co-optimization continue for more iterations, the mass will finally reaches 0 and the length becomes infinitely large. That might be because our physical model is over simplified, and we do not apply any bounds on the design parameters. But overall this corresponds perfectly to our intuition that a lighter pendulum will definitely be easier to control.

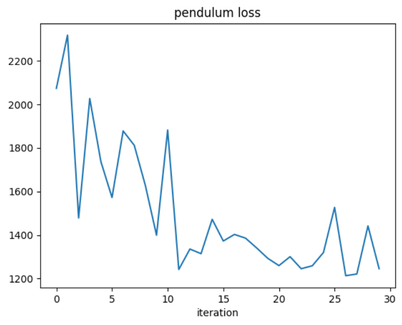
c) Initial and final position: Figure 12 shows the initial and final positions of co-optimized pendulum using iLQR and MPPI controllers. Comparing to Figure 6, we can see that the length of the pole obviously increases, as the result of co-optimization. Additionally, we can see that the final positions of optimized pendulums are more closer to the target position than the initial pendulums.

2) Acrobot:

a) Cost: Figure 13 shows the cost change of acrobot trajectory during co-optimization using iLQR controller. Here we do not use MPPI controller because MPPI is not even capable of balancing the

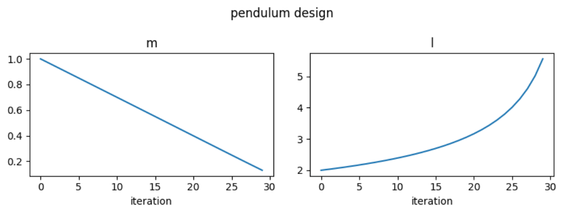


(a) iLQR

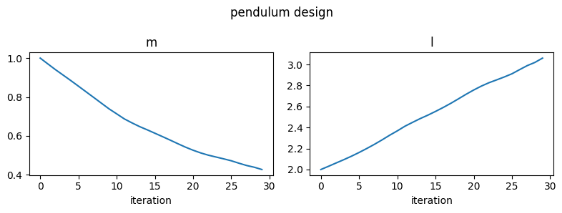


(b) MPPI

Figure 10: The cost change of pendulum trajectory during co-optimization using different controllers

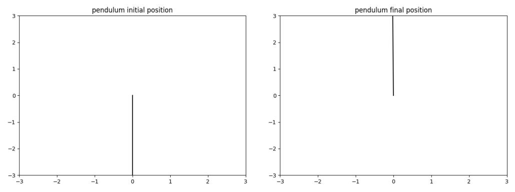


(a) iLQR

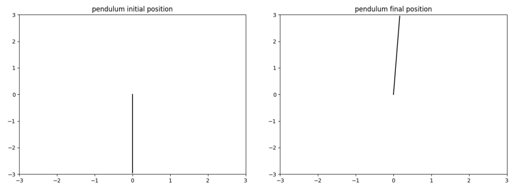


(b) MPPI

Figure 11: The design change of pendulum during co-optimization using different controllers



(a) Initial (iLQR) (b) Final (iLQR)



(c) Initial (MPPI) (d) Final (MPPI)

Figure 12: Initial and final positions of co-optimized pendulum using different controllers

initial design of acrobot. We can conclude that the co-optimization works as the cost of acrobot keeps decreasing with better designs.

b) Design: Figure 14 shows the design change of acrobot during co-optimization. Given the constraints of constant inertia of both poles and the constant sum of length of two poles, we find that as the result of co-optimization, m_1 increases and l_1 decreases while m_2 decreases and l_2 increases. This phenomenon indicates that the acrobot is learning to put more weight on its first pole to control the lighter second pole more easily.

c) Initial and final position: Figure 15 shows that the initial and final positions of co-optimized acrobot. The optimized acrobot can perfectly balance itself at the top target position.

VII. Discussion and future works

Three clear future directions for this co-optimizing framework is to make such a scheme work for legged robot, airplane and submarine, where analysis of contact dynamics, aerodynamics and fluid dynamics are required respectively. Also it would be interesting to try methods beyond gradient descent

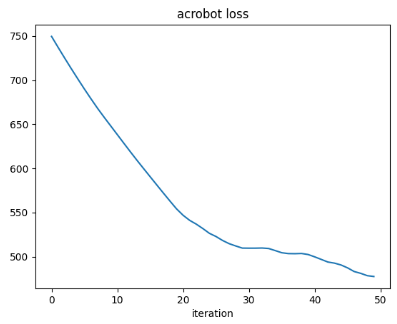


Figure 13: The cost change of acrobot trajectory during co-optimization

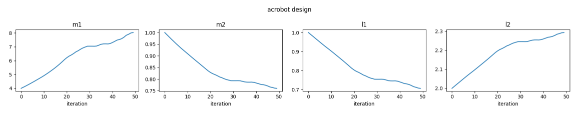
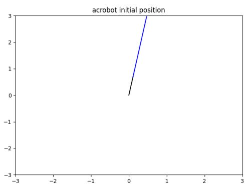
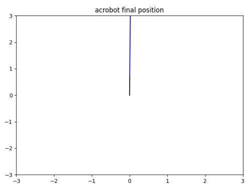


Figure 14: The design change of acrobot during co-optimization



(a) Initial



(b) Final

Figure 15: Initial and final positions of co-optimized acrobot

for the design optimization stage of the algorithm.

Reference

[1] A. Spielberg, B. Araki, C. Sung, R. Tedrake, and D. Rus, "Functional co-optimization of articulated robots," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5035–5042.

[2] D. Ha, "Reinforcement learning for improving agent design," *Artificial life*, vol. 25, no. 4, pp. 352–365, 2019.

[3] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," *arXiv preprint arXiv:1810.13400*, 2018.

[4] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.