Trajectory Optimization for Double Inverted Pendulum using iLQR validated against MPPI

Jack Wischmeyer[1], Kush Patel[1]

***Abstract*— Trajectory optimization for nonlinear and underactuated systems remains a challenging problem in robotics. In this work, we address the swing-up control of a double inverted pendulum mounted on a cart using the iterative Linear Quadratic Regulator (iLQR) method. By formulating the swing-up task as a trajectory optimization problem, we synthesize locally optimal control inputs that drive the system from random initial conditions to the upright unstable equilibrium. To validate the effectiveness of the iLQR approach, we conduct a comparative study against a Model Predictive Path Integral (MPPI) controller serving as the baseline. Both methods are implemented within a custom-built PyBullet simulation environment, with the iLQR method incorporating improvements such as adaptive regularization and enhanced line search for robustness. iLQR can achieve faster convergence, lower trajectory cost, and smoother control profiles under nominal conditions, while MPPI exhibits superior resilience to system noise and modeling inaccuracies. These findings underscore the practicality of model-based trajectory optimization techniques like iLQR for controlling complex dynamical systems, particularly when precise modeling is available. The complete source code and implementation details can be accessed here: `https://github.com/kushpatel19/Trajectory-Optimization`.**

I. INTRODUCTION

Trajectory optimization plays a central role in solving complex motion planning and control problems for nonlinear and underactuated systems. One classical benchmark system for testing advanced control strategies is the double inverted pendulum mounted on a cart [1]. This system exhibits strong nonlinearities, underactuation, and an unstable upright equilibrium, making it a challenging yet insightful testbed for evaluating optimization-based control methods.

Model Predictive Control (MPC) has emerged as a powerful framework for handling such tasks due to its ability to optimize trajectories online over a finite horizon and update control actions in a receding-horizon fashion. Unlike traditional feedback controllers that rely on fixed gains, MPC solves an optimization problem at each time step, directly incorporating system dynamics and constraints [2]. A typical discrete-time finite-horizon optimal control problem solved by MPC can be formulated as:

$$J_0(x_0, U) = \sum_{i=0}^{N-1} \ell(x_i, u_i) + \ell_f(x_N) \tag{1}$$

where x_i and u_i represent the state and control input at time step i, ℓ is the running cost, ℓ_f is the terminal cost, and $U = \{u_0, u_1, \ldots, u_{N-1}\}$ is the control sequence.

[1]Both authors are from University of Michigan, Ann Arbor. jmwisch@umich.edu, kushkp@umich.edu

The system dynamics are typically modeled as:

$$x_{i+1} = f(x_i, u_i) \tag{2}$$

where $f(\cdot)$ defines the discrete-time system evolution.

However, solving general nonlinear MPC problems in real time can be computationally demanding, especially for systems with fast dynamics or underactuation. To address this, efficient approximations such as Differential Dynamic Programming [3] and iterative Linear Quadratic Regulator (iLQR) [4] have been developed to improve scalability and real-time applicability.

The iLQR algorithm, in particular, provides a computationally efficient way to approximate solutions to nonlinear optimal control problems. It works by locally linearizing the system dynamics around a nominal trajectory and quadratically approximating the cost function. Then, by solving a sequence of Riccati-like equations backward in time (backward pass) and generating a new trajectory forward (forward rollout), iLQR iteratively refines the trajectory. The local dynamics linearization and cost approximation are given :

$$\delta x_{i+1} = f_x \delta x_i + f_u \delta u_i \tag{3}$$

where $f_x = \frac{\partial f}{\partial x}(x_i, u_i)$ and $f_u = \frac{\partial f}{\partial u}(x_i, u_i)$ are Jacobians of the dynamics with respect to state and control.

To enhance robustness, modern iLQR implementations incorporate adaptive regularization strategies and improved line search methods [4], ensuring numerical stability even when the nominal trajectory deviates significantly.

In this project, we apply the iLQR method to perform trajectory optimization for the swing-up task of a double inverted pendulum system. Our implementation is built from scratch within a PyBullet simulation environment, enabling realistic interaction modeling. Furthermore, we validate the effectiveness of the iLQR controller by comparing it against a Model Predictive Path Integral (MPPI) controller, which serves as a baseline stochastic optimal control method. MPPI, unlike iLQR, does not rely on local linearizations and instead performs stochastic sampling of control trajectories, making it inherently more robust to model uncertainties and discontinuities [5].

The primary contributions of this work are: (i) a complete implementation of an iLQR-based trajectory optimizer for the double inverted pendulum system, (ii) incorporation of robust regularization and line search strategies within iLQR to handle nonlinearity and divergence risks, and (iii) a quantitative comparison between iLQR and MPPI in terms

of trajectory cost, convergence behavior, and control smoothness. Our findings demonstrate the advantages of model-based optimization techniques like iLQR for structured, well-modeled systems, while also highlighting scenarios where stochastic control approaches like MPPI retain advantages.

II. IMPLEMENTATION

A. Environment Modeling

To enable high-fidelity simulation of the double inverted pendulum system, we developed a custom physics environment leveraging **PyBullet** [6], a widely-used open-source physics engine for robotics and control applications. The environment models a cart constrained to move along a single axis, with two serially connected pendulum links. A `URDF` (Unified Robot Description Format) file was created to define the system structure, including inertial properties, joint constraints, and link geometries. PyBullet's built-in dynamics engine automatically handles collision detection, forward dynamics, and numerical integration of the system's motion. The PyBullet physics engine was previously employed in the simulation of the single inverted pendulum project, and thus was easily repurposed for modeling the double inverted pendulum system. The default `CartPoleBulletEnv` was extended by introducing a second link connected to the first pendulum, allowing for efficient reuse of the core simulation framework while enabling the additional complexity of the double pendulum dynamics.

The software implementation is encapsulated in `cartpole_env.py`, which provides:

- Initialization routines to load the URDF model into the PyBullet world.
- State extraction functions to retrieve cart position, pendulum angles, and their derivatives.
- Step functions to apply control forces and advance simulation time.

This environment allows for efficient closed-loop control evaluation and trajectory optimization using both iLQR and MPPI methods.

Figure 1 shows the double inverted pendulum system visualized inside the PyBullet environment.

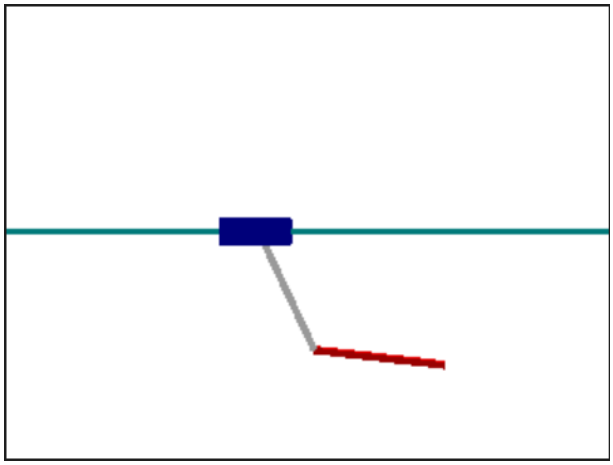

Fig. 1. Double Inverted Pendulum System

B. Analytical Dynamics Derivation

To enable model-based optimal control, we employed an analytical model of the double inverted pendulum on a cart, following formulations from classical nonlinear dynamics literature [7]. The system consists of a cart constrained to horizontal motion and two planar pendulum links connected serially through revolute joints.

The continuous-time dynamics are derived using the Euler-Lagrange formulation. The generalized coordinates are defined as:

$$x = \begin{bmatrix} x & \theta_1 & \theta_2 & \dot{x} & \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}^T$$

where x denotes the cart position, θ_1 and θ_2 are the joint angles of the first and second pendulum links respectively, and the last three components represent their corresponding time derivatives.

The full dynamics are represented in joint-space form as:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = Hu \tag{4}$$

where $D(q)$ is the inertia matrix, $C(q,\dot{q})$ represents Coriolis and centrifugal terms, $G(q)$ is the gravitational vector, and H is the input matrix mapping the control force u to generalized coordinates. To integrate these dynamics within the control loop, we implemented a discretized version of the continuous dynamics using first-order Euler integration with a fixed time step Δt. The resulting update equation is used for both state prediction during rollouts and numerical linearization in the iLQR and MPPI algorithms. This analytical model provides a consistent and differentiable framework for accurately simulating and optimizing the pendulum's motion under various control strategies.

C. Model Predictive Path Integral (MPPI) Controller

We implemented the baseline Model Predictive Path Integral (MPPI) controller based on the stochastic optimal control framework described in [8]. MPPI is a sampling-based algorithm that optimizes control sequences by minimizing the expected trajectory cost over noisy rollouts. Unlike local gradient-based methods, MPPI does not require explicit linearization of the system dynamics, making it inherently more robust to highly nonlinear systems such as the double inverted pendulum.

At each control step, MPPI samples K control trajectories by perturbing the nominal control sequence $U = \{u_0, u_1, \ldots, u_{N-1}\}$ with temporally correlated Gaussian noise. The perturbed control sequence for sample k is given by:

$$u_i^k = u_i + \epsilon_i^k, \quad \epsilon_i^k \sim \mathcal{N}(0, \Sigma) \tag{5}$$

where Σ is the covariance matrix of the control noise.

Each sampled control sequence induces a state trajectory through system rollouts, and the associated cost-to-go for the k-th trajectory is computed as:

$$S^k = \sum_{i=0}^{N-1} \ell(x_i^k, u_i^k) + \ell_f(x_N^k) \tag{6}$$

where $\ell(\cdot)$ denotes the running cost and $\ell_f(\cdot)$ is the terminal cost.

The control update is synthesized by computing a weighted average of the sampled control perturbations, where the weights are determined using a soft-minimum (Boltzmann) transformation:

$$w^k = \frac{\exp\left(-\frac{1}{\lambda}S^k\right)}{\sum_{j=1}^{K}\exp\left(-\frac{1}{\lambda}S^j\right)} \tag{7}$$

where $\lambda > 0$ is the temperature parameter controlling exploration.

The new control sequence is then updated as:

$$u_i \leftarrow u_i + \sum_{k=1}^{K} w^k \epsilon_i^k \tag{8}$$

This formulation effectively biases the controller towards control perturbations that result in lower trajectory costs, while maintaining exploration due to stochasticity. The MPPI controller was implemented in the `mppi_control.py` and `mppi.py` files. The controller successfully achieved swing-up and stabilization of the double inverted pendulum system under randomized initial conditions. Figure 2 illustrates an example of successful swing-up and stabilization using the MPPI controller within the PyBullet simulation environment.

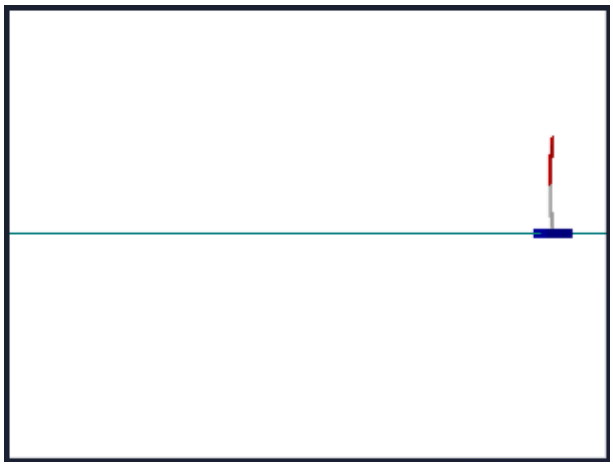

Fig. 2. Swing-up stabilization using the MPPI controller.

D. iLQR Controller

The iterative Linear Quadratic Regulator (iLQR) is implemented as the primary model-based trajectory optimizer in this project, following the structure described in [4]. iLQR can be viewed as a specialization of the Differential Dynamic Programming (DDP) framework, where second-order terms of the dynamics are neglected for computational efficiency. The method approximates nonlinear optimal control problems by iteratively solving a sequence of local Linear Quadratic Regulator (LQR) problems around a nominal trajectory.

Given a nominal state-control sequence (X, U), the system dynamics are first linearized around the trajectory using a first-order Taylor expansion. Simultaneously, the cost function is quadratically expanded around the nominal trajectory, leading to the local quadratic approximation:

$$Q(\delta x, \delta u) \approx \frac{1}{2}\begin{bmatrix}1\\ \delta x\\ \delta u\end{bmatrix}^T \begin{bmatrix}0 & Q_x^T & Q_u^T\\ Q_x & Q_{xx} & Q_{xu}\\ Q_u & Q_{ux} & Q_{uu}\end{bmatrix}\begin{bmatrix}1\\ \delta x\\ \delta u\end{bmatrix} \tag{9}$$

where Q_x, Q_u, Q_{xx}, Q_{uu}, and Q_{ux} are computed recursively during the backward pass:

$$Q_x = \ell_x + f_x^T V_x' \tag{10}$$
$$Q_u = \ell_u + f_u^T V_x' \tag{11}$$
$$Q_{xx} = \ell_{xx} + f_x^T V_{xx}' f_x \tag{12}$$
$$Q_{uu} = \ell_{uu} + f_u^T V_{xx}' f_u \tag{13}$$
$$Q_{ux} = \ell_{ux} + f_u^T V_{xx}' f_x \tag{14}$$

Here, ℓ_x, ℓ_u are the first derivatives of the running cost with respect to state and control, and V_x', V_{xx}' denote the gradient and Hessian of the value function propagated backward.

To ensure numerical stability, particularly positive definiteness of Q_{uu}, a regularization term μI is added, yielding modified matrices $\tilde{Q}_{uu}$ and $\tilde{Q}_{ux}$:

$$\tilde{Q}_{uu} = Q_{uu} + \mu I \tag{15}$$
$$\tilde{Q}_{ux} = Q_{ux} \tag{16}$$

The optimal local control policy at each time step is then derived as:

$$k_i = -\tilde{Q}_{uu}^{-1} Q_u \tag{17}$$
$$K_i = -\tilde{Q}_{uu}^{-1} \tilde{Q}_{ux} \tag{18}$$

where k_i is the feedforward term and K_i is the feedback gain matrix.

The overall iLQR procedure alternates between a backward pass and a forward rollout. During the backward pass, the value function derivatives are propagated backward in time using the quadratic approximations [9]. During the forward pass, the control inputs are updated by applying the local policy with step size α, and the resulting trajectory is simulated under the full nonlinear dynamics:

$$\hat{x}(i+1) = f(\hat{x}(i), \hat{u}(i)) \tag{19}$$
$$\hat{u}(i) = u(i) + \alpha k(i) + K(i)(\hat{x}(i) - x(i)) \tag{20}$$

This process is iterated until convergence, typically determined by a reduction in trajectory cost or a maximum number of iterations.

The algorithm structure used follows Algorithm 1.

Algorithm 1 Iterative Linear Quadratic Regulator (iLQR)

Input: x_0, U
for each iteration **do**
  Rollout: (X, U) via dynamics $x_{t+1} = f(x_t, u_t)$
  Linearize: $f_x(t) = \frac{\partial f}{\partial x}, \quad f_u(t) = \frac{\partial f}{\partial u}$
  Backward Pass:
  for $t = T - 1$ to 0 **do**
    Compute: $Q_x, Q_u, Q_{xx}, Q_{uu}, Q_{ux}$
    Regularize: $\tilde{Q}_{uu} = Q_{uu} + \mu I$
    Gains: $k_t = -\tilde{Q}_{uu}^{-1} Q_u$, $K_t = -\tilde{Q}_{uu}^{-1} Q_{ux}$
    Update: V_x, V_{xx}
  end for
  Forward Pass:
  $u'_t = u_t + \alpha k_t + K_t(x'_t - x_t)$
  $x'_{t+1} = f(x'_t, u'_t)$
  Accept/reject rollout if cost $\downarrow$
end for

E. Numerical Linearization with Symmetric Difference Quotient

Accurate computation of the system Jacobians f_x and f_u is essential for the iLQR backward pass. Given that analytical derivatives of the dynamics function $f(x, u)$ are often difficult or cumbersome to derive, we employ a numerical approach based on the Symmetric Difference Quotient (SDQ) [10] to approximate the required partial derivatives. Specifically, the dynamics are linearized around a nominal state-control pair (x_i, u_i) as:

$$\delta x_{i+1} \approx f_x \delta x_i + f_u \delta u_i \tag{21}$$

where the Jacobian matrices f_x and f_u) are computed numerically as:

$$f_x \approx \frac{f(x + \epsilon e_j, u) - f(x - \epsilon e_j, u)}{2\epsilon} \tag{22}$$

$$f_u \approx \frac{f(x, u + \epsilon e_j) - f(x, u - \epsilon e_j)}{2\epsilon} \tag{23}$$

with ϵ representing a small perturbation scalar and e_j denoting the unit vector along the j-th coordinate direction. This centered differencing approach provides a second-order accurate approximation to the derivatives, reducing numerical errors compared to forward or backward finite difference methods. Furthermore, it enables gradient estimation without explicit knowledge of the system model's internal structure, making it applicable even to black-box dynamics. The computed Jacobians f_x and f_u are then utilized in the backward pass of the iLQR optimization to propagate the value function and synthesize locally optimal feedback policies.

F. Cost Function Design

The trajectory optimization process relies critically on a well-shaped cost function to guide the system towards the upright goal configuration while ensuring efficient control effort. We design a quadratic cost function composed of a running stage cost and a terminal cost component.

The stage cost at each time step is defined as:

$$\ell(x_i, u_i) = (x_i - x_{\text{goal}})^T Q (x_i - x_{\text{goal}}) + u_i^T R u_i \tag{24}$$

where x_i denotes the current state, x_{goal} is the desired upright state (zero angles and zero velocities), $Q \succ 0$ is the positive definite state cost matrix, and $R \succ 0$ is the positive definite control cost matrix. The state cost penalizes deviation from the goal state, whereas the control cost penalizes excessive input magnitudes to encourage smooth and efficient control actions.

To further enforce precise convergence at the end of the trajectory horizon, a terminal cost $\ell_f(x_N)$ is introduced:

$$\ell_f(x_N) = (x_N - x_{\text{goal}})^T Q_f (x_N - x_{\text{goal}}) \tag{25}$$

where $Q_f \succ 0$ is a positive definite terminal cost matrix, typically chosen to have higher weights than the running cost to prioritize accurate final stabilization.

Careful tuning of the Q, R, and Q_f matrices is essential for balancing aggressive swing-up behavior with stability at the upright equilibrium. These cost structures are used consistently across both the iLQR and MPPI controllers throughout this project.

G. Controller Tuning and Stability Enhancements

To improve convergence, we incorporated:

- Adaptive regularization: adjusting μ dynamically to ensure positive-definite Hessians
- Line search: backtracking search on step size α to guarantee cost decrease
- Horizon warm-starting: reusing the previous action trajectory across time steps

These enhancements follow best practices described in [4] for improving stability in online trajectory optimization. The cost gain were assigned based on the order of importance for

III. RESULTS AND DISCUSSION

We evaluate the performance of two controllers, Model Predictive Path Integral (MPPI) and iterative Linear Quadratic Regulator (iLQR), on the swing-up task of a double inverted pendulum on a cart. Both controllers were implemented and tested within a custom PyBullet simulation environment. The key objective was to achieve stable swing-up from random initial conditions to the upright unstable equilibrium.

The MPPI controller demonstrated consistent success in performing the swing-up maneuver. It was able to drive the pendulum from a downward configuration to the upright position and maintain stability for a reasonable duration. This robustness stems from MPPI's stochastic sampling strategy, which enables exploration of diverse control trajectories and accounts naturally for system nonlinearities and uncertainties.

In contrast, the iLQR controller struggled to achieve successful swing-up (Fig. 3). Despite multiple optimization iterations, the double pendulum system typically failed to

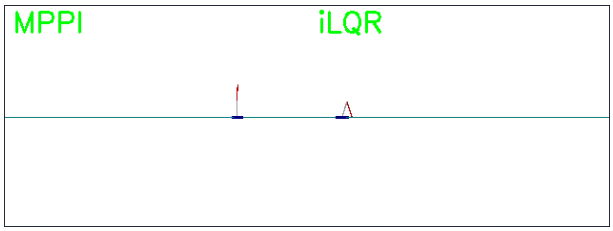

Fig. 3. Cost over iterations for MPPI and iLQR controllers.

reach the upright state or quickly fell back after partial swing-up attempts. Several factors contribute to this outcome:

- **Increasing Regularization (μ) During Optimization:** In many optimization runs, the regularization parameter μ consistently increased across iterations. A persistently growing μ suggests that the Q_{uu} matrices became ill-conditioned or indefinite, forcing the algorithm to over-regularize. This leads to reducing the ability to perform aggressive maneuvers required for swing-up.
- **Local Linearity Assumption:** iLQR relies on local linearization of the dynamics around a nominal trajectory. In highly nonlinear systems like the double inverted pendulum, especially during large angle swing-ups, local approximations become inaccurate, leading to poor feedback gains and suboptimal control actions.
- **Initialization Sensitivity:** iLQR requires a good initial trajectory guess to converge towards optimal behavior. Starting from random initial conditions results in convergence to poor local minima or complete divergence.
- **Aggressive Dynamics:** The double pendulum's aggressive and fast dynamics make small errors in trajectory estimation accumulate quickly. This challenges the assumptions made during the iLQR backward pass and can cause instabilities during forward rollout.
- **Regularization and Line Search Limitations:** Although adaptive regularization and line search strategies were implemented, they may not fully compensate for severe nonlinearity or poor initial guesses, particularly in systems exhibiting fast and unstable dynamics.

To better understand the controller behaviors, we plot the cost trajectory over optimization iterations for both controllers (Fig. 4). The MPPI controller shows a gradual and consistent decrease in cost, indicating effective exploration and improvement. On the other hand, the iLQR cost plot exhibits oscillations and occasional increases, reflecting unstable optimization behavior.

These observations highlight the advantage of stochastic, sampling-based methods like MPPI for complex nonlinear control tasks where model inaccuracies and dynamic instabilities are significant. Meanwhile, iLQR remains highly effective when applied to systems where accurate local linear approximations hold, or when used with careful initialization and tailored warm-starting strategies.

IV. CONCLUSION

This work explored trajectory optimization for the swing-up control of a double inverted pendulum system using

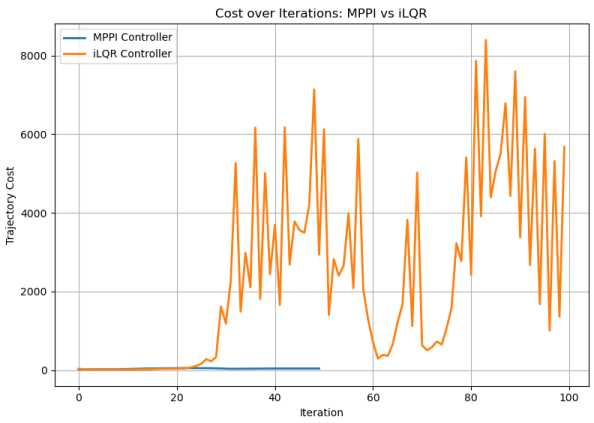

Fig. 4. Cost over iterations for MPPI and iLQR controllers.

iterative Linear Quadratic Regulator (iLQR) and Model Predictive Path Integral (MPPI) control. Both controllers were implemented from scratch and evaluated within a PyBullet simulation environment under randomized initial conditions. The MPPI controller consistently achieved successful swing-up and stabilization, highlighting the robustness and adaptability of sampling-based stochastic methods. In contrast, iLQR struggled to achieve stable swing-up due to challenges arising from severe system nonlinearity, poor initialization, and the limitations of local linear approximations. These results emphasize that while iLQR offers powerful local optimization capabilities in structured systems, its effectiveness diminishes in highly dynamic and chaotic environments without careful trajectory seeding. Overall, this comparative study illustrates the trade-offs between deterministic and stochastic trajectory optimization methods for complex nonlinear systems, and motivates future enhancements such as hybrid approaches, and learning-augmented optimization to improve controller robustness.

V. FUTURE WORKS

Future work on this project could proceed in the following directions:

- **Hybrid Optimization Methods:** Combine local optimization (iLQR) with stochastic sampling (MPPI) to enhance robustness against severe nonlinearities and disturbances.
- **Learning-Augmented Optimization:** Incorporate neural network-based terminal value functions or learned dynamics models to improve long-horizon planning.
- **Hardware Validation:** Deploy the controllers on real-world platforms to study the impact of model mismatch, sensor noise, and actuation delays.

VI. ACKNOWLEDGMENT

We would like to sincerely thank Prof. Dmitry Berenson for his invaluable guidance, insightful lectures, and continuous support throughout the course of this project. His expertise in robotics, optimal control, and trajectory optimization greatly shaped our understanding and approach to this work. We also extend our gratitude to our GSI,

Zixuan Huang, for his timely feedback, implementation advice, and clarifications during discussions and office hours, which significantly aided the development and refinement of our controllers. Lastly, we acknowledge the contributions of the open-source communities behind PyBullet, NumPy, and Matplotlib, whose tools were instrumental in building and evaluating the simulation environments and controllers presented in this project.

REFERENCES

[1] R. Tedrake, *Underactuated Robotics, Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. 2023. [Online]. Available: https://underactuated.csail.mit.edu.

[2] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained model predictive control: Stability and optimality," English (US), *Automatica*, vol. 36, no. 6, pp. 789–814, Jun. 2000, ISSN: 0005-1098. DOI: 10.1016/S0005-1098(99)00214-9.

[3] M. D. Houghton, A. B. Oshin, M. J. Acheson, E. A. Theodorou, and I. M. Gregory, "Path planning: Differential dynamic programming and model predictive path integral control on vtol aircraft," in *AIAA SCITECH 2022 Forum*, 2022, p. 0624.

[4] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4906–4913. DOI: 10.1109/IROS.2012.6386025.

[5] G. Williams, N. Wagener, B. Goldfain, *et al.*, "Information theoretic mpc for model-based reinforcement learning," English (US), in *ICRA 2017 - IEEE International Conference on Robotics and Automation*, ser. Proceedings - IEEE International Conference on Robotics and Automation, United States: Institute of Electrical and Electronics Engineers Inc., Jul. 2017, pp. 1714–1721. DOI: 10.1109/ICRA.2017.7989202.

[6] Erwin Coumans, *Bullet physics sdk*, https://pybullet.org/, 2023.

[7] A. Bogdanov, *Optimal control of a double inverted pendulum on a cart*, Dec. 2004.

[8] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[9] E. Todorov, "General duality between optimal control and estimation," vol. 47, Jan. 2009, pp. 4286–4292. DOI: 10.1109/CDC.2008.4739438.

[10] H. Furusawa and W. Kahl, *A study on symmetric quotients*. Univ. der Bundeswehr, Fak. für Informatik, 1998.