Learning to Backflip: An RL approach

Nishanth Rao
nishrao@seas.upenn.edu

Nithasree Somanathan
nithi10@seas.upenn.edu

Kunpeng Liu
kunpengl@seas.upenn.edu

***Abstract*—Model-free reinforcement learning algorithms are robust to uncertainties like sensor noise and external disturbances, without requiring any knowledge about the system dynamics. Model-based control approaches inherently lack this robustness, and thus rely on additional blocks like sensor estimation and disturbance rejections. The same is established in this project, where a model-free DQN agent and a model-based MPC controller are designed to perform a backflip on a bipedal-SLIP model. The robot is spawned in `MuJoCo` and the states are corrupted by adding sensor noise along with random external disturbance. While the DQN agent is able to perform a successful backflip, the model based-MPC fails to accomplish this, thus demonstrating the robustness of model-free DQN methods over model-based controllers.**

I. Introduction

In the field of robotic control, a significant challenge arises from the reliance on accurate state information, which is often corrupted by sensor noise and model inaccuracies. This *uncertainty* severely impacts the effectiveness of traditional model-based control systems. Model-free RL approaches show some resilience against these uncertainties, as they have to interact with the environment to *learn* what the optimal behavior is, without any knowledge of the model dynamics. This process allows them to *generalize well*, making them to perform well in uncertain environments.

This project delves into a comparative analysis between the two control paradigms: a *model-based controller*, consisting of the Model Predictive Controller (MPC) along with Iterative Linear Quadratic Regulator (iLQR) for trajectory generation, and a *model-free controller* employing a Deep Q-Network (DQN) agent. The central goal is to achieve a good backflip maneuver on a Bipedal Spring-Loaded Inverted Pendulum (SLIP) along with uncertainties like sensor noise and small external disturbances. For this purpose, a custom environment is developed in `MuJoCo`, where the bipedal-SLIP model is spawned and desired backflip behavior is achieved.

II. Preliminaries

This section *briefly develops* the theory of model-free reinforcement learning algorithms. A more rigorous detailed explanation can be found in [1], [2].

A. Model-based and Model-free RL

In general, *the value function* $v_\pi(s)$ of a state s under the stochastic policy $\pi(a|s)$ is defined as the *expected reward accrued* when starting from a state s, and following the policy $\pi(.)$ till the goal state is reached. Similarly, the *q-function* $q_\pi(s,a)$ of a state-action pair is defined as the *expected reward accrued* when taking an action a in a state s, and thereafter following the policy $\pi(.)$:

$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle| S_t = s\right] \tag{1a}$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle| S_t = s, A_t = a\right] \tag{1b}$$

These functions satisfy a recursive relationship between the current state s and all the possible successor states and actions (s', a') called the *Bellman equation*:

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(s_{t+1})|\, S_t = s\right] \tag{2a}$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})|\, S_t = s, A_t = a\right] \tag{2b}$$

$$q^*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q^*(s_{t+1}, a')\middle| S_t = s, A_t = a\right] \tag{2c}$$

Unfortunately, to evaluate the above equations, one requires full knowledge of the environment dynamics $p(s', r|s, a)$ and the probability distribution of $\pi(.)$. The following two methods are used to *estimate the value functions* when knowledge of the environment dynamics is unknown (model-free).

Monte Carlo methods: Here, the *expectation* in Eq. (2) is approximated by averaging *sample returns* $G_t = \sum_{k=0}^{T}\gamma^k R_{t+k+1}$ over *many* episodes by interacting with the environment. After the end of an episode, the value functions for the states visited are updated and the process is repeated. In the limit of infinitely many episodes, Monte-Carlo methods converge to the true value functions.

Temporal Difference methods: Instead of waiting for the entire episode's termination to estimate the value functions, *TD methods estimate* them *online*, by using sample returns G_t whenever they are available for a particular visited state s.

In general, TD methods converge faster than Monte Carlo methods. They are also more useful in online robotic-control settings where one cannot wait till the entire "episode" terminates.

B. Q-learning and DQN

Q-learning is a TD algorithm, where the immediate reward R_{t+1} is used to update the q-value of a state-action pair, by using the Bellman equation of the *optimal q-function* q^* (2c) as an update rule:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_{a'} q(s_{t+1}, a') - q(s_t, a_t)\right] \tag{3}$$

As Eq. (3) tries to directly approximate the optimal Bellman equation, convergence is seen to be faster.

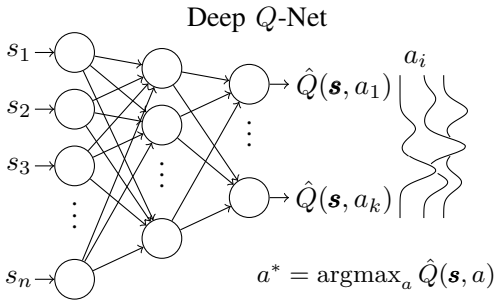

Fig. 1. A fully connected Network serves as an approximation to the q-function. The output is an ensemble of continuous q-functions for each possible (*discrete*) actions a_i.

So far, all the equations assume that the states and actions are *discrete*. In these cases, q-functions can be seen as a *lookup table*, where for a given state-action pair, a q-value is assigned. However, there exist many systems where the states of a system happen to be *continuous* in nature. In these settings, a *neural-network* can be used to estimate the q-function.

Fig. 1 shows a Deep Q-Network that takes in an input $\boldsymbol{s} \in \mathbb{R}^n$ and outputs a continuous function $Q(\boldsymbol{s}, a_i)$ for each action $a_i \in \boldsymbol{a} \in \mathbb{R}^k$. The *optimal action* can then be computed as $a^* = \text{argmax}_a\, Q(\boldsymbol{s}, a)$. Note that the *action-space* is discrete.

III. Methodology

This section briefly discusses the details of the model-based and model-free controllers used, along with the training details of the DQ-Network.

A. Model Based Control

A system model becomes essential to any model-based controller design. Here, the dynamics of the bipedal-SLIP model are simply that of a freely falling body with a *double integrator* for the torque control of both legs:

$$\ddot{z} = -g \tag{4a}$$

$$\ddot{\theta}_1 = \tau_1 / I \qquad \ddot{\theta}_2 = \tau_2 / I \tag{4b}$$

where z denotes the vertical distance of the SLIP's center of mass from the ground, I denotes the moment of inertia and $\theta_{1,2}$ denotes the angle of the two legs w.r.t the vertical axis. The system state is defined as $x = [z\ \theta_1\ \theta_2\ \dot{z}\ \dot{\theta}_1\ \dot{\theta}_2]^T$ and $u = [\tau_1\ \tau_2]^T$. The backflip is triggered when the SLIP jumps to its highest point; after the backflip is done, it continues to just bounce and stay balanced.

Iterative Linear Quadratic Control is used initially to generate the nominal backflip trajectory by providing the control input guesses, initial state, and the final state of the system. The trajectory (desired states and inputs) generated by the iLQR is then passed through a *Model predictive controller* (MPC) (Eq. 5) which interacts with the actual simulator model and outputs the control input required for the backflip.

$$J = \sum_{k=0}^{T_f} \left[\|x_k - x_{d,k}\|_Q^2 + \|u_k - u_{d,k}\|_R^2 \right] \tag{5a}$$

$$s.t. \quad x_k = f(x_k, u_k) \tag{5b}$$

$$u_{min} \leq u \leq u_{max} \tag{5c}$$

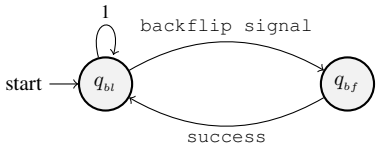

Fig. 2. A finite-state-machine governs when to balance q_{bl} vs backflip q_{bf}.

B. Model-Free Control

Model-free RL methods learn a controller directly by maximizing the cumulative reward obtained by interacting with the environment. Thus, it is crucial to design the right reward functions that can *guide* the agent towards learning the desired behavior. To perform a backflip maneuver, the following reward function is used:

$$r_t = k_{spin}\dot{\theta} + k_{alive} + \lambda_c k_{upright} - k_\tau |\tau| - k_{die} \tag{6a}$$

$$\lambda_c = \begin{cases} 1 & \text{if } \texttt{contact} \text{ and } |\theta| < 20^\circ \\ 0 & \text{otherwise} \end{cases} \tag{6b}$$

i.e., the agent gets a small reward k_{alive} for being alive, or a large negative reward k_{die} if it fails to land properly. It gets a reward of $k_{spin}\dot{\theta}$ i.e., an incentive to spin and a reward $k_{upright}$ when the SLIP lands upright ($|\theta| < 20^\circ$). It receives a small negative reward $k_\tau|\tau|$ for the control effort it exerts to spin around.

Another behavior that is desired is the SLIP just keeping its balance in the presence of external disturbances and uncertainties like sensor noise. This way, a *finite-state-machine*, as illustrated in Fig. 2 can be constructed that controls when to backflip vs. balance. The reward function for balancing is the same as in Eq. (6), with $k_{spin} = 0$.

C. DQN Training Details

Two separate *DQN* agents are trained to balance and backflip the SLIP system respectively. The network is trained to output the q-values for each of these behaviors by minimizing the *l2-squared norm* between the estimated q-value vs. the q-value obtained by applying the *Q-learning* update rule (with $\gamma = 0.995$) of Eq. (3):

$$\min_{\boldsymbol{W}} L(\boldsymbol{W}_t) = \left\| r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{W}_{t-1}) - \hat{Q}(s, a; \boldsymbol{W}_t) \right\|_2^2 \tag{7}$$

The DQN agent eventually becomes *greedy* w.r.t choosing the action that maximizes the cumulative reward $a^* = \text{argmax}_a\, Q(s, a; \boldsymbol{W})$ which is evident from the structure of Eq. (3). However, *exploration* is crucial in the early stages of network training so that the agent is *maximally* aware of the different rewards it can seek. Thus, an ϵ-greedy strategy is employed, where the agent chooses a random action a_i with a probability of ϵ or becomes *greedy* with a probability of $1-\epsilon$. As training progresses, the value of ϵ is reduced, in the hope that the agent *has understood what is optimal*, after which it can become greedy w.r.t this optimal policy π^*.

IV. Experiments

A. Trajectory generation

For the model-based MPC controller, a nominal trajectory for backflip is provided. The initial state $x_0 = [2.5, 0, 0, 0, 0, 0]$ is chosen, where the bipedal-SLIP jumps to the highest $z = 2.5m$ point (here, the legs are 1.5m above the ground). The final state $x_f = [1.27, 2\pi, 2\pi, -2.45, 0, 0]$ is chosen to be satisfied at $t = 0.5s$. This is because the two legs of the bipedal-SLIP touch the ground at $t = 0.553s$. Satisfying $\dot{\theta}_i = 0$ at the end ensures that the legs don't continue to swing after a full rotation. The generated trajectory is shown in Fig. 3.

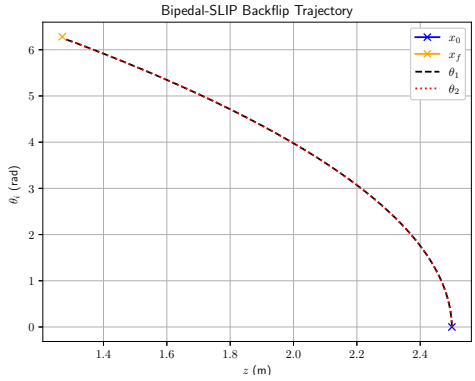

Fig. 3. The figure shows the variation in θ_i vs z.

B. DQN Details and Performance

A fully connected network with 2-hidden layers, each with 1024 neurons is chosen. The DQN is initially *trained on single-SLIP model*, where the state $x \in \mathbb{R}^6$ is $6-dimensional$ (x = $[y, z, \theta, \dot{y}, \dot{z}, \dot{\theta}]$). Unlike in Model-Based Controller, here the forward $y - axis$ is also considered, as there are disturbances and sensor noise due to which the SLIP can hop front or back. The action space $a \in [-1, 1] \in \mathbb{R}$ is discretized into 21 possible actions $a_i \in \{-1., -0.9, \ldots, 0.9, 1.\}$ and thus, there are 21 output neurons.

Observation: It was seen that training the DQN agent with a bipedal-SLIP resulted in some suboptimal policies. This is because adding another dimension to the action space caused a *slower exploration* during the initial training, along with an increase in the network output (42 neurons). In many cases, due to asynchronous movements of the legs, the model jumped to lower heights (and just kept jumping as the height to perform a backflip wasn't enough), causing it to accrue massive rewards pertaining only to being alive. Perhaps, a more sophisticated reward function may have helped; however, it was observed that training the agent on a single SLIP resulted in faster convergence to the optimal policy; it also transitioned nicely to a bipedal-SLIP during testing, where the same action was applied to both the legs.

To solve the problem of *catastrophic forgetting*, the network employs a replay buffer that stores sample state-action-reward transitions, which enables the network to learn from its past

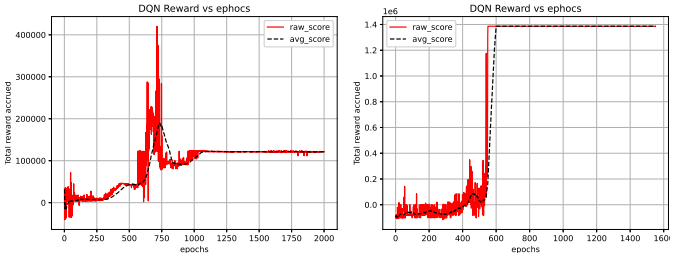

Fig. 4. Reward vs epochs during DQN training. The reward stabilizes at a constant value eventually. The left figure corresponds to the backflipping, while the right figure corresponds to balancing.

experiences [3]. The network is trained using `Adam` optimizer in `PyTorch`, with the `MSELoss` defined in Eq. (7) and the weights are updated using *stochastic gradient descent*. The reward curve for DQN training (2000 epochs) is shown in Fig. 4.

Around epoch 750, it seems that the agent does well, however, the agent was just spinning, randomly landing and spinning till it died. The DQN agent eventually learns to do a backflip, followed by an upright landing and then performing backflips till it falls on the ground. Perhaps with greater exploration, the agent may have continued to land and backflip, but mostly resulting in much longer training times (and a small chance that it explores the *right thing*!). For the task of balancing, the agent learns the optimal policy around 500 epochs and remains constant. The episode is terminated after $10,000$ time steps (around 10 secs), as the agent can balance for arbitrarily long times.

C. Simulation Results

A custom `MuJoCo` [4] environment is developed for both single SLIP as well as bipedal-SLIP (as shown in Fig. 6). The states of the model are corrupted using *white Gaussian noise*. External disturbances (along the y-axis) are added randomly ($\pm 1N$). A `Python Wrapper` is then written to interface OpenAI's `gym` [5] library with this custom `MuJoCo` environment. The time step is chosen as $dt = 0.001$ with RK4 numerical integrator. The leg consists of a *sliding joint* with `stiffness` enabled that allows it to act like a spring. The hip joint(s) are actuated with `damping` enabled. The whole system is confined to moving along the $y - z$ plane with the x-axis pointing out of the screen. Thus, the model cannot fall sideways; if the base (red-ball) touches the ground, the *episode is terminated* and is reset back to the upright position, dropped from a height of $2.5m$ (with the legs at a height of $1.5m$ above the ground). The action values $a \in [-1, 1]$ from `gym` are mapped with a gear ratio of 30.

Model-based Controller: The desired trajectory and control commands generated by the iLQR algorithm is given to the MPC controller that interacts with the `MuJoCo` simulator. **Without** any external disturbances and sensor noise, the controller evidently performs a backflip successfully, by tracking the desired trajectory without any problems. The MPC tracking performance is shown in Fig. 7. However, with uncertainties,

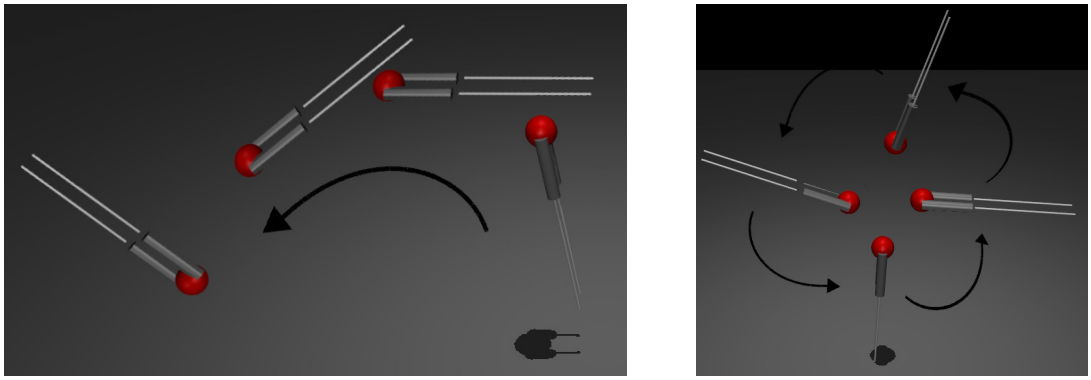

Fig. 5. Figures show the snapshopts of the bipedal-SLIP in `MuJoCo` simulator. The left figure illustrates an unsuccessful backflip when using a model-based controller with uncertainties and disturbances. The right figure illustrates a successful backflip maneuver with the trained DQN agent. Full Video: https://drive.google.com/file/d/1NHEhHNqFRhMN6Eb94Cts4mJyb09CfVv6/view?usp=drive_link

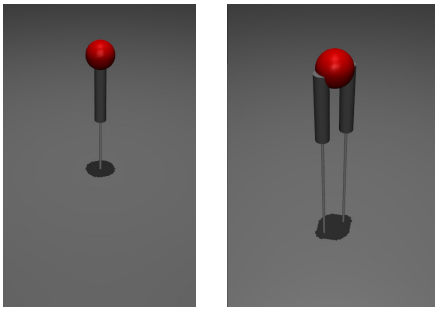

Fig. 6. Single-SLIP and Bipedal-SLIP in `MuJoCo`. The horizontal, forward, and vertical, upward directions (w.r.t SLIP model) correspond to y, z-axis respectively.

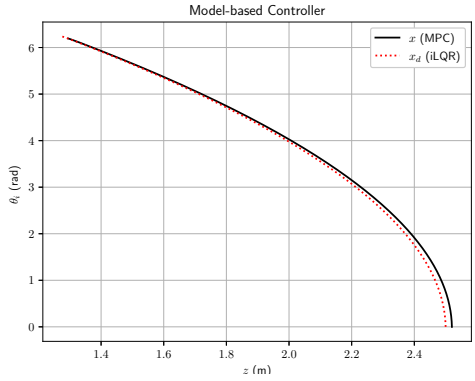

Fig. 7. MPC Tracking performance. Differences in the initial state are seen as during reset, the states are randomly assigned around the nominal reset state values (to induce robustness for the DQN agent).

the bipedal-SLIP fails to stand upright while trying to perform a backflip (illustrated in Fig. 5). This is obvious, as the controller isn't capable of handling the robustness. With supporting blocks like sensor estimation and disturbance rejection, maybe it might perform a successful backflip; however, that discussion is outside the scope of this project.

Model-free Controller: As the DQN is trained using *domain randomization* from the replay buffer, the agent learns to become robust against external disturbances and sensor noise. The DQN agent switches from *balance* to *backflip* according to the FSM diagram shown in Fig. 2. Moreover, the DQN agent is now controlling the bipedal-SLIP instead of the single-SLIP model with which it was previously trained. This shows that model-free RL methods may be successfully up-scaled to some extent and are robust in general. The snapshots of the backflip are best shown in Fig. 5.

V. Conclusion

This project presents a comparative analysis of model-based and model-free control approaches in robotic control, specifically for executing a backflip maneuver with a bipedal-SLIP model. Model-free DQN was found to be more robust against sensor noise, and external disturbances and executed a backflip successfully, while MPC wasn't able to execute a successful backflip in the presence of these external disturbances. This highlights the potential of model-free RL methods, in managing real-world uncertainties in robotic systems. Future work could explore enhancing model-based approaches with disturbance rejection and sensor estimation mechanisms to improve their adaptability successfully. Further, more sophisticated model-free RL algorithms could be developed for more complex robotic systems like humanoids.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] D. Bertsekas, *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[6] J. Zhang, J. Shen, Y. Liu, and D. W. Hong, "Design of a jumping control framework with heuristic landing for bipedal robots," *arXiv preprint arXiv:2304.00536*, 2023.

[7] X. Xiong and A. D. Ames, "Sequential motion planning for bipedal somersault via flywheel slip and momentum transmission with task space control," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 3510–3517.