Reinforcement Learning for Cat-Inspired Midair Robot Reorientation

Kerlina Liu
kerlina@mit.edu

Ethan Chun
elchun@mit.edu

Abstract

Falling is an ever present risk to mobile robots in the real world. To minimize fall induced damage, we proposed a system which imitates the righting reflex of cats, enabling a robot to land in an upright position when dropped from arbitrary orientations. We modeled our robots after the two-cylinder model of a cat [6] and initialized the robot at arbitrary poses to demonstrate consistent foot facing landings. We propose to solve this problem using two reinforcement learning methods: Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC). Additionally, we solve the problem using trajectory optimization methods (direct collocation) and compare against the RL methods. While PPO does not learn to fully right itself, SAC eventually does.

1 Introduction

Four legged robots such as Boston Dynamics' Spot [11] and MIT's Mini Cheetah [7] are built to travel on varying terrain, which may lead to steep drops in elevation. A fall from great enough heights will break any robot; however, the robot's orientation at landing can reduce this damage. Drawing inspiration from the remarkable ability of domesticated cats to reorient themselves in midair, known as the "falling cat problem" or their "righting reflex," we develop methods to minimize fall damage for robots by enabling them to land on their feet.

The righting does not require outside forces, relying on the conservation of angular momentum [2], thus the solutions we have found for robots also apply to zero-gravity environments, such as reorienting astronauts or satellites in space.

While solutions using trajectory optimization exist [5], the righting reflex must be computed and executed before making contact with a surface, often eliminating such methods due to the time required to compute a sufficient trajectory. Additionally, such optimizations are specific to bounded sets of initial states and final states are therefore not robust. We proposed building a policy using PPO and SAC which, trained offline, can be executed to successfully land a falling robot.

2 Related Work

Our work builds upon a substantial body of research in cat-inspired robot reorientation. These approaches vary in their interpretation of the dynamics of a falling cat, including whether the tail, legs, or torso is responsible for mid-air reorientation [2]. Additionally, a variety of methods are used to solve for the trajectory of the robot, including numerical optimization methods, as well as reinforcement learning.

2.1 Reorientation Methods

Tail-based reorientation. Tang et al. (2022) recently proposed a method that employs a 3-DoF articulated tail to reorient a quadrupedal robot during a short fall [12]. This approach is largely independent of the base quadruped design, placing the majority of mechanical complexity in the tail. However, the added mass of a non-functional tail may impede the robot's performance, making other approaches more desirable when the entire robot can be redesigned.

Leg-based reorientation. Similar to tail-based reorientation, these methods use an appendage of the cat to aid in reorientation. Rudin et al. (2021) and Kurtz et al. (2021) both employ the motion of the cat's legs to drive its rotation [8, 10]. Additionally, they add weighted boots to the robot's legs to increase their moments of inertia. While these approaches are clean in their implementation, they contradict the design philosophy of modern quadrupeds, which rely on low-inertia limbs for agility. Thus, we seek to develop a method that does not depend on an external appendage.

Two-cylinder reorientation. T. R. Kane and M. P. Scher proposed a now widely accepted solution to the falling cat problem in 1969, modeling a falling cat as two cylinders connected by a driven universal joint [6]. They proved that two cylinders actuated in this manner can reproduce the dynamics of a falling cat. Unlike the previous methods, this solution exploits the intrinsic mass of the cat's body to generate rotation, but it requires a robot designed specifically to test this problem. Iwai and Matsunaka (2012) and Ge et al. (2019) both adopted this approach, numerically simulating a simplified two-cylinder system and demonstrating mid-air reorientation [3, 4]. Given our ability to design a robot from scratch, we aim to follow these approaches and create a two-cylinder-based robot.

2.2 Trajectory Generation

Optimization methods. The falling cat problem is a non-holonomic problem, requiring specific orderings of states to achieve a desired orientation. A variety of works leverage this property, proposing optimization-based methods to generate trajectories for cat-inspired robots. Ge et al. (2019) use a quasi-newton method to solve for a two-cylinder cat [3], Iwai and Matsunaka (2012) model the cat using a port-controlled Hamiltonion system [4], Liang et al. (2016) uses particle swarm optimization [9], and Kurtz et al. (2021) [8] use trajectory optimization. Given the ease of implementation, we plan to use trajectory optimization as our baseline method.

Learning approaches. In contrast to previously discussed approaches, Rudin et al. (2021) use Proximal Policy Optimization (PPO) trained for 200 million steps with a dense reward function to reorient their robot. This method is highly effective on a large quadrupedal robot, demonstrating sim to real transfer and robust performance. Notably however, their robot utilizes high mass legs to reorient itself which is not ideal for robot agility. To the author's knowledge, no works have combined deep reinforcement learning with a two-cylinder robot model, presenting an interesting research direction.

3 Problem Formulation

3.1 Robot Model

We model our robot after Kane and Scher's two cylinder model for cats [6]. This model describes a system with a pair of cylinders, A and B, connected by an actuated universal joint with limited travel. We define a revolute joint to join our model to the world frame and restrict whole-body rotation to one axis. This helps us simplify our analysis. Additionally, since there is no universal joint in our simulator, we replace the universal joint with a set of four revolute joints.

Specifically, the central revolute joint is defined in the y-z plane with value $\theta \in [-\pi, \pi]$ with respect to the positive z axis. This denotes the overall rotation of the cat robot.

To reconstruct this universal joint, we define joints $\varphi_A, \varphi_B, \gamma_A, \gamma_B$ for both cylinders. Intuitively, γ denotes the "waist" of the cat model while φ denotes the "spinal rotation". We limit the total bend of γ_A and γ_B to $\gamma_A, \gamma_B \in [-\pi/3, \pi/3]$ where γ_A is measured from the $-x$ axis and γ_B is measured from the $+x$ axis. We limit φ_A, φ_B to $\varphi_A, \varphi_B \in [-\pi/2, \pi/2]$ from the plane defined by γ_A and γ_B. This plane is shown by the red bar in Fig. 1 These joint limits mimics the limitations of a real cat [4, 6]. See Fig. 1 for a graphical illustration of our model.

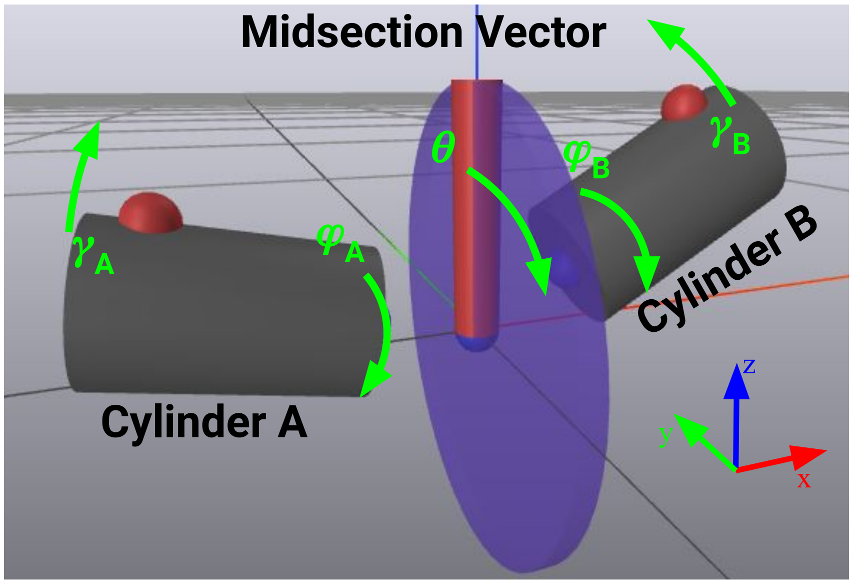

Figure 1: Robot Model – We simply the robot as a pair of connected cylinders. Both cylinders rotate about about their axis of symmetry φ_A, φ_B and pivot in the plane containing both cylinders and the midsection vector γ_A, γ_B. Additionally, the whole assembly can pivot about the normal to the blue cylinder θ. The red dots on each cylinder denote the leg direction of our robot. Our state space is the position and velocity of $\{\theta, \varphi_A, \varphi_B, \gamma_A, \gamma_B\}$ while our action space is the commanded position of $\{\varphi_A, \varphi_B, \gamma_A, \gamma_B\}$.

3.2 State and Action Space

We define a robot state of size ten where the first five values are the position of each joint $(\theta, \gamma_A, \gamma_B, \varphi_A, \varphi_B)$ and the last five are their angular velocities. These are described as follows:

- θ: Denoting the rotation of the entire body, it is an angle in the world's y-z plane, measured with respect to the +z axis. $\theta \in [-\pi, \pi]$
- γ_A, γ_B: denoting the contribution of each cylinder to the bend angle at the waist of the cat. These are measured in the y-z plane of θ's frame, measured with respect to θ's x-axis. $\gamma_A, \gamma_B \in [-\pi/3, \pi/3]$
- φ_A, φ_B: denoting the angles between the cylinders' normals to θ. The cylinders' normals are defined by the direction that the feet of the robot point. $|\varphi_A - \varphi_B| \in [0, \pi]$

We define our action space as a set of four scalars, $\gamma_{d_A}, \gamma_{d_B}, \varphi_{d_A}$, and φ_{d_B}. These are desired positions which are sent to a PD controller running in our simulator. That controller calculates the torques for joints $\gamma_A, \gamma_B, \varphi_A, \varphi_B$.

- $\gamma_{d_A}, \gamma_{d_B}$: Desired position of γ_A and γ_B, the bend angle at the waist from either half of the model
- $\varphi_{d_A}, \varphi_{d_B}$: Desired position of φ_A and φ_B, the rotation of top/bottom half of the model

3.3 Goal State

To ensure that our robot lands on its feet, we define the ground as the $-\hat{z}$ direction. To account for a range of acceptable landing orientations, we include an error term of $\pi/8$ from the axis. Our goal is to generate and execute a trajectory that achieves a successful landing within 0.5 seconds, corresponding to a fall from a height of approximately 4 feet or 1.226 meters.

3.4 Objective Function

We want the z coordinates of both cylinders to point in a downward facing direction ($-z$ in the world frame) in as short a time as possible. To do so, we defined two reward functions, one sparse, and one dense.

Our sparse reward was defined as follows:

$$R_{\text{sparse}}(q,t) = \begin{cases} 100 & \left((\theta+\varphi_A)^2 < (\pi/8)^2\right) \text{ AND } \left((\theta+\varphi_B)^2 < (\pi/8)^2\right) \\ \\ 0 & \text{otherwise} \end{cases}$$

The $\pi/8$ represent our aforementioned error term and the squares provide a differentiable alternative to absolute values.

Our dense reward is defined similarly, but with an added term based on the angle of the robot from our goal:

$$R_{\text{dense}}(q,t) = R_{\text{ori}} + R_{\text{goal}} \tag{1}$$

$$R_{\text{ori}} = ((\varphi_A + \theta \mod 2\pi) - \pi)^2 + ((\varphi_B + \theta \mod 2\pi) - \pi)^2 \tag{2}$$

$$R_{\text{goal}} = \begin{cases} 100 & \left((\theta+\varphi_A)^2 < (\pi/8)^2\right) \text{ AND } \left((\theta+\varphi_B)^2 < (\pi/8)^2\right) \\ \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Intuitively, R_{ori} is the sum of the squared angular error of each cat's leg from the goal state.

These reward functions are used by our learning algorithms as the value of a given state. Each learning algorithm is ultimately minimizing the objective function:

$$\min(J(q))$$
$$J(q) = -\mathbf{E}_t[log\pi_\theta(a_t|s_t)R_t]$$

Note that both PPO and SAC incorporate additional reward terms to regularize exploration and update their internal polices during training.

$$\theta_{t+1} \leftarrow \theta_t + \alpha \cdot \text{clip} \quad \left(\hat{\nabla}\theta J(\theta_t) + \beta \cdot \text{var}\pi_{\theta_t}(\hat{\nabla}_\theta J(\theta_t))\right) \tag{4}$$

$$r = \sum_{t=0}^{\infty} \gamma^t \left(R_t + \alpha \cdot H(\pi_\theta(a_t|s_t))\right) \tag{5}$$

Where $H(\cdot)$ is measures the entropy within the policy.

3.5 Base Learning Algorithm

Proximal Policy Optimization. Proximal Policy Optimization (PPO) is known for its simplicity and effectiveness. It is an on-policy algorithm, meaning that data is collected as our agent explores the world. We elected to use PPO as is due to this simplicity, data efficiency, and easy of implementation.

Soft Actor Critic. In contrast to PPO, Soft Actor-Critic (SAC) is an off-policy actor-critic reinforcement learning algorithm. Given that we are training fully in sim, we can collect data extremely efficiently, mitigating any concerns with the data usage of either algorithm. We decided to try SAC as it explicitly encourages exploration by using entropy regularization, hopefully reducing the effects of local minimum in our state space.

3.6 Simulation Environment

We used Drake [13] as our physics simualtor and OpenAI gym [1] as our RL wrapper. To simulate free fall, we configured the Drake environment to have no gravity. Additionally, we jointed that robot to world with a single rotational degree of freedom about the x axis. This allowed us to reduce the computational resources required to generate trajectories for the robot.

4 Results

We explored the falling cat problem with three different approaches: Trajectory optimization, Proximal Policy Optimization, and Soft Actor Critic. We used trajectory optimization as our known baseline while testing PPO and SAC.

4.1 Trajectory Optimization

Cao et al. proposed that the two cylinder model of a cat follows a trajectory consisting of a series of bends without rotating the two halves, where the cat first bends forward ($\theta < \pi$) before bending to one side, then backwards ($\theta > \pi$), and finally bending to the other side, ultimately landing upright [2]. However, Iwai and Matsunaka's work on the falling cat problem suggests that the "no rotation" constraint in the two-cylinder model is not necessary, which is why our model includes state variables $\varphi_{A,B}$ [4]. To gain intuition on this "ideal" trajectory, we utilized trajectory optimization methods.

Specifically, we used trajectory optimization with direct collocation to find an "ideal" solution to the falling cat problem. The trajectory produced shows cylinders A and B hinging (γ_A and γ_B) in opposite directions so that the bodies still lie in a straight line but outside of the world's x-y plane. Then, both cylinders hinge into a "crunch" formation and straighten while rotating (φ_A and φ_B) both cylinders to land upright. This is illustrated in Fig. 2.

Empirically, we found that trajectory optimization was very sensitive to the initial condition, only producing viable trajectories when the robot was oriented in close to perfectly supine positions. However, when trajectories were produced, the resulting movement was extremely smooth.

4.2 PPO

Drawing inspiration from Rudin et al, our first test algorithm was PPO.

After training PPO for two million steps using a dense reward, we found that PPO fails to reach the goal state from states the fully supine position. In these cases, PPO manages to rotate the body only partially, where both feet do not point downwards. This trajectory is illustrated in Fig. 2 while the reward curve is show in Fig. 3. However, we found that when started nearer to the goal state, with the feet within $2\pi/3$ from the downward direction, PPO was able to reach the goal state.

Analysing the trajectories generated by PPO in Fig. 2, we see that the trajectories follow the start of the trajectory generated by trajectory optimization, but stop after a single bend and rotation. We hypothesize that this single motion is enough to reach the goal state when within $2\pi/3$ but not sufficient when initialized farther. Furthermore, we suspect that any deviations from this trajectory will produces lower rewards for initialization within $2\pi/3$ from the goal state for a short time, leading PPO to find this as a local minimum.

4.3 SAC

After attempting PPO, we decided to explore SAC as an alternative algorithm.

Dense Reward. We found that when given a dense reward function, SAC would learn to reach the goal states from similar states as PPO, but failed for the fully supine position. As with PPO, we suspect that this is due to the presence of local minimum. See Fig. 3 for an illustration of this failure mode.

Sparse Reward with Dense Pre-training In an attempt to circumvent this failure mode, we opted to use the same model trained with a dense reward, but switch to a sparse reward. This may be interpreted as a variant of curriculum learning. In contrast to PPO, we found that after training for

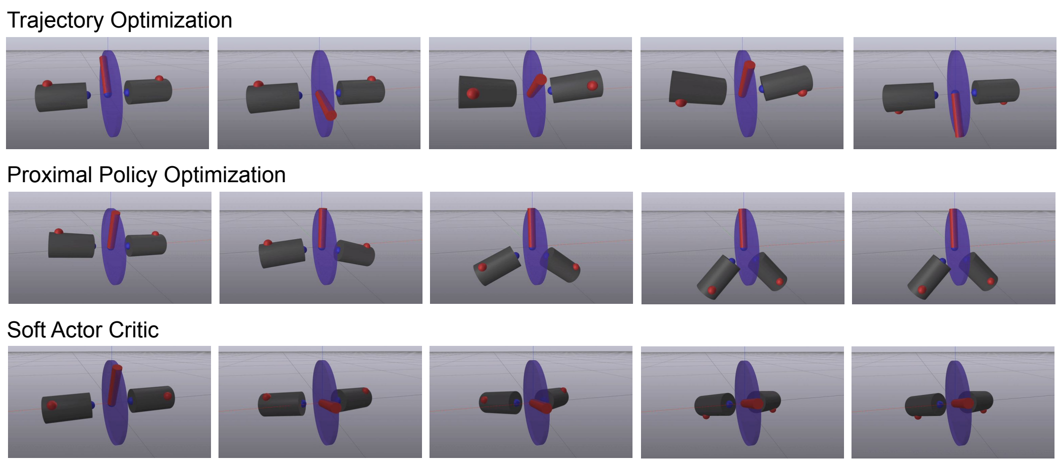

Figure 2: Trajectory Comparison – We compare the trajectories generated by trajectory optimization, PPO, and SAC. The inital state is shown on the left while the end state is shown on the right. The red dots on each cylinder represent the legs of our robot. Note that both trajectory optimization and SAC reach the goal state of with legs down. In contrast, PPO gets stuck with the legs pointing sideways.

two million steps, this instance of SAC was able to reach the goal from the fully supine position. Additionally, SAC reached the goal state from many other initialisations, including those within $2\pi/3$. We also noticed that the policy was able to repeat specific motions, sometimes hinging multiple times before reaching the goal state. In general, we found that the trajectories produced by SAC most closely matched those produced by trajectory optimization and those taken by real cats. See Fig. 2 for additional details.

Qualitatively, we noticed that SAC produced solutions with higher movement velocities and randomness in these movements. We suspect that this is the result of our sparse reward not penalizing movement cost, an issue that could be mitigated by introducing a movement cost term.

We had hoped to test this procedure with PPO as well, however we were unable to do so due to time and resources constraints.

4.4 Sparse vs Dense Rewards

Referring to Fig. 3, we can see that dense rewards struggled to surmount local minima, regardless of the algorithm. As above after two million training steps, PPO and SAC had average rewards of -500 and -450, respectively, indicative of this failure mode.

Seeing as PPO had a lower average reward, we opted to fine tune SAC. using this pretrained model, we switched to a sparse reward. After an additional two million steps, our SAC implementation had an average reward of 1000, which was continually increasing, with a positive trend by the end of the two million additional training steps.

5 Unexpected Issues

We had to build our own cat model from scratch and there exist several different ways to model a cat [2]. Determining a "correct" model or even determining the main factor responsible for the cat righting reflex is difficult to pinpoint. For the sake of training time and limiting the state and action space, we had to determine if the legs and claws [2], tail [12], or even head were important to the righting reflex. This thus confused us as to the appropriate state space for this problem. Eventually we settled on the two-cylinder model, one of the most popular models to date [6] with modifications [4] as described in section 3.1.

While designing our RL algorithms, we had lots of issues developing rewards that led to successful trajectories from supine positions. From class, we recognized the trend that dense rewards tended to improve policy learning, but we tried thirteen iterations before finding a reward function which lead

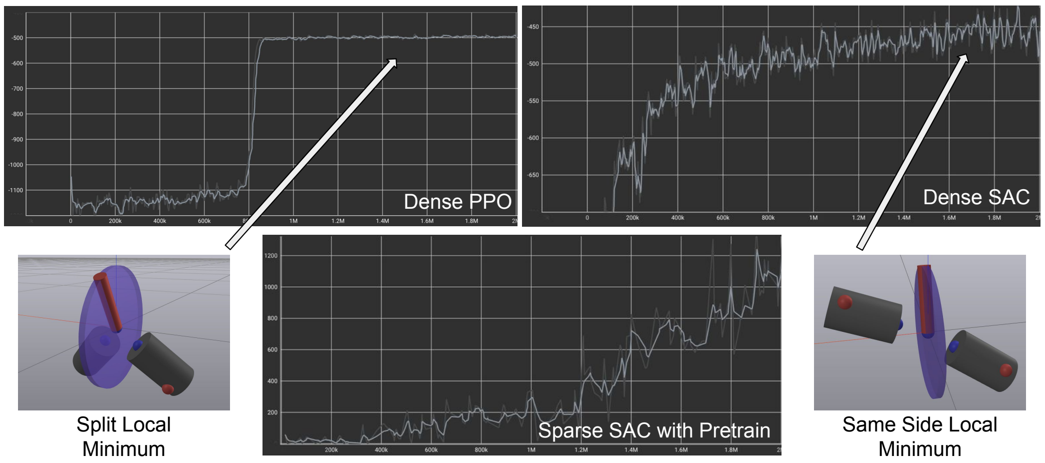

Figure 3: Reward Graphs – We compare the reward values for PPO with a dense reward function, SAC with a dense reward function and SAC pretrained with a dense reward and subsequently trained with a sparse reward. Steps are show on the x axis while reward values are shown on the y axis. Failure cases of both dense rewards are shown.

to positive rewards consistently. In fact, contrary to our intuition, we found that using a sparse reward function *after* training with a dense reward function led to more successful trajectories than dense rewards alone.

6 Conclusion/Discussion

6.1 Policy Limitations

Our trajectory optimization problem took about two minutes to solve when it could find a solution, much too long for physical robots to reorient themselves in midair before hitting the ground. Additionally, a failure to solve for a trajectory means that the robot would not even try to move, hence the need for pre-prepared policies.

After training PPO with a dense reward for two million steps, we noticed that it had reached a local minimum, as shown in Fig. 3. In contrast, when we trained SAC with a sparse reward (after pre-training for 1.6 million steps with dense reward), we observed a steady increase in reward over the two million steps, eventually reaching around 1000 by the end of training.

While PPO struggled to find successful trajectories, SAC did produce some successful trajectories, although they were often not smooth and far from our ideal. Despite this progress, none of our policies were able to land from a supine position within half a second. Although the "ideal" trajectory took two seconds to execute, the policies generated by PPO and SAC were quick to produce, but the executions were limited by the gains of the PID controller.

6.2 Dense vs Sparse Rewards

Our difficulties in finding a successful dense reward function lead us to use a sparse reward, which seemed counter intuitive. Dense rewards are common within the realm of reinforcement learning, but we found that, relying solely on dense rewards would lead to local minima.

The reorientation trajectory is non-holonomic [3], which suggests that successful trajectories will have to move to states further away from the goal state before being able to move to the goal state. If this is indeed the case, then dense rewards would be prone to falling into local minima. Dense rewards in both PPO and SAC being unable to find successful trajectories is therefore unsurprising because despite employing exploration bonuses, not only are the state (size 10, continuous) and action (size 4, continuous) space of our robot large, but the dynamics are much more complicated than the 2D examples we have explored in class. Thus, successfully exploring enough to exit from the local minima may be very hard.

In fact, Rudin, et al. worked on a similar zero-gravity reorientation problem, and trained their PPO algorithm for over 150 million steps [10] with a complicated dense reward function (7 terms). While dense rewards can possibly find success, we found that sparse rewards were quicker to implement and train after pretraining on a dense reward and gave us fairly nice trajectories. It suggests that sparse rewards may be particularly useful for non-holonomic trajectory problems. This must be balanced by the fact that sparse rewards suffer from data inefficiency and still need to get "lucky" to receive any reward at all, thus requiring a sort of "jumpstart" from the dense reward pre-training. It's also interesting to see that the average reward for the SAC model with dense rewards plateaued at around one million training steps; the transition to sparse occurred over 600k steps later, suggesting that our policy would have behaved similarly if we had pretrained on fewer steps.

6.3 Conclusion

We explored the use of trajectory optimization and reinforcement learning to solve the "falling cat problem" in robotics. We found that using a sparse reward function with SAC, pre-trained with dense rewards, led to more successful trajectories than using PPO with dense rewards. However, none of our policies were able to land from a supine position within half a second. Our results suggest that this is a promising direction for reorientation of mobile robotics, however more work is needed to develop policies that can effectively solve this problem.

7 Contributions

Model design + lit review- Kerlina & Ethan
PPO + SAC implementation -Ethan
Reward design - Ethan & Kerlina
Traj. Opt - Kerlina
conclusion/discussion - Kerlina & Ethan

References

[1] Greg Brockman et al. "OpenAI Gym". In: *CoRR* abs/1606.01540 (2016). arXiv: `1606.01540`.

[2] Jian Cao et al. "A Review of Research on Falling Cat Phenomenon and Development of Bio-Falling Cat Robot". In: *2022 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*. 2022, pp. 160–165.

[3] Xin-Sheng Ge, Wei-Jia Zhao, and Yan-Zhu Liu. "Nonholonomic Motion Planning for a Free-Falling Cat Using Quasi-Newton Method". In: 31 (2019), 42–49.

[4] Toshihiro Iwai and Hiroki Matsunaka. "The falling cat as a port-controlled Hamiltonian system". In: *Journal of Geometry and Physics* 62.2 (2012), pp. 279–291.

[5] Se Hwan Jeon, Sangbae Kim, and Donghyun Kim. "Real-time Optimal Landing Control of the MIT Mini Cheetah". In: *CoRR* abs/2110.02799 (2021). arXiv: `2110.02799`.

[6] T.R. Kane and M.P. Scher. "A dynamical explanation of the falling cat phenomenon". In: *International Journal of Solids and Structures* 5.7 (1969), pp. 663–670.

[7] Benjamin Katz, Jared Di Carlo, and Sangbae Kim. "Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control". In: *2019 International Conference on Robotics and Automation (ICRA)* (2019), pp. 6295–6301.

[8] Vince Kurtz et al. "Mini Cheetah, the Falling Cat: A Case Study in Machine Learning and Trajectory Optimization for Robot Acrobatics". In: *CoRR* abs/2109.04424 (2021). arXiv: `2109.04424`.

[9] Xingcan Liang et al. "Research on trajectory planning of a robot inspired by free-falling cat based on numerical approximation". In: *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2016, pp. 631–636.

[10] Nikita Rudin et al. "Cat-like Jumping and Landing of Legged Robots in Low-gravity Using Deep Reinforcement Learning". In: *CoRR* abs/2106.09357 (2021). arXiv: `2106.09357`.

[11] *Spot® - The Agile Mobile Robot | Boston Dynamics*. URL: `https://www.bostondynamics.com/products/spot` (visited on 05/13/2023).

[12] Yunxi Tang et al. *Towards Safe Landing of Falling Quadruped Robots Using a 3-DoF Morphable Inertial Tail*. 2022. arXiv: `2209.15337 [cs.RO]`.

[13] Russ Tedrake and the Drake Development Team. *Drake: Model-based design and verification for robotics*. 2019.