2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Interruptive Language Control of Bipedal Locomotion

Ashish Malik, Stefan Lee, Alan Fern

Abstract— **We study the problem of natural language-based control of dynamic bipedal locomotion from the perspective of operational robustness and hardware safety. Existing work on natural language-based robot control has focused on episodic command execution for stable robot platforms, such as fixed-based manipulators in table-top scenarios. These scenarios feature non-overlapping phases of instruction and execution, with execution mishaps usually posing no threat to the robot safety. This allows for non-trivial failure rates to be acceptable. In contrast, our work involves indistinguishable instruction and execution stages for a dynamically unstable robot where execution failures can harm the robot. For example, interrupting a bipedal robot with a new instruction in certain states may cause it to fall. Our first contribution is to design and train a natural language-based controller for the bipedal robot Cassie that can take in new language commands at any time. Our second contribution is to introduce a protocol for evaluating the robustness to interruptions of such controllers and evaluating the learned controller in simulation under different interruption distributions. Our third contribution is to learn a detector for interruptions that are likely to lead to failure and to integrate that detector into a failure mitigation strategy. Overall, our results show that interruptions can lead to non-trivial failure rates for the original controller and that the proposed mitigation strategy can help to significantly reduce that rate.**

I. Introduction

Advances in robotic control and artificial intelligence have enabled legged robots to execute maneuvers necessary for operating in human environments such as climbing stairs, opening doors, and traversing challenging terrain. Language offers a natural means for humans to interact with robots. It lowers the barrier of entry for robot control and is infinitely configurable, which can allow precise control of robot behaviors. Recent advances in large language models (LLMs) have allowed the development of language-based controllers that extend beyond fixed or pre-defined expressions. Many such controllers have been proposed for robot navigation, table-top manipulation and human-robot interactions. These works show impressive performance on various simulation benchmarks and real-world tasks.

Prior works have been based on various major presuppositions, such as static stability of robot operations, environments with regulated noise, absence of background chatter, and episodic contexts featuring distinct phases for instruction and execution. In reality, guaranteeing statically stable operations is not always possible, particularly for highly dynamic legged locomotion. In addition, human habitats cannot be noise regulated and are also often riddled with language utterances, not all of which will pertain to the robot. Furthermore, bare episodic efficiency does not serve as the ultimate criterion of functional efficacy when it comes to continuous operations alongside humans where there are not distinct instruction and execution phases. These concerns become more important as robot control becomes more intuitive with natural language and accessible to the lay-person without any training in robot control. This work aims to highlight, evaluate, and mitigate the challenges stemming from these strong assumptions in the field of natural language controlled robotics.

Non-episodic real-world settings blur the line between task specification and execution. Consequently, the system might receive new commands while executing ongoing ones. In instances where these command interruptions are not optimally timed—particularly when the robot is prone to disruption by the new command—they can destabilize the robot and jeopardize its safety. This can happen for a variety of reasons during routine procedures: operators may request new behavior, amend previous commands, or make corrections to the current behavior. On the other hand, environmental noises coupled with errors by the speech recognition module may inadvertently input unintended commands to the system. Additionally, the robot may encounter hostile agents which can issue harmful commands based on the robot's current state to compromise its safety.

We underscore the importance of the aforementioned failure modes using Cassie as our test platform. Cassie is a human-sized bipedal robot that is capable of executing highly dynamic locomotion behaviors, such as running, hopping, skipping etc. We begin by developing a language-based hierarchical locomotion controller for Cassie. The controller incorporates a pre-trained language model and a pre-trained control policy capable of diverse sets of agile behaviors. Subsequently, we highlight the significance of command interruptions by quantifying their impact on the system. Next, we formulate and implement a learning based method to detect harmful command interruptions and show it's efficacy in improving the robustness of the system. The main contributions of this work can be summarised as:

- Designing and developing a natural language-based controller for highly dynamic locomotion for the bipedal robot Cassie.
- Introducing command interruptions as an important consideration for language controlled dynamic locomotion and quantify the extent of failures they induce.
- Developing a learning-based method to detect harmful command interruptions and using it bolsters the system's resilience against such disruptions.

*This work is supported by the NSF Grants 2321851 and IIS-1724360.

All authors are with Collaborative Robotics and Intelligent Systems Institute, Oregon State University, Corvallis, Oregon, 97331, USA. Email: {malikas, leestef, alan.fern}@oregonstate.edu

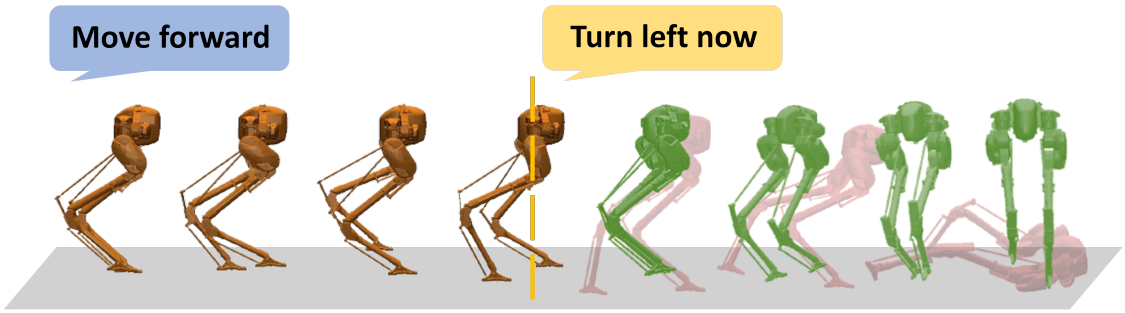


Fig. 1: Command interruptions are an important concern for natural language controlled dynamic locomotion. Interruptions with a new command at an unfavorable moment can destabilize the robot (red). The robot can take preventative measures if it can anticipate such an outcome, such as waiting until enacting the new command is safe (green).

II. Related Works

Language for robot control: Many recent works have leveraged LLMs for controlling robots using natural language. Most prior works [1–5] have exclusively focused on fixed-based manipulators. A few distinctions include [6–8], which work on language-guided navigation in human-environments. A few prior works have studied language-based control for legged robots such as [9], [10] and [11]. However, these works are limited to stable robot operations.

Non-episodic settings: Natural language-guided control is a well studied problem in robotics from a task-success point of view. However, most studies are assessed within an episodic context, usually free from operational disruptions and generally founded on the implicit presumption of an absence of poor vocabulary choices. These assumptions do not always hold during real-world deployment. A closely aligned work is [12], where online language corrections are given to the manipulator robot by modulating the state context for any given input. However, the testing scope of the system was limited to a single setting. Another notable work is [13] that trains a manipulator policy on a large set of human-annotated trajectories on a table-top environment to interactively and iteratively provide language guidance.

This work differs from prior works along two major dimensions. First, we develop a language-guided agent for a highly dynamic biped robot Cassie. No prior works have studied language-guided dynamic biped locomotion to the best of our knowledge. Second, we highlight the potential pitfalls and maladaptive behaviors that can arise when utilizing natural language to control dynamic robots, and we introduce strategies to counteract these problems.

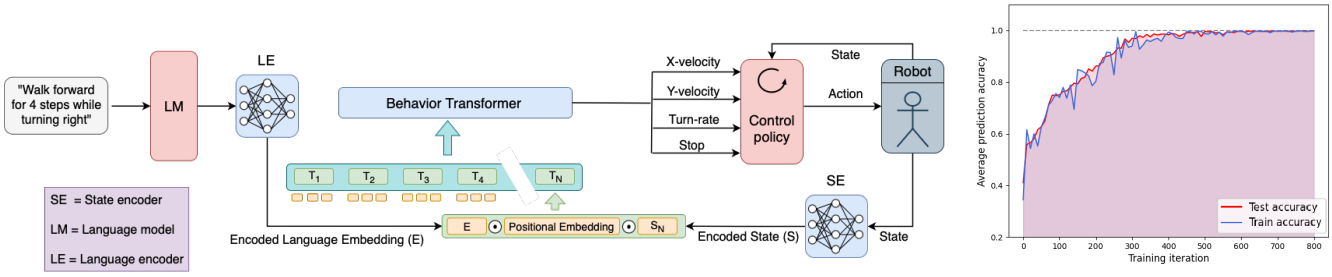
III. Language Controlled Dynamic Locomotion

In this section, we describe our locomotion control system for the biped Cassie that takes natural language locomotion commands at any time and ideally produces the desired behavior. This system can be adapted for any mobile robot by replacing the hardware specific components. It is built around a base locomotion controller from prior work [14], which we refer to as the *control policy*. The control policy is typically commanded via a hand-held remote control, which requires significant experience to operate safely and effectively. The key idea of our approach is to translate natural language into the appropriate sequence of remote control commands, which significantly lowers the skill and expertise required. Below, we first describe the scope of locomotion behaviors considered in this work, followed by an overview of the system architecture, training data generation and system training.

A. Scope of Locomotion Behaviors

To focus on the interruptive aspects of language control, we restrict this work to blind locomotion, where the only input to the control policy is the user command and proprioceptive information. Even in this setting, there are a large variety of possible locomotion behaviors that could be considered and we focus on a limited, but useful, subset in this study. In particular, we consider language control of walking, running, and turning with a fixed set of available velocities. Below we describe three classes of language commands over these behaviors that our system will support.

- *Primitive Commands*: These commands initiate one of 8 simple locomotion primitives that include walking, turning, running and stepping in place (stop). Walking can be commanded in the forward/backward directions with a fixed velocity of $\pm$ 0.8 m/s, or the left/right side-stepping directions with a fixed velocity of $\pm$ 0.25 m/s. Turning is constrained to $\pm$ 0.2 rad/s while running is constrained to a forward direction with velocity 2.0 m/s. Examples of the language commands for these locomotion primitives include “walk forward”, “start running”, “turn left”.
- *Constrained Primitive Commands*: These commands allow for specifying constraints on the execution time or number of footsteps for primitive commands. The time and the footstep constraints are integers in the range 1 to 6 (inclusive). Examples are “turn right for 4 seconds” and “walk backward for 5 steps”.
- *Composition Commands*: These commands allow different combinations of commands from either primitives or constrained primitives. The combinations can be either sequential or parallel (if compatible). Examples include “walk forward for 3 steps while turning left” (parallel) and “walk forward for 2 steps then walk backward for 3 steps” (sequential). Our architecture limits the maximum time duration that combinations can span as described later.



(a) (b)

Fig. 2: (a) Overall System Architecture. The models in red are pre-trained and always kept frozen. The models in blue are learned during training. (b) Training and test accuracy of the behavior transformer.

B. Architecture

Figure 2a shows the overall architecture of our system for language-based locomotion control. At a high-level, the natural language command is given to a pre-trained language model, which produces a language embedding vector. At each time step, this embedding is combined with the recent history of robot states to produce a tokenized sequence that is input to the behavior transformer. The transformer outputs the command to be passed to the pre-trained control policy at that timestep. The control policy directly controls the robot in a closed loop. The main components are described below.

Pre-trained Control Policy. Following [14], the input to the control policy is the current robot proprioceptive states (position/velocity of joints and pelvis velocities/orientation), a periodic clock signal, and locomotion commands specifying target x-velocity, y-velocity and turn-rate. The output is the joint level PD setpoints for each of the robot's 10 joints (5 per leg). The control policy updates the PD setpoints at 50Hz and the underlying PD controller runs at 2 kHz. The controller architecture is a GRU recurrent neural network and is trained in simulation with dynamics randomization using the PPO reinforcement learning [15]. This same controller architecture and training methodology was used in a variety of prior work that demonstrated sim-to-real transfer [16–19].

Pre-trained Language Model. The language model L, takes the natural language command I as input and outputs the language embedding $L(I)$. In all of our experiments we use the freely available *e5-mistral-7b-instruct* language model [20], which has 7.11B parameters and produces 4096 dimensional embeddings.

Language and State Encoders. The robot state and language embedding are processed by state encoder E_{state} and embedding encoder E_{emb} respectively. Let s, e_{emb}, and e_s be the robot state, encoded language embedding and encoded robot state such that $e_s = E_{state}(s)$ and $e_{emb} = E_{emb}(I_{emb})$. Both encoders are parameterized using fully connected networks with ReLU activations, one hidden layer of size 256 and have output size of 128.

Behavior Transformer. The input to the behavior transformer at each timestep t starting with the new command is a tokenized sequence of vectors $V_t = (v_1, v_2, \ldots, v_t)$ that encodes the current language command and history of the robot states since the latest command. Each token v_i is produced by concatenating the following information: 1) a 128-dimensional sinusoidal positional embedding p_i, 2) a learned 128-dimensional encoding e_i of the language embedding $L(I)$ produced by an MLP encoder, and 3) a learned 128-dimensional encoding u_i of the robot proprioceptive state at time i. Both the language and state encoders are 2-layer [256,128] MLPs with ReLU activations. The history sequence is reset each time a new command is received.

The behavior transformer is a 3-layered transformer model that uses 8 attention heads and 1024 feed-forward dimensions. The four output heads of the transformer output the velocity commands for the control policy (x-velocity, y-velocity, turn-rate) and a binary "stop" token $\hat{b}_t$ for every timestep t to indicate when the current command execution has finished. Upon such prediction, the command velocities to the control policy are changed to zero which we treat as the control system's default behavior. Due to computational capacity we limit the transformer context length w to 300, which corresponds to 6 seconds of real-world operations.

C. Training Data Generation and Training

Our system is trained on a dataset D of natural language-annotated demonstrations of locomotion behavior. Specifically, each data point in D is of the form (I, d), where I is the natural language command, and $d = (s_1, a_1, s_2, a_2, \ldots, s_T, a_T)$ where s_t and a_t are the robot-state and control policy commands respectively at time t. This dataset is collected in simulation with dynamics randomization. Note that, unlike recent work on language-based robot control [21] we do not have a previously collected set of such annotated demonstrations for our target robot platform. Rather than spending resources on large-scale annotation, we instead explore a strategy that involves automated annotation over a hand-crafted space of language instructions, relying on the language model for test-time generalization.

To generate language instruction I, we first handcraft an equal number of language commands (four) for each locomotion primitive. We then define templates for generating instructions for constrained primitives that can enumerate over different primitive commands, the type of constraint, and numeric values. Finally, for composition commands, we define templates that combine two language commands (primitive

or constrained) either sequentially (via "then") or in parallel (via "while") as appropriate for the selected commands.

Given a generated language instruction I, we created a parser that automatically generates the correct sequence of commands for the control policy, including the appropriate stopping conditions. To produce a demonstration d for I, we initialize the robot in the simulator using a randomized start state distribution and operate the robot using the control policy with randomized velocity commands for 2 seconds (100 simulation steps) to randomize the initial conditions for d. We then run the control policy with the command sequence from the parser and record the state and command sequence to produce d. Our full dataset is comprised of an equal number of such annotated demonstrations from each of the three command categories.

For training, we optimize the parameters of the behavior transformer B and the language and state encoders used to construct the tokenized sequence. The parameters of the pre-trained language model and control policy are held fixed. The collected data is divided into an 80:20 training/testing split. From our definition of the language command categories in this work, we can enumerate the possible target outputs of the transformer (i.e. control policy commands). Thus, we use cross-entropy loss to train the transformer. Note that, we have also experimented with the MSE loss and found little difference between the two. We use the Adam optimizer (learning rate 1e-4) and gradient clipping for training stability.

D. Prediction Evaluation

Figure 2b shows the evolution of training and test accuracy during training. These results show that the selected architecture and hyperparameters yield strong base performance on our automatically generated demonstration data.

We now consider the generalization capabilities of our system to language outside of the generated data. In particular, we used a publicly available language model API to generate a set of 100 new natural language commands corresponding to the primitive commands. These new language commands are not present in any of our automatically generated data. We test each new command in simulation by randomly initializing the robot before the new language command is given to the system. We then measured the average accuracy over 8 trails each of the 100 commands further averaged by using models trained with 5 random seeds. The overall accuracy was $86.68 \pm 0.04\%$ compared to the nearly perfect accuracy for language commands in the original training and testing data. A clear direction to improvement is to train on an extended dataset including these and other automatically generated language commands. However, we leave that to future work given the focus of this work—interruptive performance of language-controlled dynamic systems.

IV. Evaluating Stability Under Interruptions

A language controlled robot can experience command interruptions for a multitude of reasons. The operator may intentionally interject to correct or modify the current behavior, or the robot may unintentionally interpret environment sounds as commands. Below, we introduce our methodology for evaluating the system's stability under different types of interruption patterns. We then evaluate our learned system to show that interruptions do cause non-trivial failure rates.

Evaluation Methodology. We evaluate the stability under interruptions in simulation by interrupting our system with new commands at varying frequencies and number of interruptions. Specifically, each evaluation episode involves an interruption frequency between 0.4 to 2.0 interruptions per second and a total number of interruptions from 1 to 4 to capture different interruption patterns. Each episode begins by initializing the robot in simulation using a randomized start state distribution. We provide a random language command to the robot and let it run uninterrupted for 2 seconds of real world time. Afterwards, we interrupt the system at the selected interruption frequency for the selected number of interruptions using new commands. These new commands can be sampled from the following two distributions of commands:

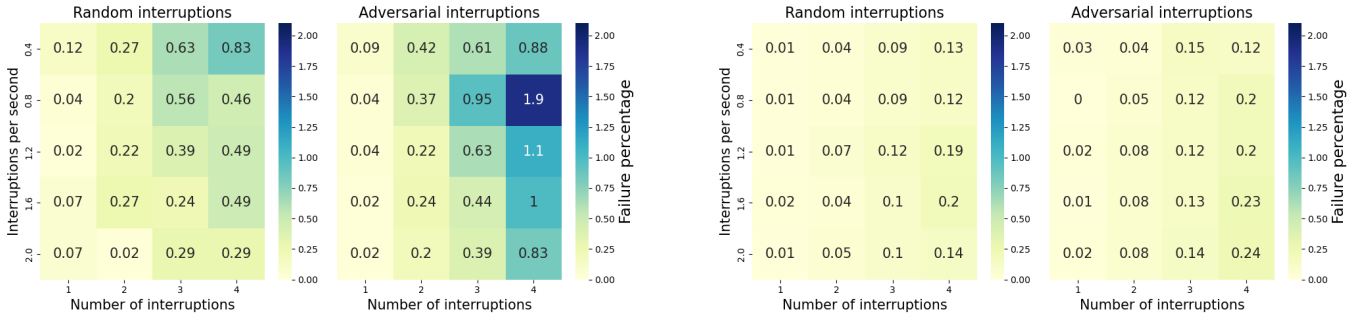
- **Random interruptions**: The interruption commands are sampled uniformly from the set of all commands, from all three command categories, in the training data.
- **Adversarial interruptions**: For every interruption, with a probability of 0.5, we generate a random interruption as described above. Otherwise, we sample a language command from a hand-designed adversarial distribution that aims to maximally conflict with the currently executing command. For example, if the current command is "Walk backward while turning right", the adversarial command might be "Run forward while turning left". This distribution is defined via a deterministic function over the adversarial behavior, which is used to sample a randomized language command for that behavior.

An episode ends as a failed episode when the robot falls down, otherwise the episode ends as a success 3 seconds after the last interruption.

Evaluation Results. We conducted evaluation trials for our learned language controller for all combinations of interruption frequencies (interruptions per second) $f \in \{0.4, 0.8, 1.2, 1.6, 2.0\}$ and number of interruptions $n \in \{1, 2, 3, 4\}$. For each combination we evaluated the failure percentage over 2048 simulated episodes.

Figure 3a quantifies the results for both random and adversarial interruptions. Overall, the system usually achieves a small failure percentage of less than 1%. Adversarial interruptions generally lead to more failures than random interruptions. We see that the failure percentages tend to be significantly smaller for $n = 1$. This indicates that successive interruptions tend to compound stability issues. Indeed, we see that in general the failure percentage increases as n continues to increase. We also note a weaker trend of the failure percentage decreasing as f increases. While this may appear counter-intuitive, a likely explanation is that for higher frequencies an interruption can occur before the controller can produce a significant response to the previous command.

While these observed failure rates appear small in magnitude, it is important to note that even these small rates can be



(a) Failure percentage without mitigation

(b) Failure percentage with mitigation

Fig. 3: Failure percentage (0%-100%) for each combination of interruptions per second and number of interruptions for both random and adversarial interruptions. (a) Unmitigated command interruptions and (b) Mitigated command interruptions.

unacceptable for real-world settings involving long periods of robot operation. This motivates investigating approaches for mitigating these failures. It's essential to highlight that due to the small magnitude of failure rates, reliably and consistently replicating these failures on the real robot is challenging. As a result, we restrict our interruption experiments to the simulated environment.

V. Predicting and Mitigating Interruptions

In this section, we describe a learning-based approach to predict whether an interruption will lead to a failure. We then develop and evaluate a corresponding mitigation approach.

Predicting Unstable Interruptions. We developed two different datasets to train our classifier that identifies interruptions potentially leading to failures, referred to as unstable interruptions. Each example in the dataset is a tuple (s, I, l), where s is a robot state, I is the language instruction (i.e. interruption) given at that state, and l is binary label indicating whether the interruption is unstable or stable. We generate two different datasets that differ in whether they are based on random or adversarial interruptions. An example is generated by first simulating either a random or an adversarial interruption episode as described in Section IV for randomly selected interruption frequencies and counts. If the episode is a failure with I being the instruction issued at time t before the failure, then we create an example $(s_{t'}, I, \text{unstable})$ for all $t' \geq t$ to the end of the episode. If the episode is not a failure, then, for each time t with active interruption instruction I we generate an example (s_t, I, stable). Each dataset (Random and Adversarial) contain 145k examples. The rarity of failures leads to a severe class imbalance, therefore, we downsample the stable examples to yield a balanced class distribution.

The classifier architecture consists of three separate MLPs. We first generate the language embedding $L(I)$ from the language command I using the pretrained language model L. This embedding is then processed by a single hidden layer MLP of size [512,512]. The robot states are separately processed by another single hidden layer MLP of the same size. The output of both MLPs is then concatenated and fed into another MLP with 2 hidden layers of size [512,32,1] to output the probability of an unstable interruption. The classifier is trained end-to-end using cross entropy loss with an 80:20 train/test split.

We measured the area under the ROC (AUROC) curve for each of the classifiers on each of the test datasets. The Adversarial classifier achieves an AUROC of 0.995 on the Adversarial data and 0.996 on the Random data. The Random classifier achieved an AUROC of 0.966 on the Adversarial data and 0.971 on the Random data. The near perfect AUROC values, particularly for the Adversarial classifier indicate that unstable interruptions have strong distinguishing characteristics that the classifiers can capture. Intuitively, we might expect Adversarial examples to be easier to classify and not capture the general types of failures that might occur. Thus, it is interesting that the Adversarial classifier, which was trained on only the Adversarial data, outperforms the Random classifier on the Random data. Understanding this observation is a potential line of future work.

Mitigating Command Interruptions. We incorporate the unstable-interruption classifier learned on the Adversarial dataset into our system to enhance the resilience to unstable-interruptions. We evaluate the stability of our system with the integrated classifier by modifying our stability evaluation methodology as follows: For any new command interruption, we use the current state of the robot (s) and the language embedding ($L(I)$) of the incoming interruption command I to classify the interruption as stable or unstable. If the classifier predicts stable interruption, the new command is promptly executed by the system system. Conversely, if the classifier identifies the interruption as unstable, the system continues executing the preceding command. For unstable prediction, we reassess the classifier prediction at every time-step with the current robot state until safe interruption is predicted. If safe interruption is never predicted before we receive another interruption command, then we say the interruption was ignored. This approach can lead to some delay in executing commands, which we refer to as command delay.

By analyzing the classifier's ROC curve, we selected a probability threshold for detecting an unstable interruption so that the true-positive rate was greater than 99%, which yields a false-positive rate of less than 3%. This prioritizes robot safety over delaying or ignoring instructions due to

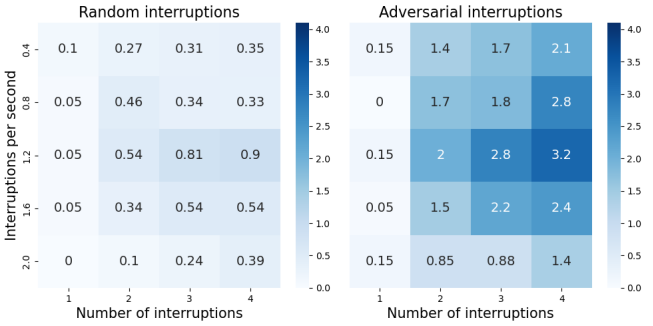


Fig. 4: Percentage of ignored commands (0%-100%).

false positives. Figure 3b shows the stability to interruptions with the integrated interruptions classifier, which reduced the number of failures for Random interruptions and Adversarial interruptions by 73.5% and 80.1% on average respectively. Similar to the previous evaluation of unmitigated interruptions, we observe that adversarial interruptions tend to lead to more failures than random interruptions. We also note that the failure percentage generally increase with the number of successive interruptions. Overall, the new system usually achieves a failure percentage of less than 0.2%.

The percentages of ignored interruption commands for various numbers of interruptions and their frequencies are shown in Figure 4. There is a higher percentage of ignored commands for adversarial interruptions. Overall, the percentages of ignored commands remains less than 1% for random interruptions and less than 3% for adversarial interruptions. The average command delay for random and adversarial interruptions across all frequencies and numbers of interruptions was 0.066 and 0.41 time-steps respectively. We conclude that the integrated classifier significantly reduces the failure rates across all frequencies and numbers of interruptions, and therefore boosts our system's robustness to command interruptions. The classifier inadvertently also trades off system's responsiveness by adding some delay to the interruption response and occasionally failing to enact some command interruptions. However, our experiments indicate that the percentage of failed interruptions and interruption-delay are quite negligible.

VI. Summary

We developed a natural language controller for the highly dynamic robot Cassie and focused on the challenge posed by command interruptions. Our results highlight the importance of considering command interruptions in the design, implementation, and evaluation of language-controlled locomotion systems. Additionally, we introduced a learning-based command interruptions mitigation method, which significantly lowered the incidence failures with minimal impact on the system's responsiveness. Potential future work includes extending the language command space, improving the generalization capabilities of the system, and exploring more sophisticated methods for detecting and mitigating command interruptions as well as hardware experiments.

References

[1] P.-L. Guhur, S. Chen, R. G. Pinel, M. Tapaswi, I. Laptev, and C. Schmid, "Instruction-driven history-aware policies for robotic manipulations," in *Conference on Robot Learning*. PMLR, 2023, pp. 175–187.

[2] K. Mo, Y. Deng, C. Xia, and X. Wang, "Learning language-conditioned deformable object manipulation with graph dynamics," *arXiv preprint arXiv:2303.01310*, 2023.

[3] J. Yang, W. Tan, C. Jin, B. Liu, J. Fu, R. Song, and L. Wang, "Pave the way to grasp anything: Transferring foundation models for universal pick-place robots," *arXiv preprint arXiv:2306.05716*, 2023.

[4] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2023, pp. 785–799.

[5] S. Nair, E. Mitchell, K. Chen, S. Savarese, C. Finn *et al.*, "Learning language-conditioned robot behavior from offline data and crowd-sourced annotation," in *Conference on Robot Learning*. PMLR, 2022, pp. 1303–1315.

[6] J. Krantz and S. Lee, "Sim-2-sim transfer for vision-and-language navigation in continuous environments," in *European Conference on Computer Vision*. Springer, 2022, pp. 588–603.

[7] C. Huang, O. Mees, A. Zeng, and W. Burgard, "Visual language maps for robot navigation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 10 608–10 615.

[8] J. Krantz, T. Gervet, K. Yadav, A. Wang, C. Paxton, R. Mottaghi, D. Batra, J. Malik, S. Lee, and D. S. Chaplot, "Navigating to objects specified by images," *arXiv preprint arXiv:2304.01192*, 2023.

[9] A. Bucker, L. Figueredo, S. Haddadin, A. Kapoor, S. Ma, S. Vemprala, and R. Bonatti, "Latte: Language trajectory transformer," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 7287–7294.

[10] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik *et al.*, "Language to rewards for robotic skill synthesis," *arXiv preprint arXiv:2306.08647*, 2023.

[11] Y. Guo, Y.-J. Wang, L. Zha, Z. Jiang, and J. Chen, "Doremi: Grounding language model by detecting and recovering from plan-execution misalignment," *arXiv preprint arXiv:2307.00329*, 2023.

[12] Y. Cui, S. Karamcheti, R. Palleti, N. Shivakumar, P. Liang, and D. Sadigh, "No, to the right: Online language corrections for robotic manipulation via shared autonomy," in *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, 2023, pp. 93–101.

[13] C. Lynch, A. Wahid, J. Tompson, T. Ding, J. Betker, R. Baruch, T. Armstrong, and P. Florence, "Interactive language: Talking to robots in real time," *IEEE Robotics and Automation Letters*, 2023.

[14] J. Siekmann, Y. Godse, A. Fern, and J. Hurst, "Sim-to-real learning of all common bipedal gaits via periodic reward composition," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7309–7315.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[16] H. Duan, A. Malik, M. S. Gadde, J. Dao, A. Fern, and J. Hurst, "Learning dynamic bipedal walking across stepping stones," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 6746–6752.

[17] D. Crowley, J. Dao, H. Duan, K. Green, J. Hurst, and A. Fern, "Optimizing bipedal locomotion for the 100m dash with comparison to human running," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 12 205–12 211.

[18] H. Duan, A. Malik, J. Dao, A. Saxena, K. Green, J. Siekmann, A. Fern, and J. Hurst, "Sim-to-real learning of footstep-constrained bipedal dynamic walking," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 428–10 434.

[19] J. Dao, K. Green, H. Duan, A. Fern, and J. Hurst, "Sim-to-real learning for bipedal locomotion under unsensed dynamic loads," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 449–10 455.

[20] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei, "Improving text embeddings with large language models," *arXiv preprint arXiv:2401.00368*, 2023.

[21] A. Padalkar, A. Pooley, A. Jain, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Singh, A. Brohan *et al.*, "Open x-embodiment: Robotic learning datasets and rt-x models," *arXiv preprint arXiv:2310.08864*, 2023.