Reinforcement Learning-based Modeling and Runtime Repair of Safe Stand-up Routine for a Bipedal Robot

GHOSH Suman, KHANNA Parag
Supervisor: Prof. Armando Tacchella

European Master on Advanced Robotics+ (EMARO+)
Dipartimento di Informatica, Bioingegneria, Robotica, Ingegneria Dei Sistemi (DIBRIS)

Università Degli Studi Di Genova, August 2018

Abstract*—Standing up is a critical recovery task for bipedal locomotion due to high occurrence of falls in legged humanoid robots. A reliable stand-up strategy is required to ensure that the robot avoids certain unsafe states (collision, fall etc.) that could cause critical failure. We use epsilon-greedy Q-Learning around a state space formed using a scripted action sequence, to model the stand-up routine as a parametric Discrete Time Markov Chain. We subsequently use Probabilistic Model Checking and repair the learned model greedily to minimize reachability of unsafe states. We focus on monitoring the robot during runtime to catch discrepancies between the learned and real model, caused by sudden environmental changes. We test the framework extensively by introducing synthetic faults in the robot model that were absent during initial modeling. Results illustrate that our framework can successfully repair learned strategies when the faults introduced are not catastrophic.

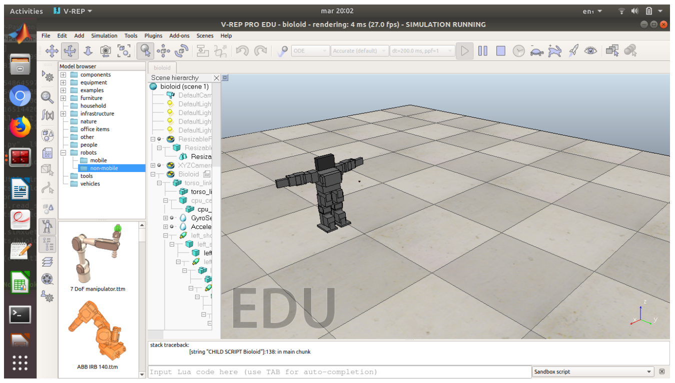
I. Introduction

Bipedal locomotion is a challenging task for a humanoid robot. It is essential for the overall robustness to have reliable procedures to get the robot back into an upright posture from a fallen state. Standing-up requires that the robot's center of mass (COM) projection to the ground leaves the convex hull spanned by the feet contact points. This results in whole-body motions with sequences of support points at knees, shoulders, palms etc [1]. The many degrees of freedom of humanoid robots and the changing contact points make it difficult to apply conventional motion-planning techniques. This approach further lacks flexibility as the routines are scripted as predetermined command sequences which are fed to the motors in an open-loop. Sudden changes in the environment or in the robot may violate the underlying assumptions causing the routine to fail. Proposed solutions in existing literature (e.g., [2], [3], [4]) are limited as they do not consider a full humanoid for case study.

In this work, we investigate, validate and extend the approach used in [5] to improve the stability of scripted standing-up strategies using reinforcement learning. The robot learns how to get up on its feet by interacting with the environment, observing the effects of its actions, and trying to come up with a policy that maximizes its probability to reach the desired goal state. However, reinforcement learning cannot guarantee that the robot will avoid unsafe states (critical joint values, unstable poses, collisions etc.) with a given probability. Static repair strategies based on probabilistic model checking can be used to modify the robot's learned policy to make it follow a safer path. However, this safety property does not necessarily transfer to the real system with all its uncertainties. We thus require runtime monitoring to update the learned model into a more realistic representation. Policy repair can then be iteratively applied to generate safer action sequences coherent with the updated model.

We introduce synthetic faults in the robot model during runtime to simulate discrepancies between training and runtime environment. We subsequently test the runtime monitoring loop to investigate cases where it succeeds and fails. We then analyze the results to hypothesize why runtime model and policy repair fails in case of catastrophic faults.

The article structure is as follows: Section I contextualizes and introduces our work. Section II gives a technical description of the robot and how we model it. Section III presents an overview of the algorithm framework employed, while briefly describing each component involved. We describe our experimental setup and analyze the results in Section IV and V respectively. Finally, Section VI concludes proceedings and proposes future improvements.



```
stack traceback:
        [string "CHILD SCRIPT Bioloid"]:138: in main chunk
Input Lua code here (use TAB for auto-completion)
```

Figure 1: Bioloid simulated in V-REP

II. Robot Description

We use the humanoid robot Bioloid by ROBOTIS [6]. It has 18 degrees of freedom (DOF) and can stand up without exploiting dynamics from both prone and supine posture due to its powerful Dynamixel AX-12A actuators [7]. The original robot is illustrated in figure **??** However, in our work, we use a simulated model of such a robot [8]. We use the robot simulator V-REP [9], which allows the creation of fully customizable simulations that can be controlled through an API. It is illustrated in figure 1. Our experiments are carried out using the Open Dynamics Engine (http://www.ode.org).

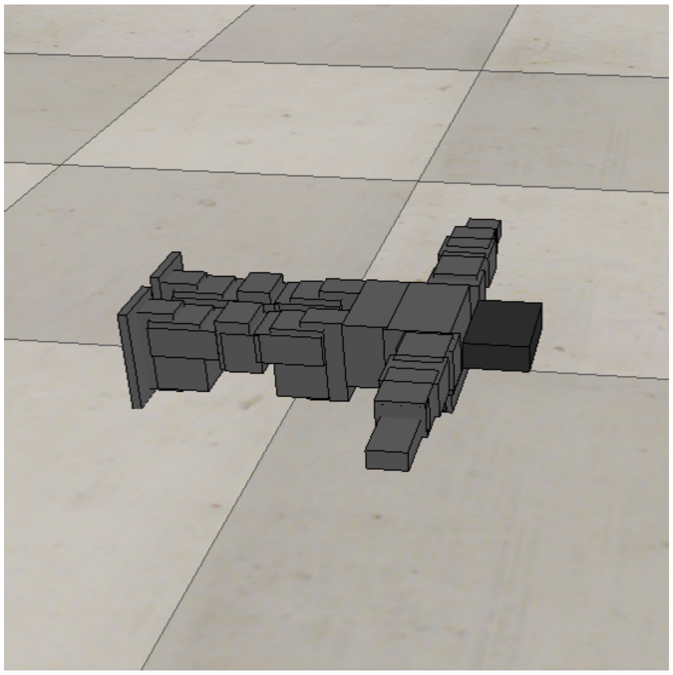


Figure 2: Robot Initial State

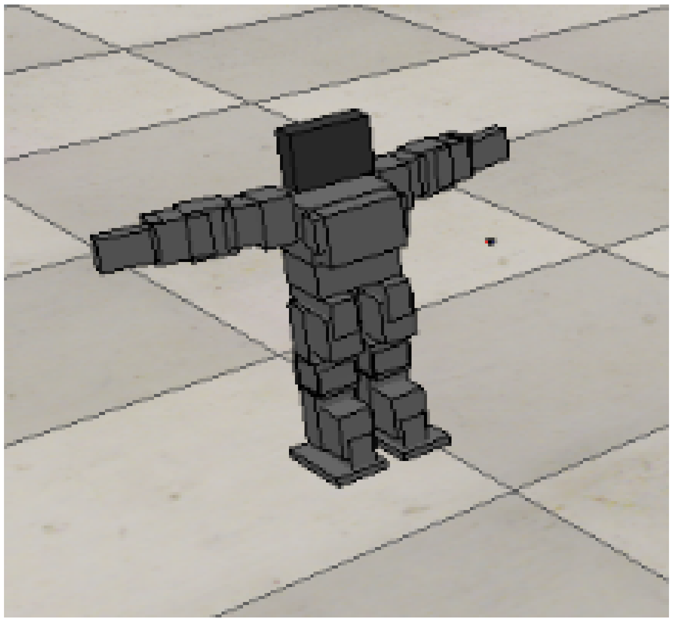


Figure 3: Robot Goal State

III. Algorithm Overview

This section describes the overview of the entire framework, and briefly describes each component involved. The overall flow of the framework is illustrated in figure 4.

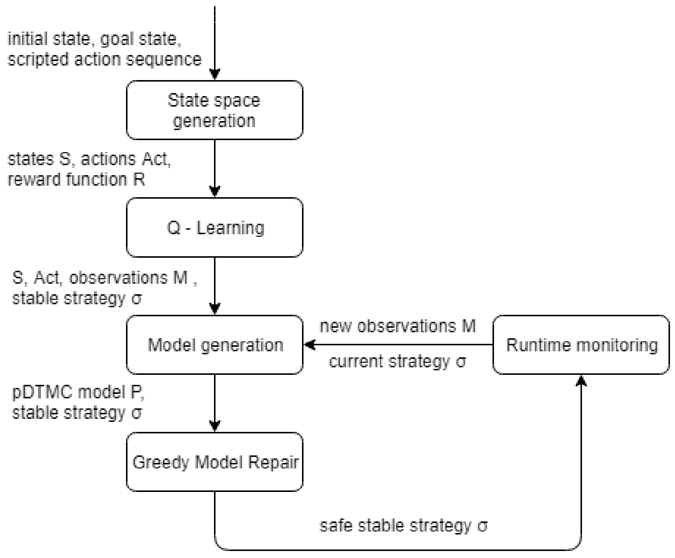

initial state, goal state, scripted action sequence
State space generation
states S, actions Act, reward function R
Q - Learning
S, Act, observations M , stable strategy σ
Model generation
new observations M
current strategy σ
Runtime monitoring
pDTMC model P, stable strategy σ
Greedy Model Repair
safe stable strategy σ

Figure 4: Framework flowchart

A. *State-Action Space Generation*

The initial state s^{init} for the robot is the supine posture, and the goal state s^{goal} is defined as the standing up posture, both with outstretched arms, as depicted in figures 2 and 3. The unsafe states are defined as s^{coll} (corresponding to self-collision), s^{fall} (corresponding to ground collision) and s^{far} (corresponding uncertain states far from our defined state-space). Since the state-action space is intractably large, we start by employing techniques used in [1] to obtain a sequence of scripted actions $A = \{a_0, a_1, ..., a_k\}$ that guide the robot from its initial to final state. We then build the state space a pipe around this action sequence, thus exploring only nearby states. We assume that scripted sequence is fairly "good", and alternative safe strategies for standing up can be found in its vicinity. The implementation details of encoding the state and action space has been discussed in Section IV. This module also encodes the standing up task with a reward function that is subsequently used for Q-Learning.

B. *Q-Learning*

We use epsilon-greedy Q-Learning for exploring various states during the training phase, and populate the state-action pair entries in the Q-table accordingly using the well-known Bellman Equation. During learning, we restrict our state-action space as described in the previous module to ensure fast convergence of Q values. These Q-values are used to generate a policy σ that tells the robot which action to take (using a stochastic or deterministic strategy) during a particular current state. During learning, we also populate our initial Observation Matrix $M(s, a, s')$ that will be used to generate the probability transition model as described in the next module. It maps the frequency of occurrence of action a after state s leading to the next state s'.

C. *Markov Model Generation*

For the standing up routine, we must provide not only stability, but also safety. Q-learning relies on an estimation obtained implicitly during the learning process, whereas verification requires an explicit model. In order to enable formal methods to check the safety properties of our model, we need a parametric Discrete Time Markov Chain (DTMC) model of the robot in its environment. We generate the DTMC model using the learned policy values σ and the observation matrix M.

D. *Model Checking and Repair*

We used Storm [10] for probabilistic model checking. From the DTMC model, we basically compute the probability to reach any unsafe state from an initial state. We strive to put an upper bound λ on this probability. Hence, we need to repair the model if the above bound is violated.

We employ a greedy model repair approach as in [11], wherein we select for each state s, the safest action a_{safe} (, i.e., the action which has minimum probability of leading the robot into an unsafe state), and increase its corresponding policy value $\sigma(s, a_{safe})$ by δ while decrementing σ values for all other actions corresponding to s by δ. To speed up the repairing process, we first select a relatively large δ and decrement it when no further repair is possible (and the safety threshold is not yet reached).

E. Runtime Monitoring

This module is critical to lessen the gap between the underlying model assumed while performing reinforcement learning, and the real world. Here, we continuously monitor the runtime performance of our learned and repaired policy. Whenever an unexpected change in the environment or robot occurs, the observation matrix M will not be consistent with DTMC model that we had built initially. This will cause the unsafe probabilities computed using Model Checker to violate the safety bounds significantly. This implies that inconsistency between the learned and real model is causing the robot to reach unsafe states. In such a situation, we should consequently repair the model. Hence, the feedback loop of runtime monitoring accumulates the input in the form of M and updates σ, each time performing model generation, checking and repair. Each runtime monitoring feedback iteration executes after a certain number of episodes of the stand-up routine. We continuously iterate and repair until safety bounds are respected or the reduction in unsafe probability is below a threshold ϵ.

IV. Experimental Setting

Now, we shall describe the implementation details of the framework that we used for experimenting. Each state of the robot system is encoded as a vector $s = \{x, y, z, q_0, q_1, q_2, q_3, \rho_1, ..., \rho_{18}\}$, where (x, y, z) are the COM coordinates; (q_0, q_1, q_2, q_3) are the quaternions denoting the orientation of the torso according to the absolute coordinate system of the simulator; $(\rho_1, ..., \rho_{18})$ are the 18 joint angles corresponding to the 18 DOFs of the robot. To reduce the action, we coupled joints of the left and right limbs and controlled the actuators of only 6 joints: elbow, shoulder lateral, shoulder swing, hip swing, knee and ankle swing. Thus, each action is represented by a 6D vector where each element can be any one of $\{-1, 0, 1\}$, where "1" denotes 30 degree increment in joint angle. Skip action and reset actions were also included in the action space.

During state-action space generation, for the reward function, we use the constants for goal = 1000, for fall = -100, for collision = -100, for far states = -100 and for normal execution = 1. In our experiments, we used $\lambda = 0.001$, $\delta = \{0.2, 0.1, 0.05, 0.01\}$ in decreasing order, and $\epsilon = 0.000001$.

We had to update the existing code-base (written 2 years ago), particularly the model checker based on Storm (which was then in its beta form). The latest version of Storm is now public with Stormpy as its set of python bindings. Various installation scripts for the whole environment were also needed to be corrected to get the Learning, Model Checking and the Run-Time Monitoring running. Due to lack of time and computational resources needed to perform Q-learning on this massive state-action space (17614 X 730), we used pre-trained Q-tables for testing the runtime monitoring loop. The updated code-base is available at : https://github.com/ghoshsuman/bioloid-standup. We were able to verify in simulation that the policy and model acquired from pre-trained Q-tables lead to successful standing up task of the robot.

V. Results and Discussion

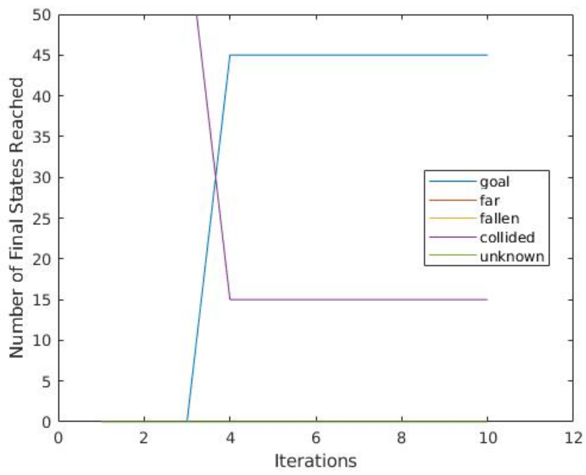


Figure 5: Number of terminating states reached with faulty action 579

Here, we give an introduction to the entire section, and give an overview of the different cases that we use for analyzing the effectiveness of runtime monitoring. For each case, we

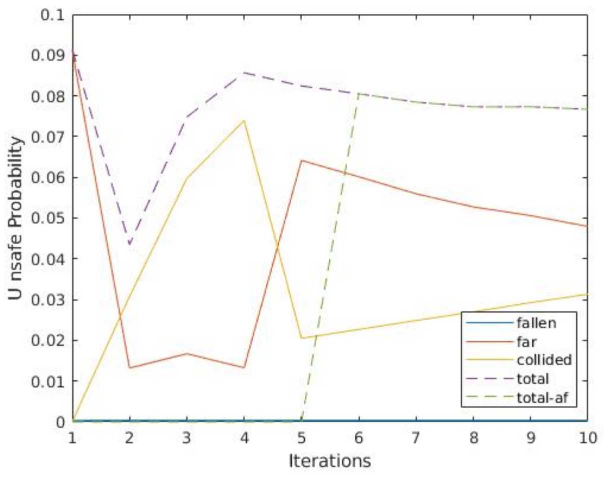


Figure 6: Variation of unsafe probabilities for faulty action 579

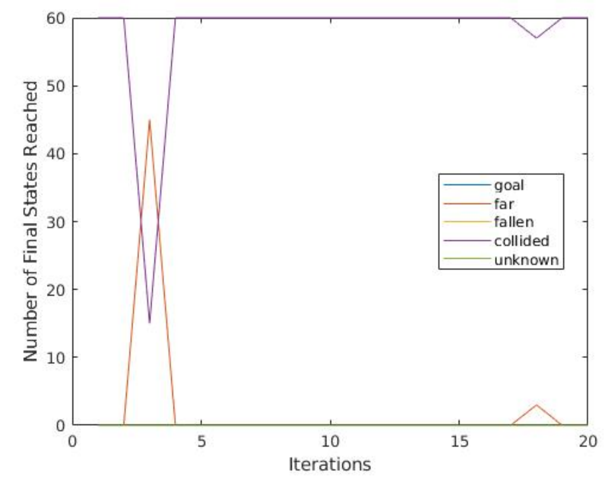


Figure 7: Number of terminating states reached with faulty actions 579 and 337

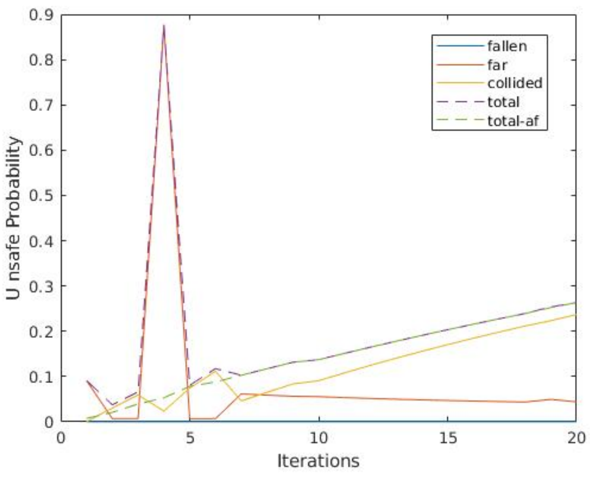


Figure 8: Variation of unsafe probabilities for faulty actions 579 and 337

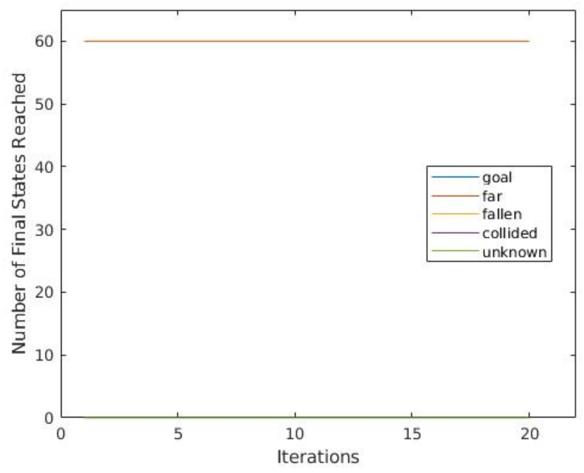


Figure 9: Number of final states reached with dysfunctional left ankle swing actuator

describe why we chose that particular case for analysis, and then describe how runtime monitoring performs in those cases. The run-time monitoring was run for certain number of iterations, certain episodes each with the model repair being run before every iteration. For every case two plots : Number of terminating states reached vs iteration , Figure 5, and the variation of unsafe probabilities, total unsafe probaility before(*total*) and total unsafe probabilities after model repair(*total-af*) , Figure 6, are discussed. Finally, we try to hypothesize and analyze conceptually the outcomes of runtime monitoring in each such setting.

A. *Faulty Isolated Action*

In this part, we describe all the cases that where a single action leads to an unsafe state directly. In our case, we presume that such an action will cause the self-collision in the robot. The various cases are:

- Action 579 causes collision
- Action 579 and 337 cause collision

The actions chosen as faulty were the actions occuring more frequently when the learned pol-

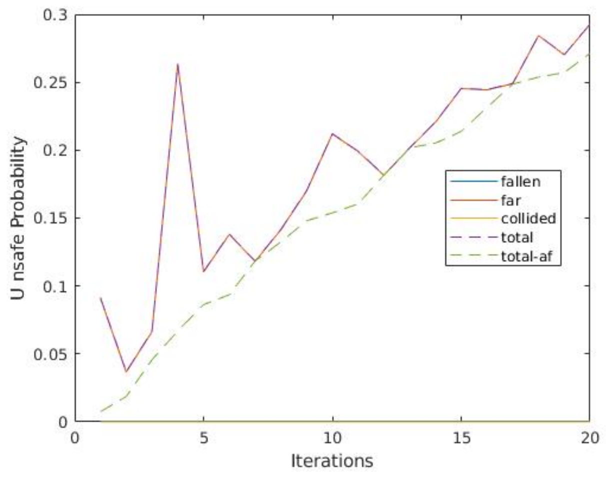


Figure 10: Variation of unsafe probabilities for dysfunctional left ankle swing actuator

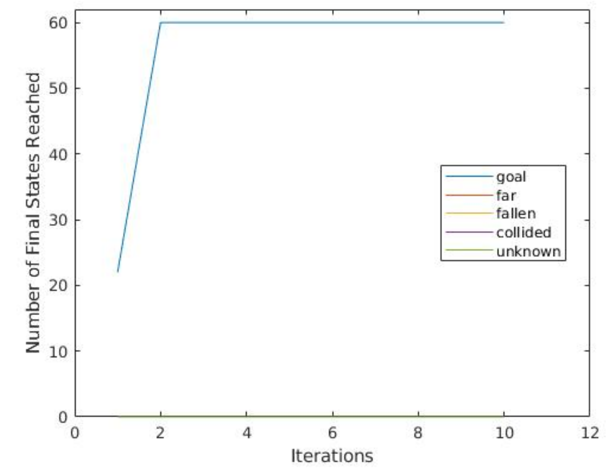


Figure 11: Number of final states reached with dysfunctional left elbow actuator

icy was executed. The runtime monitoring was run for 10 iterations, 50 episodes each.

For the first case, as seen in Figure 5 , for the first iteration all episodes end in collided state. As per Figure 6, the small total unsafe probablility(*total*) in the start corresponds to the previous model generated from the pre-trained policy. Model-repair is able to reduce this to a even smaller value(*total-af*) but this also shows that it is unable to detect the faulty action since the *collided* probability remains small. As more iterations occur, the updated model shows increase in the *collided* probability, causing increase in total unsafe probabilty after initial

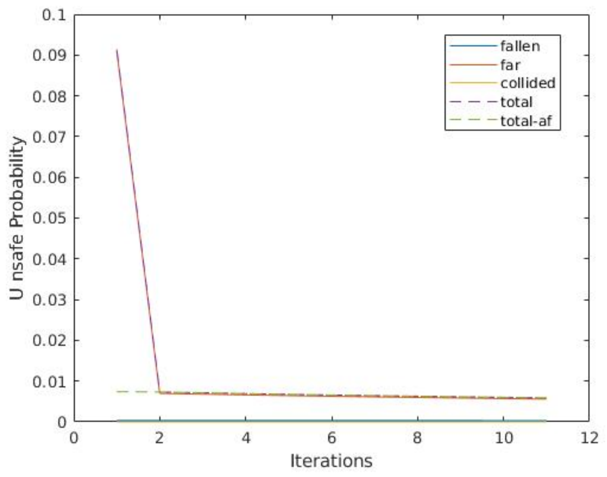


Figure 12: Variation of unsafe probabilities for dysfunctional left elbow actuator

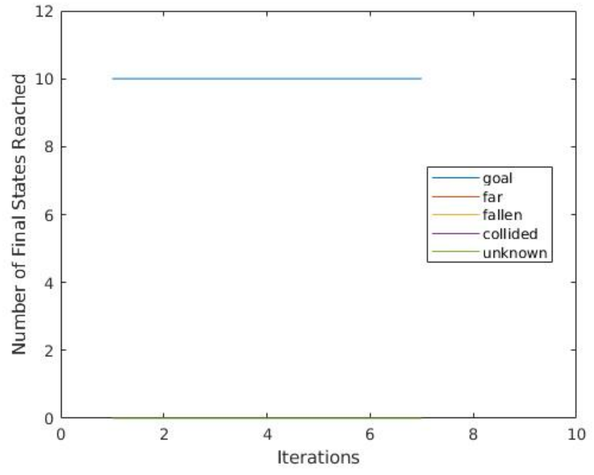


Figure 13: Number of final states reached with both dysfunctional elbow actuators

dip. It is also to be noted that the unsafe total probability ater model repair is again reduced to a very small value as before, indicating runtime monitoring is trying to correct the model and policy. This is evident from the 4th iteration where 45 episodes end in Goal state compared to 15 episodes ending in Collided state, which corresponds to the dip in the *collided* probabilities. The rise in *far* probabilities correspond to new states reached. After this, for further iterations, similar behaviour (45 goal vs 15 collided) is seen i.e. run-time monitoring is unable to improve further. This corresponds to

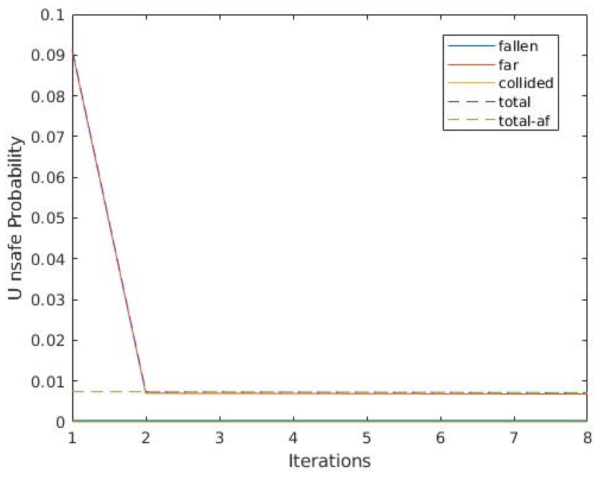


Figure 14: Variation of unsafe probabilities for both dysfunctional elbow actuators

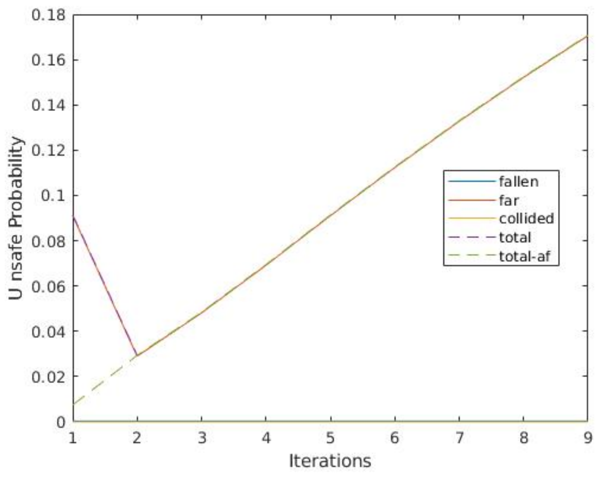


Figure 16: Variation of unsafe probabilities for both dysfunctional shoulder lateral actuators

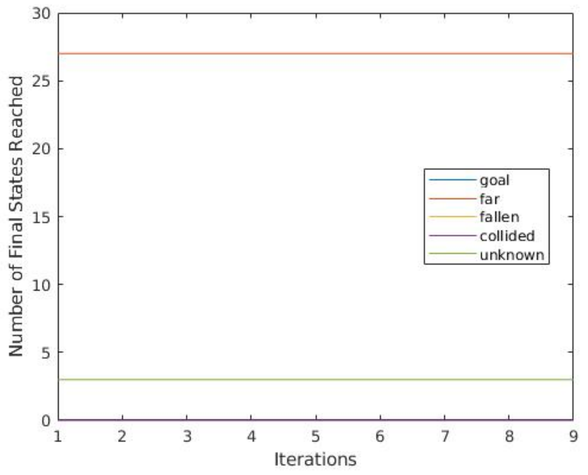


Figure 15: Number of states reached with both dysfunctional shoulder lateral actuators

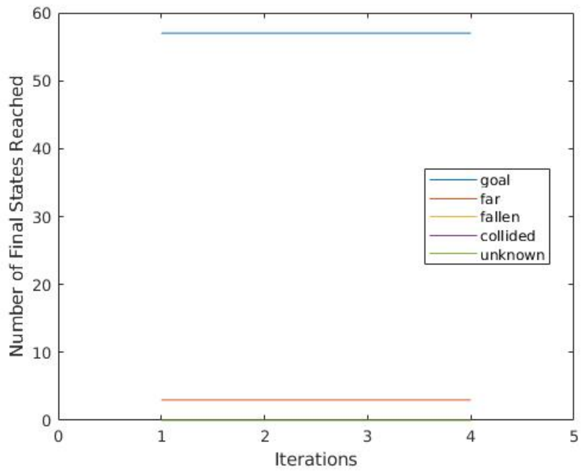


Figure 17: Number of final states reached with both dysfunctional elbow encoders

the rise of total unsafe probabilities and the inability of the model repair to reduce it (*total & total-af* coincide).

In the 2nd case, 20 iterations of 60 episodes each were run. All of episodes reach 'collided' terminal state initially inspite of model-repair reducing total unsafe probability, corresponding to inability to reach goal by pre-trained policy and model generated from it, seen in Figure 7 and Figure 8. It can be observed in further iterations that the *collided* and *total* probabilty increases as model is updated, which are reduced by model repair *total-af*. The effect of this is seen in next iteration, when the *collided* is reduced, i.e. policy and model are modified to reach different states, leading it to terminate in 'far' states due to unexplored states reached. This is also seen in Figure 7 when in 3rd iteration 45 episodes ended in 'far' and 15 in 'collided'. The high increase in *total and far* unsafe probability is corrected by model repair (*total-af*) as in further iterations all episoded again end in 'collided' state. The runtime monitoring fails to further correct the model as the *total & total-af* probabilties are seen increasing and coinciding, with the increasing *collided*

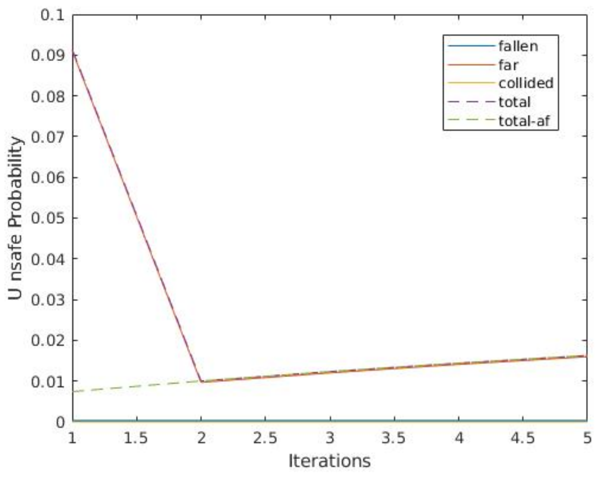


Figure 18: Variation of unsafe probabilities for both dysfunctional elbow encoders

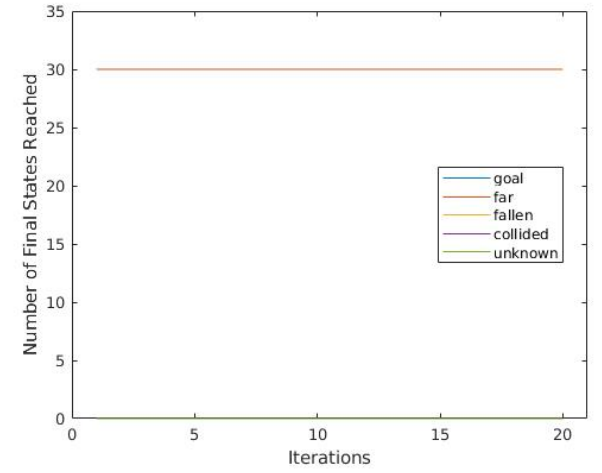


Figure 19: Number of states reached with both dysfunctional shoulder lateral joint encoders

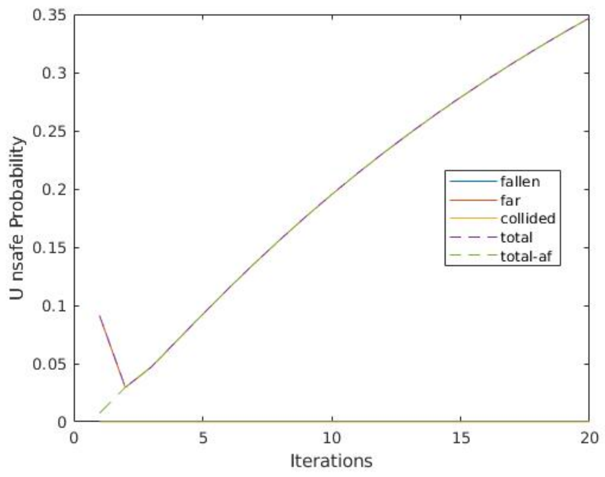


Figure 20: Variation of unsafe probabilities for both dysfunctional shoulder lateral joint encoders

probability being major one.

B. Non-functional Actuators

Here we describe cases where there is no output from the actuator(always remain at zero position). We assume that specific joint actuators have failed in these cases. This scenario is more realistic from a robotic point of view. The various cases are:

- Left ankle swing actuator disabled
- Right elbow actuator disabled
- Both elbow actuators disabled
- Both shoulder lateral actuators disabled

The left ankle swing actuator clearly forms a major part of the standing-up routine, thus its failure being catastrophic. As seen by results in Figure 9 & 10, the robot never reaches the goal state with the pre-trained policy and model-repair as seen in the first iteration, though the model repair reduces the total unsafe probability. The robot reaches ’far’ terminal state as this failure leads to robot being in unexplored states. Runtime monitoring identifies this failure as the *far & total* probabilities increase with iteration. The model repair does reduce the *total-af* probability but is not effective enough to reach the ’goal’ state. This trend continues in further iterations.

The left elbow actuator failure can be termed as non catastrophic as is not a major part of the standing-up routine. The results as shown in Figures 11 & 12 indicate this in first iteration where only 22 out of all episodes reach ’goal’, while others reach ’far’ due to robot reaching unexplored states. The runtime monitoring is able to reduce the total unsafe probability and corrects the model from 2nd iteration, where all episodes reach ’goal’. Further, the *total & total-af* probabilities coincide and keep reducing minimally in further iterations.

The both elbow actuators’ failure is again non-catastrophic failure w.r.t. the standing-up routine. The results in Figures 13 & 14 indicate that pretrained policy with initial model repair is sufficient for reaching the 'goal', while run time monitoring keeps decreasing the coinciding *total & total-af* probability minimally in further iterations.

The both shoulder lateral actuators' failure is a catastrophic failure w.r.t. the standing-up routine. The results in Figures 15 & 16 indicate that pretrained policy with initial model repair is insufficient and robot majorly(27/30) reach 'far' state in first iteration. Even with runtime monitoring the robot never reaches 'goal' state and same behaviour contintues. But run time montioring does reflects the failure in the updated model as the *total & total-af* probabilities coincide and keep increasing in further iterations.

C. Non-functional Encoders

Here we describe cases where the robot receives no information about certain state variables due failure of its encoders. We consider that specific joint encoders are relaying faulty information about the joint angle. We assume that faulty encoders are stuck at their initial (zero) readings. This scenario is also realistic from a robotic point of view, demonstrating how the system can deal with loss of observability. The various cases are:

- Both elbow encoders fail
- Both shoulder lateral encoders fail

Since the defined states encode the joint-encoders' values, such failures may lead to unexplored states causing the 'far' state termination. The both elbow encoders failure can be seen as a non-catastrophic failure, the pretrained policy with model repair performs well to reach 'goal' end state 57/60 times in first iteration, while 3 ended in 'far' state as seen in Figure 17. The runtime monitoring is not able to further correct this behaviour but the coinciding and slightly increasing *far & total & total-af* probabilities shows that it updates the model as per the new behaviour of robot, Figure 18.

Both shoulder lateral encoders failure can be again seen as a catastrophic failure. The results in Figure 19 and Figure 20 depict clearly that pretrained policy with initial model repair and also further model correction in runtime monitoring are not able to drive robot to 'goal' terminal state. All episodes end in 'far ' state. But runtime monitoring does update the model to correspond to new behaviour of the robot as the *far & total* unsafe probabilities increase after the first iteration. The ineffectiveness of the runtime monitoring to correct is also seen by coinciding *total & total-af* probabilities after first iteration.

VI. Conclusions and Further Work

It can be concluded that single iteration of learning and state-space generation for the standing-up task enables the bioloid to stand up. But the policy generated from the Q-learning can be further improved by using model-generation and model-repair which is evident from decreased total unsafe probability in all the cases in first iteration. It was also found that learning with the model repair alone is robust to handle coupled non-critical joint failures, both actuator and encoder based, but it fails in case of non critical single actuator failure and critical joint failures.

The runtime monitoring and model correction provides significant improvement over learning for possible practical faults that might arise due to difference in simulation and real-world environment and systems. This works well particularly for the cases when it is needed to find slightly different strategies to stand-up, e.g. in case of single action collision fault. It performed well for non-critical single actuator failure(elbow) where it is able to correct the model completely, whereas the learning failed. But it fails when a drastic change in the learned strategy is needed, evident from double action collision fault case. It also fails to correct the model in case of catastrophic single or double actuator or encoder based joint failure (though it is an open research question that whether a humanoid can be made to stand-up with such catastrophic shoulder/ankle joint failures).

It is observed that performing the learning for only one iteration limits state space, hence it is difficult for runtime monitoring to readjust the policy evident from the 'far' state episode termination. Thus, it can be argued that providing a feedback at learning level may help the robot to recover from catastrophic failures. This means adding an outer loop connecting runtime monitoring and state-space generation and Q-learning in the Framework shown in Figure 4, which forms a major future update to be done in the existing architecture.

The performance of the algorithm with dynamic robot faults also needs to be tested in future. An example of this can be a faulty actuator which doesn't reach the commanded value, but reaches some lesser value or a random value. The algorithm is also to be tested on the real Bioloid robot instead of the simulated one.

Further improvement can be in the model repair heuristics used. Overall, the framework satisfactorily performs policy repair to avoid unsafe states dynamically, thus tackling environmental uncertainties in cases where re-learning is not viable.

REFERENCES

[1] J. Stückler, J. Schwenk, and S. Behnke, "Getting back on two feet: Reliable standing-up routines for a humanoid robot.," in *IAS*, pp. 676–685, 2006.

[2] J. Morimoto and K. Doya, "Reinforcement learning of dynamic motor sequence: Learning to stand up," in *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, vol. 3, pp. 1721–1726, IEEE, 1998.

[3] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and Autonomous Systems*, vol. 36, no. 1, pp. 37–51, 2001.

[4] E. Schuitema, M. Wisse, T. Ramakers, and P. Jonker, "The design of leo: a 2d bipedal walking robot for online autonomous reinforcement learning," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 3238–3243, IEEE, 2010.

[5] F. Leofante, S. Vuotto, E. Ábrahám, A. Tacchella, and N. Jansen, "Combining static and runtime methods to achieve safe standing-up for humanoid robots," in *International Symposium on Leveraging Applications of Formal Methods*, pp. 496–514, Springer, 2016.

[6] "Bioloid premium kit." http://en.robotis.com/index/product.php?catecode=121010.

[7] "Dynamixel actuators." http://en.robotis.com/index/product.php?catecode=101010.

[8] "Bioloid urdf model." https://github.com/dxydas/ros-bioloid.

[9] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 1321–1326, IEEE, 2013.

[10] "Storm – a modern probabilistic model checker." http://www.stormchecker.org/index.html.

[11] S. Pathak, E. Ábrahám, N. Jansen, A. Tacchella, and J.-P. Katoen, "A greedy approach for the efficient repair of stochastic models," in *NASA Formal Methods Symposium*, pp. 295–309, Springer, 2015.