First do not fall: learning to exploit a wall with a damaged humanoid robot

Timothée Anne[1], Eloïse Dalin[1], Ivan Bergonzani[1], Serena Ivaldi[1], and Jean-Baptiste Mouret[1]

***Abstract*—Humanoid robots could replace humans in hazardous situations but most of such situations are equally dangerous for them, which means that they have a high chance of being damaged and falling. We hypothesize that humanoid robots would be mostly used in buildings, which makes them likely to be close to a wall. To avoid a fall, they can therefore lean on the closest wall, as a human would do, provided that they find in a few milliseconds where to put the hand(s). This article introduces a method, called D-Reflex, that learns a neural network that chooses this contact position given the wall orientation, the wall distance, and the posture of the robot. This contact position is then used by a whole-body controller to reach a stable posture. We show that D-Reflex allows a simulated TALOS robot (1.75m, 100kg, 30 degrees of freedom) to avoid more than 75% of the avoidable falls and can work on the real robot.**

***Index Terms*—Machine Learning for Robot Control, Humanoid Robot Systems**

I. Introduction

HUMANOID robots are some of the most versatile machines ever designed [1]. They can grasp, pull, push, hold, and reach for both low or high places, but they can also walk, climb stairs, or crawl in a tunnel. Thanks to their small footprint, they can navigate in narrow spaces, and, more generally in all the environments designed for humans.

This versatility makes humanoids ideal machines to be deployed in risky and complex situations, like industrial disasters or space operations, during which more versatility means a higher probability of having the right set of capabilities to solve the problem at hand [1], [2]. This contrasts with industrial robots, which are designed to perform the same set of tasks continuously in a well-defined environment.

Nevertheless, the versatility of humanoid robots comes at the cost of an increased fragility: they have more joints than most robots and a single joint failure often results in a fall [2]. This fragility is especially concerning because humanoid robots would be the most useful in situations that are too dangerous for humans, which are likely to be equally dangerous for a robot. A deployed humanoid robot is therefore likely to be damaged during some of its missions.

Preprint June, 2022. Full reference: Anne et al. (2022). "First do not fall: learning to exploit a wall with a damaged humanoid robot". In Robotics and Automation Letters.

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). This project is supported by the CPER SCIARAT, the CPER CyberEntreprise, the Direction General de l'Armement (convention Inria-DGA "humanoïde résilient"), and the Creativ'Lab platform of Inria/LORIA.

[1]Université de Lorraine, CNRS, Inria Nancy - Grand Est. Contact: jean-baptiste.mouret@inria.fr

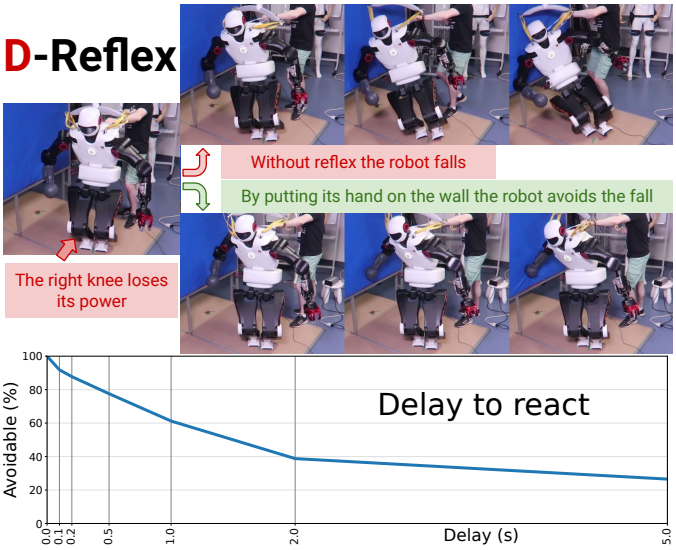

Fig. 1: (top) A humanoid robot detects a fault on one of its legs. If it does nothing, it falls; but it can recover its stability by putting its "hand" on the wall at the appropriate location (depending on its posture, the wall distance, and the wall orientation). (bottom) The percentage of avoidable falls (see Sec. V-C for the definition of "avoidable") decreases when the delay taken between damage and reflex increases; the robot has only a few milliseconds to react with a high success rate.

The traditional approach for robot damage recovery is to identify the damage, update the model, then use the updated model to perform the tasks [3]. Unfortunately, a falling humanoid robot has only a few milliseconds to make a decision (Fig. 1), whereas identifying the dynamical model of such a highly redundant robot requires extensive data and specific trajectories [3], [4]. A few learning-based damage recovery algorithms for 2-legged [5], 4-legged [6], [7] or 6-legged robots [8]–[10] were recently published, but they all assume that the robot can try a behavior, fall, and try again until it finds a compensatory behavior. Humanoids usually cannot afford to fall, and trial-and-error would require an ability to stand up without a perfect knowledge of the robot model, which is very challenging.

Given the envisioned missions, humanoid robots are more likely to be used in indoor environments, for instance in damaged buildings, than in open fields. As a result, these robots are likely to have walls or furniture within reach: taking inspiration from human reflexes, a promising strategy in case of leg damage is, therefore, to lean on the nearest wall to avoid the fall.

In this article, our main contribution is a method that allows damaged humanoid robots to avoid many falls by leaning on a nearby wall (Fig. 1). Once the robot is stable, another algorithm could update the model and activate compensatory behaviors (e.g., walking on one leg using the wall). Our main assumptions are that (1) we can detect the occurrence of a fault in the leg, but we do not know its origin (missing parts, disconnected from power, or locked); meaning that we do not know the exact model of the damaged robot and (2) we know the distance and the angle to the nearest wall (for instance with a dedicated sensor on the shoulder). With these assumptions, our method, called D-Reflex (for "Damage-Reflex"), finds in a few milliseconds where to put the hand on the wall, and reaches a stable state using a whole-body controller [11], [12].

The main idea of D-reflex is to learn a neural network that predicts the success chance of each potential contact location on the wall given a posture and a wall configuration. This neural network is learned with supervised learning using a dataset created by simulating many situations and many potential contact positions. Splitting the process into these two steps—creating a dataset in simulation, then learning a predictor—makes it possible to exploit the mature and well-understood supervised learning algorithms while separating the learning process from the simulation process. Once learned, the neural network can be queried in a few milliseconds, which is ideal for a fast emergency reflex while the robot is falling.

We report experiments with both a simulated and a real TALOS humanoid robot (1.75m, 100kg, 30 degrees of freedom) [13] with a leg damaged in several ways.

II. Related Work

A. Multi-contact planning and damage identification

Multi-contact planning, for instance choosing to put a hand on a wall, has been a long-lasting research topic in the humanoid robotics community (e.g., [14], [15]). However, planning algorithms tend to require more than the few milliseconds that a falling humanoid robot can afford (e.g. about 100ms for a known target point [16]). In addition, all planning algorithms assume a known model of the dynamics, which is not the case when the robot is damaged, whereas the dynamics of the fall are typically very different with a damaged robot.

A preliminary step to any planning algorithm is therefore to identify the model of the damaged robot [3]. Unfortunately, identifying the dynamical model of a humanoid requires at least several minutes of data collection and specific "exciting" trajectories. For example, to identify the TALOS robot's [13] elbow joint, Ramuzat *et al.* [4] required 230*s* of data collection and a well-chosen exciting trajectory. It seems highly optimistic to be able to identify a new model with a few milliseconds of sensor data. Overall, we see the present contribution as a preliminary step to a system identification step: the robot takes an emergency decision, then, once stable, it can get data to identify the new model and rely on model-based planning or control algorithms with an updated model.

B. Fall and damage mitigation

Inspired by how humans react when falling in front of a wall, Cui *et al.* [17] recently proposed to move the robot arms in a way that maximizes the ellipsoid stiffness, thus increasing stability and shock absorption. Compared to the present work, Cui *et al.* only worked on collisions with a frontal wall and, more importantly, with an intact robot whose model is known.

When the fall is inevitable, several methods have been demonstrated on different humanoids to mitigate the damage. During the "pre-impact" phase, the robot adapts its posture by avoiding hand-designed "fall singularities" postures that increase the impact [18], by seeking to take a safe posture when falling on the back [19], or, by performing a rollover strategy [20]. During the "impact-time" phase, the PD-gains are automatically adapted by incorporating them in the QP-controller to prevent the actuators from reaching their torque limits while still reaching the desired posture [21]. During the "post-impact", a force distribution quadratic program distributes the exceeding linear momentum gathered during fall into the different body parts [22]. These methods do not try to avoid the fall but mitigate the damage induced by the fall. Overall, this line of work is complementary to ours: while many falls can be avoided by adding a contact to the wall, some falls cannot be avoided in that way and the only behavior left is to mitigate the damage.

C. Machine learning for damage recovery and mitigation

A promising approach to allow robots to adapt to unforeseen damage is to learn dynamical models with generic machine learning methods, like neural networks or Gaussian processes. To minimize the amount of data required, current methods leverage a simulation of the intact robot [10] or meta-learning to train the model for fast adaptation [6], [7], [9]. Successful experiments with 4-legged [6], [7] and 6-legged [9], [10] have been reported, but, to our knowledge, not with 2-legged robots. One important difference between humanoids and other multi-legged robots is that the latter can afford to fall during learning because getting up is easy.

Spitz *et al.* [5] investigated how a damaged humanoid robot can adapt its behavior by avoiding states that previously led to a fall. Similarly, Cully et al. [8] designed a learning algorithm that allows a damaged 6-legged robot to find a new behavior that does not rely on the damaged parts. In both cases, the authors used an episodic setup in which the robot can try a behavior, fall, and try again many times. This setup is not applicable when the objective is to avoid the fall immediately after damage.

To our knowledge, the closest work to ours is for a quadruped robot that performs "cat-like" acrobatics to land on its feet when falling from several meters [23]. To solve this problem, the authors used a trajectory optimization algorithm to generate a dataset of examples in simulation, then trained a neural network policy that imitates these reflexes but is fast enough to be used on the robot. Similarly, the D-reflex approach first solves the problem with extensive simulations, then uses these results to train a fast neural network that selects the most appropriate behavior.

III. PROBLEM

A. Considered Damage Conditions

A robot can be damaged in many ways and we often cannot determine them precisely. We use three damage conditions: amputation (missing parts), passive actuators (the actuators do not deliver any torque, i.e., the joints can turn freely in the corresponding axis), and locked actuators (the joint position is fixed). The first two damage conditions are critical for the stability of the robot and very often result in a fall when applied to one of the legs, but they cannot be distinguished by querying the control board of each joint: in both cases, the control boards do not answer or send an error.

To show that our method is robust to different combinations of damage conditions, we sample different combinations of those three damage conditions on the six joints of the leg. We call J the set of damaged joints.

B. Formulation

A humanoid robot suffers an unexpected damage combination on one of its legs which would often result in a fall if nothing is done. The goal is to find a contact position on the wall that allows the robot to avoid the fall and stay stable (Fig. 1).

We know:

- that one of the legs is damaged and which one: we assume that the faulty joints control boards do not answer or return an error;
- the presence of a plane wall within arm's reach, its distance d and its orientation α with regard to the robot (this position can be easily known with a RGB-D sensor on the shoulder of the robot and fitting a plane to the point cloud [24]);
- the current posture q of the robot when the fault is detected.

We do not know:

- the set of damaged joints J and the nature of the damage condition: the leg could be fully amputated, but it could also have lost the power;
- the successful contact positions on the wall (there are usually several possible successful contact positions).

We want to learn a function $f : (q,d,\alpha) \rightarrow (x^*,y^*)$ that returns a successful contact position *robust* to the nature of the damage condition. We assume that the robot has a controller that can make it move to reach this position.

IV. METHOD

A. Whole-body Control

The humanoid robot is controlled with a whole-body controller (WBC) based on quadratic programming [11], [12]. At each time step, the robot searches for the acceleration of each joint that minimizes a sum of quadratic cost functions constrained by the dynamical model of the robot (an equality constraint) and other equality/inequality constraints:

$$\begin{aligned}(\tau^*,\ddot{q}^*) = \underset{\tau,\ddot{q}}{argmin} & \sum_{k=0}^{n\_tasks} w_k \|A_k(q,\dot{q})\ddot{q} - b_k(q,\dot{q})\|^2 \\ \text{s.t. } & A_{ineq}(q,\dot{q})\ddot{q} \leq b_{ineq}(q,\dot{q}) \\ \text{s.t. } & A_{eq}(q,\dot{q})\ddot{q} = b_{eq}(q,\dot{q}) \\ \text{s.t. } & \tau = M(q)\ddot{q} + F(q,\dot{q})\end{aligned} \tag{1}$$

where τ^* and $\ddot{q}^*$ are the optimal joint torques and accelerations, w_k is the weight of the task k described by $A_k(q,\dot{q})$ and $b_k(q,\dot{q})$, M is the joint-space mass matrix and F contains all the non-linear terms (Coriolis, centrifugal, gravity, and contacts).

In this work, we used the controller described in Dalin et al. [12]. The main tasks are the Cartesian position and orientation of the hands and feet, the position of the center of mass, and a default postural task which is used to ensure the unicity of the QP problem and bias towards a "neutral" position. The constraints are the feet contact with the floor and the joint position, velocity, and acceleration bounds.

Once the optimal torque and joint accelerations are computed, we integrate them using the model of the robot to get the desired joint positions, which we pass to the low-level joint controllers. In the real TALOS robot, the joints controllers are PID controllers implemented on Ethercat boards [13]. In simulation, we use a Stable-PD controller [25] that leverages the model of the robot to compute torques given a position target, which is in our experience a good approximation of well-tuned PID controllers.

Please note that the model used by the controller is not updated after the damage because we do not know the extent of the damage condition. As a result, the solution needs to be robust to different combinations of damage conditions.

B. Data Collection

The first part of the process consists in generating random configurations of the wall and postures of the robot, and collecting the corresponding truth value of contact positions (Fig. 2, part 1).

1) Sampling random configuration: To randomly sample plausible random postures of the robot, we first randomly sample target hands positions in a cuboid facing the robot. We set these targets in our whole-body controller as Cartesian tasks and let it run for 4s which makes the robot take a posture q. Each random hands positions induces a different but feasible posture, such as knee bending to reach a lower position or rotated torso with arms raised.

2) Sampling wall positions: We need random wall configurations (distance d and orientation α) that are within arm's reach but that do not collide during the initial motion to reach the target posture (because this would make the robot fall before any damage). To ensure this property, we exclude wall configurations that collide during the first $4s$ of motion.

3) Construction of contact maps: Once we have the posture of the robot q and the wall configuration (distance d and orientation α), we discretize the wall's plane and run the simulation for each considered contact position by (1) reaching the posture q, (2) damage the robot in the simulation after $4s$ without waiting for the robot to be stable, i.e. the current

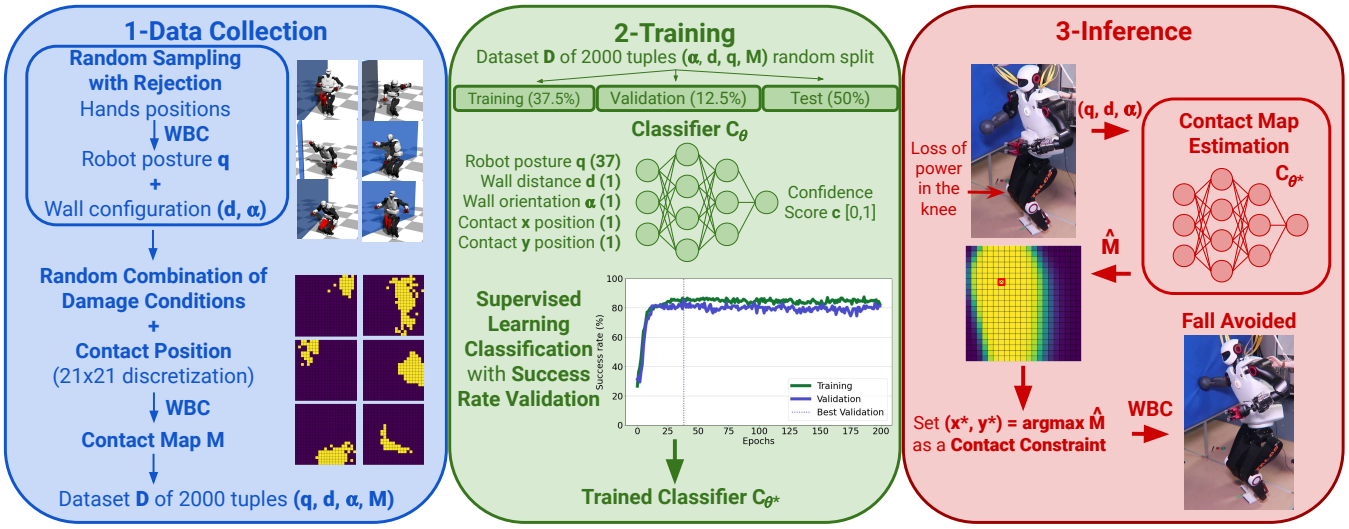

Fig. 2: Overview of D-Reflex. In the first step (Data Collection), we sample different situations: wall configurations (distance and orientation), robot postures, and damage combinations, and simulate the behavior of the robot for each possible point on a $21\times$ 21 grid on the wall (441 simulations for each situation). In the second step, we train a neural network classifier to predict the success (avoiding the fall) for each point of the grid. In the third step, we query the trained neural network for each point of the grid to select the best contact position, and we set it as a contact constraint in the whole-body controller.

momentum and angular momentum induced by the motion is kept, (3) add a Cartesian contact constraint in our whole-body controller to ask the hand to be in contact at the given position, and (4) let the simulation run for 11s. A contact position is deemed correct if at the end of this 15s-episode the robot has not touched the floor with anything else than its feet, or the wall with anything else than its hand. We only consider contacts with the hand because it allows clean contacts with known interaction forces (the wrist has a force-torque sensor) and can reach the farthest positions to make contact as soon as possible. This gives us a Boolean matrix $M \in \{0,1\}^{(21,21)}$ that we call "Contact Map". Fig. 2.1 shows some examples.

C. Training

Once we have a dataset D of samples (q,d,α,M) split into a training set (37.5%), a validation set (12.5%), and a test set (50%), we want to learn the function

$$f : (q,d,\alpha) \rightarrow (x^*,y^*)$$

where (x^*,y^*) is a successful contact position in the wall referential. As the contact map M often contains several successful contact positions, we cannot directly learn f.

Instead, we learn the classification function that predicts the success of each input point:

$$C : (q,d,\alpha,x,y) \rightarrow c$$

where $c \in [0,1]$ represents the confidence of the contact position (x,y) to be a successful contact position.

To estimate this classification function C, we train a neural network $\hat{C}_\theta$ parameterized by the set of weights θ using PyTorch with Cross-Entropy error and the Adam optimizer. After performing an extensive random search on the hyper-parameters, we have selected a fully-connected feed-forward neural network with 2 hidden layers of 1024 units with ReLU activation function, a dropout of 0.2, and a learning rate of 10^{-5}. We obtained similar results with other combinations of network architecture, cost function, and optimizer.

To get an estimation of the function f, we query the neural network with points $(x,y) \in (X,Y)$ on the wall and select the point with the highest activation:

$$\hat{f}(q,d,\alpha) = \underset{(x,y)\in(X,Y)}{argmax}\ \hat{C}_\theta(q,d,\alpha,x,y).$$

Since the points are 2-dimensional and there is no need for a millimeter-scale precision, this optimization can be effectively performed using a straightforward grid search. In that case, it typically requires a few hundred calls to the neural network (e.g., 441 calls for a 21×21 grid), which is easily done in parallel on modern GPUs (about 4ms for 441 queries on an Nvidia RTX 2080) and CPUs (about 17ms on the Talos's CPU).

In supervised learning, we would periodically evaluate the prediction score of the neural network on the validation set to avoid overfitting and select the final set of weights. Here, the utility of our learned neural network is measured by the effectiveness of the selected contact position, which we evaluate by running a simulation with a damaged robot. In other words, we evaluate the neural network by looking at $\hat{f}(q,d,\alpha)$ on the validation set instead of the traditional $\hat{C}_\theta(q,d,\alpha,x,y)$ because we want to evaluate the "end product" and not the neural network itself. At the end of the optimization, we select the set of weights θ that corresponds to the highest success rate on the validation set (Figure 2.2).

D. Inference

During its mission, the robot performs its tasks and detects damage in one of its legs by querying the boards periodically.

We get the last known posture of the robot q, the distance d, and orientation α to the closest wall. For this preliminary work, we hypothesize that the robot can evaluate α and d with a dedicated sensor.

We query the learned model $\hat{C}_\theta$ for a grid of points to estimate the best contact position $(x^*, y^*) = \hat{f}(q, d, \alpha)$. Fig 3 shows examples of predicted contact maps. We then add a contact constraint in the whole-body controller so that it puts its hand on the wall to recover its balance (Fig. 2.3).

E. Implementation

1) Triggering the reflex: We assume that the robot knows when an actuator is faulty so that we can instantly trigger the reflex. We put the robot in an "urgent mode" that removes the non-essential tasks: we only keep the feet contact constraints, the joints bounds constraints, the position task of the Center of Mass (CoM), and the postural task. Finally, we add the hand target contact at the selected location on the wall and let the whole-body controller generate the trajectory to reach it.

The CoM task requires a Cartesian position. By default, the target position is between the two feet. In our case, we hypothesize that a single leg is working, but, since we want a contact on the wall, the new CoM target should be somewhere between the remaining foot and the contact point. However, we do not know the real contact position because the model is incorrect. After some preliminary experiments, we have decided to not change the CoM target as it has shown good results: if we put the CoM target on the remaining foot or too close to the wall the success rate declined significantly. A better solution would be to learn the CoM target in addition to the hand contact position, which we will explore in future work.

2) Post-contact updates: We empirically found that, due to the mismatch between the incorrect model and the simulated damaged robot, the desired contact point is rarely the real contact point attained by the hand. To take this into account, we use the force sensor of the wrist to detect the contact between the hand and the wall. Then, we update the contact constraint with the current robot state to stop the robot from moving its hand, which would result in the robot pushing on the wall, bouncing back, making it more likely to fall.

V. Experiments

We control 30 degrees of freedom of the TALOS robot [13]: 7 for each arm, 6 for each leg, 2 for the torso, and 2 for the head. We replaced the gripper with a ball in our simulation to be close to the experiments on the real robot (to avoid breaking the gripper), but similar results are obtained with a simulated gripper. The robot is simulated using the DART physics library [26] through the RobotDART simulator[1].

The initial posture of the robot is obtained by selecting random target positions for the hands and letting the whole-body controller reach this target in 4s. The target positions are inside the cuboid: $[-0.1m, +0.2m]$ in the sagittal axis, $[-0.4m, +0.4m]$ on the frontal axis, and $[-0.5m, +0.4m]$ on the longitudinal axis. The wall distance is sampled between $0.4m$ and $1m$. The orientation is sampled in $[-1, 1]$ rad.

The contact maps of the dataset use 21 positions on the horizontal and vertical axis, which results in 441 positions. They range from $-0.75m$ and $+0.75m$ on the horizontal axis and $-0.5m$ and $+0.75m$ on the vertical axis. Fig. 3 shows examples of contact maps.

We only perform the experiments on the right side of the robot (damaging the right leg and seeking contact with the right hand). The robot being perfectly symmetrical, the data and the same neural network can be used for the other side (damaging the left leg and seeking contact with the left hand). To check this, we trained a D-Reflex using data from the right side (Sec. IV-C), then used it with a damaged left leg by symmetrizing the inputs of the classifier. In 98.5% of the cases, the robot is as successful as with the right side. The resulting discrepancy is likely to come from simulation artifacts.

We sample 2000 different situations of wall configuration, robot posture, and damage combination. We let the simulation run for 4s, after which we trigger the damage combination and immediately start the reflex. We stop the simulation after 15s or until a fall is detected. For each situation, we create a discretized contact map. Overall, this dataset requires 882,000 simulations of 15s, which we run in parallel on a multicore computer in less than 3 days.

The source code and the dataset are available online[2].

A. Baselines

We compare against two baselines:

- No Reflex: we do nothing, to highlight the importance of a reflex strategy;
- Random Reflex: we randomly select the contact position on the wall, to highlight the difficulty of the problem and the necessity of learning.

B. Ablations and Additions

To understand the contribution of each part we also compare against 3 "ablation experiments":

- Posture Ablation: we remove the posture q from the neural network inputs and re-train the network (this allows us to answer the question: "does the contact position depend on the posture?");
- Wall Ablation: we remove the wall configuration (distance d and orientation α) from the neural network inputs and re-train the network ("does the contact position depends on the wall configuration?");
- Both Ablation: the contact point is independent of both the robot posture and the wall configuration ("is there a solution that works for any wall configuration and robot posture?"). In this case, we do not train a neural network but, instead, we use the training data to select the average best contact position.

To check that we are not ignoring any useful information, we compare against 2 "addition experiments":

[1]https://github.com/resibots/robot_dart

[2]https://github.com/resibots/d-reflex

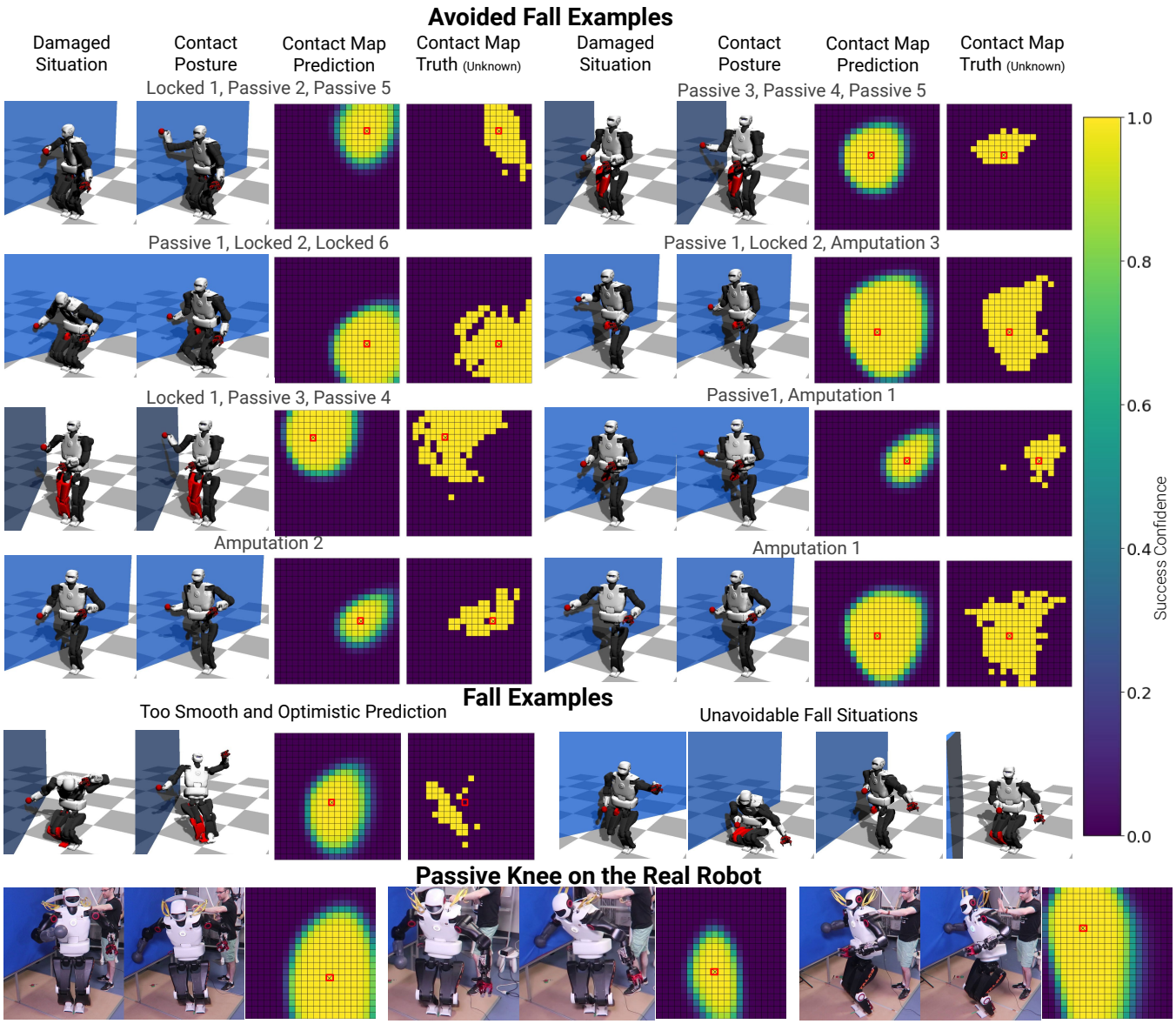

Fig. 3: Examples of damaged situations and corresponding contact or falling postures using D-Reflex on the TALOS [13] robot in simulation and on the real robot, the video https://youtu.be/hbuWr-ZNAtg shows different examples. We also show the contact map estimated by our neural network and the true contact map measured but unknown during training. We distinguished two kinds of failures: when the predicted contact map is too smooth and optimistic compared to the truth, and when there is no successful contact point, i.e., the fall is unavoidable by seeking a contact on the wall.

- J Addition: we add the set of faulty joints J to the neural network inputs and re-train the network (this allows us to answer the question: "is it useful to know which joints are damaged?");
- $J + \mathrm{d}q$ Addition: we add to the neural network inputs the set of faulty joints J and the joints velocities $\mathrm{d}q$ and re-train the network (this allows us to answer the question: "is it useful to know which joints are damaged and the joints velocity $\mathrm{d}q$?").

C. Evaluation

By analyzing the contact maps, we found that for about 31.5% of the situations of the dataset there are no position of the hand on the wall that prevents the fall. Fig. 3 shows 4 examples of such unavoidable situations. Using a decision tree classifier, we found that two criteria discriminate most of the data: the distance of the right hand to the wall and the wall orientation. If the wall is too far from the right hand the fall becomes impossible to avoid by putting the right hand on the wall. Moreover, in many cases, only a few sparse contact positions of the 21×21 grid allow the robot to avoid the fall. We consider that these contact positions are unlikely to exist with a real robot and are most likely due to a combination of "chance events" that is very sensitive to simulation details. Overall, only 37.6% of the simulated falls are realistically avoidable. As a consequence, we define a fall as "avoidable" if and only if the corresponding situation of wall configuration and robot posture leads to at least 9 (one position and its 8 neighbors on a grid) contiguous successful contact positions. We focus on these "avoidable situations" as

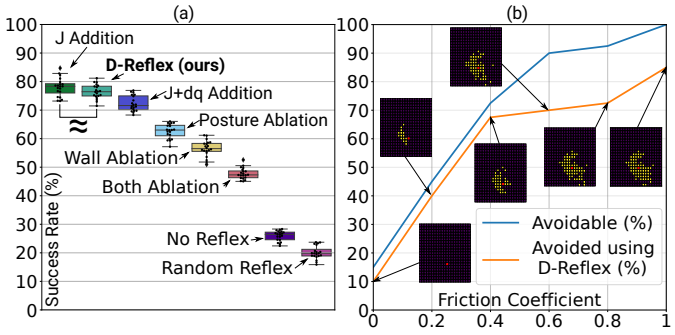

Fig. 4: (a) Success rate considering only avoidable situations. The box plots show the median and quartiles, the bullets being the values of the evaluations. For each variant, the learning algorithm was run 20 times on different splits of the dataset. All methods are significantly different from one another using a t-test with Bonferroni correction (p-value ≤ 0.001) except D-Reflex and the J Addition. (b) Percentage of avoidable situations and situations avoided using D-Reflex, with a model trained for the friction 1, depending on the wall friction coefficient, plus examples of contact maps.

it is straightforward to generate an arbitrarily high number of non-avoidable situations and thus make the avoided fall rates arbitrarily low.

We randomly split this dataset into training (37.5%), validation (12.5%), and evaluation (50%). To ensure that our results do not depend on a single, lucky run of the learning algorithms, we replicated the learning algorithm 20 times using 20 different splits of our dataset and different seeds.

D. Results

Fig. 3 shows many examples of successful balance recoveries, as well as examples of typical failures; Fig. 4 shows the comparison of the success rate for the avoidable conditions of our method against the different ablations, additions, and baselines. All methods are significantly different from each other, except the D-Reflex and the J Addition. The video (https://youtu.be/hbuWr-ZNAtg) presents more examples.

1) Our method: D-Reflex allows the TALOS robot to successfully recover from the fall for $76.4\% \pm 2.5\%$ of the avoidable conditions, Fig. 4. Fig. 3 showcases some examples of successful recoveries as well as some typical failures. Failure cases mostly happen when the neural network overestimates the size and the smoothness of the possible contact zone (Fig. 3, fifth row); more tuning of the learning process or using another regression technique (e.g., random forests) could lead to successful contact zones that are less smooth and more accurate, with the same training data. Overall, our approach significantly (t-test with p-value ≤ 0.001) increases the likelihood of avoiding the fall and thus prevents the robot from being more damaged (which could be very costly) and allows it to finish its mission despite the damage (especially if the robot has access to a locomotion strategy that can deal with a damaged leg).

2) Baselines: Doing nothing (without any reflex) results in a low success rate ($26.4\% \pm 1.7\%$ of the avoidable falls). It is not zero because some situations do not result in a fall (e.g. a locked ankle). Using a random contact position results in a lower success rate ($19.8\% \pm 2.2\%$ of the avoidable falls) because moving the arm randomly creates unnecessary motions that decrease the stability of the robot. These two cases show the need for a reflex.

3) Ablations: When both the wall configuration and the robot posture are ignored to choose the contact point ("Both Ablation"), the robot avoids $47.3\% \pm 1.9\%$ (versus 76.4% for D-Reflex) of the avoidable falls. This corresponds to choosing the best point of the wall on average on the training set, which is roughly in the middle of the wall. This demonstrates that the neural network does more than choose the same "average position". Both the wall configuration and the robot posture are necessary information for the classifier; of the two, the wall configuration is the most important (Fig. 4).

4) Additions: Adding the faulty joints information ("J Addition") does not significantly improve the performance and requires a stronger assumption about our knowledge of the damage condition. Adding the joints velocities ("$J + \mathrm{d}q$ Addition") significantly reduces the performance; as the robot does not perform dynamic motions, adding uninformative inputs only decreases the neural network performance (which could be corrected by re-optimizing the hyper-parameters).

5) Friction: DART simulates the contact using the rigid contact dynamics, taking into account friction, bouncing, and restitution, which can be parameterized to simulate different materials. In our cases, the friction has a default coefficient of 1 and a restitution coefficient of 0. Decreasing the friction coefficient removes successful contact positions, which decreases the percentage of avoidable falls. A D-Reflex trained with a friction of 1 generalizes to lower frictions down to 0.4 (Fig. 4b) without degrading too much its performance, from 85% to 67.5%. The decrease in performance comes from the large reduction of successful contact points when the friction is lower than 0.4 (Fig. 4b). During our experiments on the real robot, we did not notice a significant discrepancy between simulation and reality regarding friction.

E. Use of the updated model in the whole-body controller

In these experiments, the whole-body controller uses the model of the intact robot to compute the joint angles of the damaged one: we assumed that the model of the damaged robot was not known during the fall avoidance phase and therefore we could not use an updated model. Nevertheless, as a baseline, we checked what happens when we use the model of the damaged robot when generating the dataset; we replicated the learning 25 times, but we only considered the amputation of the knee.

The results are statistically equivalent to those obtained when the controller uses the model of the intact robot (median fall avoidance rate: $80.7\% \pm 2.5\%$ for the intact model versus $82.3\% \pm 2.4\%$ with the updated model). This lack of difference is due to the fact that our method looks at the result of a simulation to decide on the success of a particular location on the wall, and not at how well the robot performs an expected motion. Put differently, a controller with an updated model is likely to put the robot's hand closer to the target location,

whereas it often misses this target when using the model of the intact robot; but this does not matter because what is stored is the success (fall avoidance) for a target, regardless of where this target actually leads on the wall. This means that the whole-body controller is used to generate a trajectory and the simulator tells the algorithm whether this trajectory is successful or not: generating this trajectory differently does not fundamentally change the result. Similarly, a planning or model-predictive control algorithm [23], [27] could be used instead of our whole-body controller to generate these trajectories: we do not expect any significant change in the result because an algorithm will still be needed to choose *where* to put the hand, which is the problem solved by D-reflex.

F. Experiments on the real robot

In these experiments, we placed the robot in a known location with respect to a wall with a mattress (to be able to repeat falls with and without D-Reflex) and set it in a known posture. We cut the power of the knee's actuator. The loss of power is detected and triggers the D-Reflex. Overall, in three out of the four situations that we tested, the robot successfully avoided the fall, whereas it fell when no reflex was triggered (see the video https://youtu.be/hbuWr-ZNAtg). These experiments demonstrate that the robot is fast enough to execute the learned reflex and that the learned solutions are robust enough to cross the reality gap most of the time. The onboard CPU computes the contact location using the classifier on average in 16.7ms [min:14.0ms, max:21.9ms], but a more recent CPU and/or a GPU can achieve better performance.

VI. Conclusion

Given the robot posture and the configuration (distance and orientation) of a wall within arm's reach, our method, D-Reflex, uses a trained neural-network classifier to estimate a contact position that allows the robot to avoid the fall in more than 75% of the avoidable cases. Our method is robust to different combinations of damage conditions (locked actuators, passive actuators, and amputation) and does not require the knowledge of the correct damaged model of the robot. It relies on a whole-body controller [12] and sensory information that is easy to acquire.

Once stabilized, the robot will need to update its model and its position so that it can be controlled again. Numerous model identification methods can be investigated [3]. The robot can then continue its mission despite being damaged.

The main assumption of this work is that there exists a wall next to the robot. In future work, we will extend our method to select target contact positions for the hands in more complex environments (other walls, furniture, ...). In that case, we would not be able to cover the possible contact positions with a simple 2D grid: the main challenge will be to build a dataset that includes successful contact positions.

D-Reflex is designed to avoid falls in complex, high-risk/high-gain missions with humanoid robots. In such situations, one would expect the operator to be careful, avoid highly-dynamic motion, and keep the robot in static balance. Future work will extend the training set with more dynamic motions, like dynamic walking and high-speed movements, and with single-support motions.

References

[1] A. Goswami and P. Vadakkepat, *Humanoid Robotics: A Reference*, 1st ed. Springer Publishing Company, Incorporated, 2018.

[2] C. Atkeson *et al.*, "What happened at the DARPA robotics challenge finals," in *The DARPA Robotics Challenge Finals: Humanoid Robots To The Rescue*, 2018, pp. 667–684.

[3] J. Hollerbach, W. Khalil, and M. Gautier, *Model Identification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 321–344.

[4] N. Ramuzat *et al.*, "Actuator model, identification and differential dynamic programming for a talos humanoid robot," in *European Control Conference (ECC)*, 2020, pp. 724–730.

[5] J. Spitz *et al.*, "Trial-and-error learning of repulsors for humanoid QP-based whole-body control," in *IEEE Humanoids*, 2017.

[6] T. Anne, J. Wilkinson, and Z. Li, "Meta-learning for fast adaptive locomotion with uncertainties in environments and robot dynamics," in *IEEE/RSJ IROS*, 2021, pp. 4568–4575.

[7] R. Kaushik, T. Anne, and J. Mouret, "Fast online adaptation in robotics through meta-learning embeddings of simulated priors," in *IEEE/RSJ IROS*. IEEE, 2020, pp. 5269–5276.

[8] A. Cully, J. Clune, D. Tarapore, and J. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[9] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *ICLR*, 2019.

[10] K. Chatzilygeroudis and J.-B. Mouret, "Using parameterized black-box priors to scale up model-based policy search for robotics," in *IEEE ICRA*, 2018.

[11] K. Bouyarmane and A. Kheddar, "On weight-prioritized multitask control of humanoid robots," *IEEE Transactions on Automatic Control*, vol. 63, no. 6, pp. 1632–1647, 2017.

[12] E. Dalin *et al.*, "Whole-body teleoperation of the Talos humanoid robot: preliminary results," in *ICRA 2021 Workshop on Teleoperation*.

[13] O. Stasse *et al.*, "Talos: A new humanoid research platform targeted for industrial applications," in *IEEE Humanoids*, 2017.

[14] A. Kheddar *et al.*, "Humanoid robots in aircraft manufacturing: The Airbus use cases," *IEEE RA Magazine*, vol. 26, no. 4, pp. 30–45, 2019.

[15] M. P. Polverini *et al.*, "Agile actions with a centaur-type humanoid: A decoupled approach," in *IEEE ICRA*, 2021.

[16] C. Mastalli *et al.*, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *IEEE ICRA*, 2020.

[17] D. Cui *et al.*, "Human inspired fall arrest strategy for humanoid robots based on stiffness ellipsoid optimisation," *Bioinspiration & biomimetics*, vol. 16, 2021.

[18] V. Samy and A. Kheddar, "Falls control using posture reshaping and active compliance," in *IEEE-RAS Humanoids*, 2015.

[19] Q. Li *et al.*, "A minimized falling damage method for humanoid robots," *IJARS*, vol. 14, no. 5, 2017.

[20] D. Liu, Y. Lin, and V. Kapila, "A rollover strategy for wrist damage reduction in a forward falling humanoid," in *IEEE International Conference on Mechatronics and Automation (ICMA)*, 2021.

[21] V. Samy, K. Bouyarmane, and A. Kheddar, "QP-based adaptive-gains compliance control in humanoid falls," in *IEEE ICRA*, 2017.

[22] V. Samy, S. Caron, K. Bouyarmane, and A. Kheddar, "Post-impact adaptive compliance for humanoid falls using predictive control of a reduced model," in *IEEE-RAS Humanoids*, 2017.

[23] V. Kurtz, H. Li, P. M. Wensing, and H. Lin, "Mini cheetah, the falling cat: A case study in machine learning and trajectory optimization for robot acrobatics," *CoRR*, vol. abs/2109.04424, 2021.

[24] M. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, 1981.

[25] J. Tan, K. Liu, and G. Turk, "Stable proportional-derivative controllers," *IEEE Computer Graphics and Applications*, vol. 31, no. 4, 2011.

[26] J. Lee *et al.*, "Dart: Dynamic animation and robotics toolkit," *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[27] E. L. Dantec *et al.*, "Whole body model predictive control with a memory of motion: Experiments on a torque-controlled Talos," in *IEEE ICRA*, 2021.