In Rychtyckyj, Shapiro, editors, *Proceedings of the Twenty-second IAAI Conference on Artificial Intelligence*, pp. 1793--1798, AAAI, 2010.

# Predicting Falls of a Humanoid Robot through Machine Learning

**Shivaram Kalyanakrishnan**
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712, USA
shivaram@cs.utexas.edu

**Ambarish Goswami**
Honda Research Institute
Mountain View, CA 94043, USA
agoswami@honda-ri.com

**Abstract**

Although falls are undesirable in humanoid robots, they are also inevitable, especially as robots get deployed in physically interactive human environments. We consider the problem of *fall prediction*, i.e., to predict if a robot's balance controller can prevent a fall from the current state. A trigger from the fall predictor is used to switch the robot from a balance maintenance mode to a fall control mode. Hence, it is desirable for the fall predictor to signal imminent falls with sufficient lead time before the actual fall, while minimizing false alarms. Analytical techniques and intuitive rules fail to satisfy these competing objectives on a large robot that is subjected to strong disturbances and therefore exhibits complex dynamics.

Today effective supervised learning tools are available for finding patterns in high-dimensional data. Our paper contributes a novel approach to engineer fall data such that a supervised learning method can be exploited to achieve reliable prediction. Specifically, we introduce parameters to control the tradeoff between the false positive rate and lead time. Several parameter combinations yield solutions that improve both the false positive rate and the lead time of hand-coded solutions. Learned predictors are decision lists with typical depths of 5-10, in a 16-dimensional feature space. Experiments are carried out in simulation on an Asimo-like robot.

## 1. Introduction

As with their human counterparts, a majority of the activities undertaken by humanoid robots are performed from an upright posture. However, whereas maintaining such a posture is like a second nature to humans, the case with humanoid robots is opposite. Lacking the capability of balancing themselves in a robust manner, today's robots have to be shielded from falls through external support or monitored in controlled environments that involve little physical contact.

Several factors – unexpected external forces; power, component or communication failure; foot slippage – can threaten the balance of humanoid robots, especially as robots gain autonomy in realistic, possibly unforeseen environments. Falls are undesirable because they can cause catastrophic physical damage to the robot and its surroundings, which may also include people. Thus, fall is a severe failure mode that can be triggered in multiple ways. Coping with fall demands an integrated strategy that includes fall avoidance, prediction, and control.

Fall *avoidance* schemes attempt to reduce the incidence of fall. When fall does occur, methods to *control* the fall can potentially minimize damage to the robot or its surroundings. For instance, Fujiwara et al. (2007) consider optimizing the falling motion to minimize the impact force when the robot hits the ground. Yun et al. (2009) consider the problem of altering the direction of the fall to escape a possible collision with some object on the ground.

The focus of this paper is fall *prediction*, which is the critical component of the fall management strategy that decides when to switch from fall avoidance (or balance maintenance) to fall control. Specifically, our objective is to develop a fall predictor that continuously monitors the robot's state, and raises a flag as soon as it predicts an *imminent* fall (Figure 1). A trigger from the fall predictor prompts the robot to abandon the balance maintenance mode, which was just predicted to fail, and to execute a fall control mode.

While it is essential to predict fall early and maximize reaction time (the "lead time" to fall), it is also necessary to avoid *false* predictions of fall, which waste time by replacing the balance controller with a fall controller, sometimes thereby precipitating a fall. In other words, the fall predictor needs to minimize false positives. A key observation from our experiments is that in practice, trying to predict fall early typically leads to a high false positive rate, mainly because the system dynamics are quite complex. Thus, a good fall predictor must satisfactorily balance the conflicting objectives of high lead time and low false positive rate.

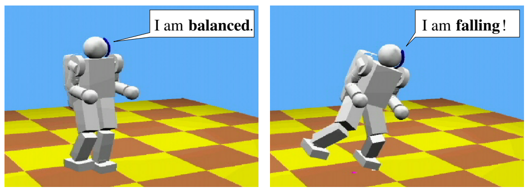


Figure 1: A fall predictor classifies the robot's current state as **balanced** or **falling**. This decision determines whether the robot should deploy balance control or fall control.

### 1.1 Why Machine Learning?

One may attempt to predict fall through intuitive rules such as thresholding relevant quantities (e.g. linear and angular momenta) or by tracking the robot's center of pressure (CoP), through which the resultant contact force between the robot and the ground acts. Alternatively, careful analytical modeling of a robot's dynamics could be used to predict if the robot's current trajectory will lead to a fall. For example, Renner and Behnke (2006) employ a model-based scheme for detecting fall on a kid-size humanoid robot (in simulation), in which deviation from the model's prediction is used to gauge whether the robot is falling. While they apply impulses of 0.15Ns to a robot weighing 2.3kg, we apply much larger impulses of up to 50Ns to a robot weighing 42.1kg.

Unfortunately, analytical methods do not scale well to large robots with complex geometries that are subjected to strong disturbances: the resulting dynamics is characterized by several degrees of freedom, variable friction, different contact configurations with one or both feet, and underactuation. Indeed Wieber (2008) concludes that while a "viability kernel" (the set of robot states from which the robot can escape fall by applying balance control) can be analytically derived for very simple control systems, for complex systems such as humanoid robots the computation is numerically intractable and often impossible. For illustration, consider the scenario depicted in Figure 2: starting from the same state, our robot is repeatedly subjected to external forces with different magnitudes. The force application point and direction are kept fixed. The figure shows the time taken for the robot to fall (if at all) for different magnitudes of applied force.

For force magnitudes less than $52N$, the balance controller is successful in preventing a fall. A magnitude of $54N$ causes the robot to fall after $3.2s$. As the force magnitude is increased, the corresponding time interval to fall is between $1.25s$ and $2.25s$. However, this trend does not continue: force magnitudes from $74N$ through $82N$ do not cause a fall! As the figure shows, there are multiple pockets of "fall" and "no fall" along the dimension of increasing force magnitude: there is no threshold below which fall is always avoided and above which fall always occurs. Interestingly some falls involve falling forwards, some backwards, and some sideways. Such non-monotonic patterns are also prevalent across state variables corresponding to center of mass (CoM) displacement, linear and angular momenta. Although the irregular nature of fall eludes precise analytical modeling, we hypothesize that a machine learning solution – driven by data – could cope better with the challenge.

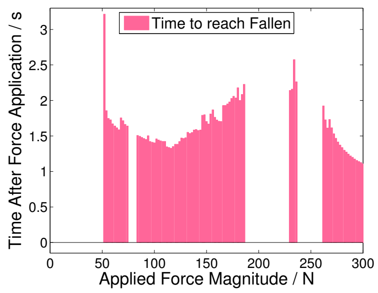


Figure 2: In this simulation experiment horizontal pushes of increasing magnitude are applied to an upright robot. The bars show the time taken for the robot to fall (as defined in Section 2.2). Gaps in the plot imply no fall.

### 1.2 Scope of Contribution

Supervised learning is today a mature discipline with the tools to reliably infer patterns among seemingly irregular data; in this paper we present a method for leveraging its strength to achieve effective fall prediction on large, high-DoF (degree of freedom) robots subjected to strong disturbances. If a robot's hardware undergoes wear and tear, or its controller changes, a learning algorithm can be re-run with little change on data gathered from the updated configuration of the robot. The manual effort required in so doing would be significantly less than that of a model-based solution, which would demand fresh calibration and revised modeling. Further, a machine learning-based solution provides a *reactive* strategy, under which predictions can be made almost instantaneously when deployed on a robot.

Ultimately fall is a problem faced by real robots. While we do not expect a fall predictor learned in simulation to register identical performance when deployed on a real robot, our results establish machine learning as a promising strategy that can be applied to data collected from a real robot. Indeed a solution learned off-line in simulation could initialize on-line learning on a real robot. We describe our learning process in Section 2. Results are presented in Section 3, which is followed by a discussion in Section 4.

## 2. Learning Process

The essential "learning" step in our solution is routine supervised learning. However, in order to successfully meet the specific demands of fall prediction, careful engineering is necessary in the preparation of the training data and the subsequent use of the learned classifier. In this section we enumerate the various stages in our learning process.

### 2.1 Generating Trajectories

We use the commercial robotics simulation software, Webots$^{TM}$ (Michel 2004), to simulate an Asimo-like robot with 26 degrees of freedom. The robot has a mass of $42.1kg$, with its CoM at a height of $0.59m$ above the ground. Each foot is $0.225m$ long and $0.157m$ wide. The robot's balance controller $C^{bal}$ implements the following strategy: if the linear momentum of the robot along either its frontal or its sagittal planes exceeds a fixed threshold, the robot widens its stance, thereby enlarging its support base and lowering its CoM. This strategy effectively thwarts falls, sometimes even when impulses of up to $40Ns$ are applied.

We obtain data for training the fall predictor by applying varying impulses to the robot at random instants of time in its walk cycle and recording the resulting trajectories. Each "push" comprises a constant force application for $0.1s$; the force magnitude is drawn uniformly randomly from the range $[0, 500N]$. The force is applied horizontally to the torso of the robot, at an angle with its sagittal plane drawn uniformly randomly from $[-180°, 180°]$, at a height above its CoM drawn uniformly randomly from $[-0.05m, 0.25m]$. These ranges are chosen such that roughly half the trajectories (out of a total of 1000) result in fall. We hypothesize that the resulting states will be similar to the states arising when the robot is subjected to more realistic disturbances.

## 2.2 Labels

We partition the state space of the robot into three classes: **balanced**, **falling** and **fallen**. Any state reached by the robot along its trajectories belongs to one of these classes, which are depicted schematically in Figure 3. The **fallen** class (most peripheral) comprises states which satisfy some rule to identify a fallen robot, such as whether parts of the robot's body other than its feet are in contact with the ground, or its CoM falls below some threshold height (set to $0.33m$ in our experiments to determine **fallen**). The **balanced** class (most interior) comprises states from which applying a balance controller $C^{bal}$ will not lead to a **fallen** configuration when the only forces acting on the robot are its weight $W$, the resultant ground reaction force $R$, and friction $F_{fr}$. For a given robot the shape and size of the **balanced** class is specific to a balance controller: likely, a "better" $C^{bal}$ will enjoy a larger **balanced** class. Intermediate states that are neither **balanced** nor **fallen** are designated as **falling**: trajectories passing through **falling** necessarily terminate in **fallen** under $C^{bal}$.

In realistic settings the robot will come under the influence of external forces $F_{ext}$, which we designate to be forces other than $W$, $R$, and $F_{fr}$. Likely sources of external forces are contact forces between the robot and obstacles, slippage and drag. As shown in Figure 3 external forces are *necessary* for reaching a **falling** state starting from a **balanced** state, although $C^{bal}$ may succeed in retaining the robot within **balanced** in some cases even with $F_{ext}$ acting. In the trajectories we have generated, we have the benefit of hindsight, once a trajectory has entered **fallen**, to label states leading up to it (but after our push – an external force – terminated) **falling**. Trajectories *not* ending in a **fallen** state have all their states labeled **balanced**.

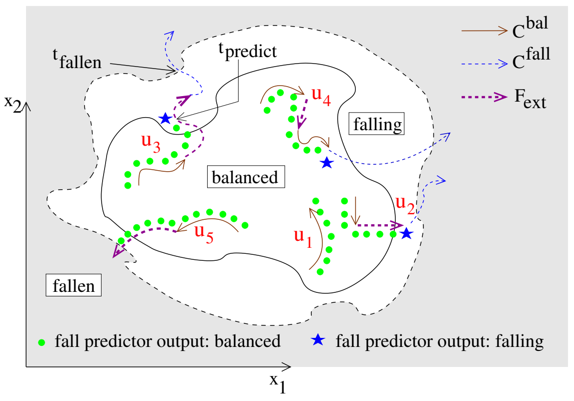


Figure 3: The robot's feature space is partitioned into **balanced**, **falling** and **fallen** classes. In general the classes might not occupy contiguous regions of the feature space. Trajectories $u_1$ - $u_5$ are described in Section 2.3.

## 2.3 Objectives of Fall Prediction

Indeed our purpose is to generalize based on the training data to construct a fall predictor that maps every robot state (represented as a vector of features, described in Section 2.5) into **balanced** or **falling**. Armed with such a predictor, the robot's policy is to apply $C^{bal}$ as long as **balanced** is predicted, and to switch to a fall controller $C^{fall}$ if **falling** is predicted. A learned fall predictor is bound to be imprecise. While no mispredictions occur along trajectory $u_1$ in Figure 3, along both trajectories $u_2$ and $u_3$, **balanced** is incorrectly predicted even after **falling** is reached. Trajectory $u_4$ corresponds to a false positive and $u_5$ to a false negative.

False negatives can be avoided easily by adding a rule to predict **falling** if the CoM drops below some vertical height threshold (0.48m for our robot). In contrast, false positives are difficult to avoid, especially if the fall predictor has to make early predictions of **falling**. We define the **False Positive Rate** (**FPR**) of a fall predictor to be the fraction of trajectories in which **falling** is predicted for a **balanced** state. Since each such incorrect prediction prompts an unnecessary invocation of $C^{fall}$, FPR needs to be minimized.

Trajectory $u_3$ in Figure 3 is annotated with the instants of time at which **falling** is predicted ($t_{predict}$) and **fallen** is reached ($t_{fallen}$). The interval between these instants is the time duration for which $C^{fall}$ acts to minimize the damage due to the fall. We define the **Lead Time** ($\boldsymbol{\tau_{\text{lead}}}$) of a fall predictor to be the average value of $t^u_{fallen} - t^u_{predict}$ over trajectories that terminate in **fallen**, assuming $C^{fall}$ is deployed from $t^u_{predict}$ onwards. Larger values of $\tau_{lead}$ imply that $C^{fall}$ gets more time on average to respond to a fall; thus $\tau_{lead}$ is a quantity to be maximized.

We see that the fall predictor with the lowest FPR (zero) is one that predicts **balanced** for every input state; unfortunately, such a predictor also has the lowest value of $\tau_{lead}$ (zero). At the opposite extreme, a predictor that always predicts **falling** has maximal $\tau_{lead}$, but correspondingly, an FPR of 100%. Neither extreme is practical; we desire a fall predictor that enjoys a low FPR and a high value of $\tau_{lead}$.

## 2.4 Parameters to Control FPR and $\boldsymbol{\tau_{\text{lead}}}$

The process of preparing training data for the supervised learning algorithm is a crucial aspect of our solution. Consider a fall predictor that has a prediction accuracy of 99% over all **balanced** states. Such a predictor could still suffer very high FPR if its few incorrect predictions – of predicting **falling** instead of **balanced** – are distributed over a large number of **balanced** trajectories, rather than contained to a few. At the other extreme, a fall predictor that has a low accuracy in identifying **falling** states correctly might still give rise to a high value of $\tau_{lead}$ if its correct predictions occur early in the **falling** trajectories, since once **falling** is predicted along a trajectory, subsequent predictions are immaterial. In short, a predictor with a higher prediction accuracy over all the recorded states does not necessarily enjoy lower FPR and higher $\tau_{lead}$. We describe two techniques that we employ to explicitly promote the learning of fall predictors that minimize FPR and maximize $\tau_{lead}$.

Consider a part of a trajectory that is within the **falling** class. States occurring early in this trajectory are likely to be less distinguishable from **balanced** states when compared to states occurring later in the trajectory. Indeed we observe that if states that occur early along the **falling** trajectory are presented as training data to the learning algorithm, then the learned fall predictor is likely to incur higher FPR. In contrast, since a **falling** trajectory will end in a **fallen** state, states close to this extreme can be separated quite easily, such as by a rule that thresholds the height of the CoM.

Figure 4 schematically depicts for one **balanced** and one **falling** trajectory the height of the CoM as a function of the time elapsed after the application of the impulse ($t_{force-end}$). In principle all the states in the **falling** trajectory are valid training examples for the **falling** class, just as all the states in the **balanced** trajectory are valid training examples of **balanced**. However, to reduce the incidence of false positives, we *withhold* from the set of positive (**falling**) training data states that occur early along **falling** trajectories. Only those positive examples that occur after a "cutoff" time are used for training. Since different **falling** trajectories have different time durations, we standardize this cutoff time by measuring it with respect to the instant $t_{height-drop}$, which is the point of time at which the height of the CoM above the ground begins to drop monotonically until a **fallen** trajectory is reached.

We define a parameter $\tau_+$, such that only **falling** states that occur beyond the instant $t_{height-drop} + \tau_+$ and before $t_{fallen}$, the time at which the trajectory enters the **fallen** class, are used as positive training instances for supervised learning. We expect that as $\tau_+$ is increased, the learned predictor will have lower FPR, but also a lower value of $\tau_{lead}$. Decreasing $\tau_+$ (note that $\tau_+$ can be negative) will likely increase both $\tau_{lead}$ and FPR. We still use all the available negative (**balanced**) examples for training; best results are achieved by weighting **balanced** instances 4 times as high as **falling** instances in the training data.

In attempting to identify a working range for $\tau_+$ based on evaluating its relationship with FPR and $\tau_{lead}$, we discover the need to formulate a second parameter, $\tau_{his}$, to also play a role in determining this relationship. In principle, the robot could switch from $C^{bal}$ to $C^{fall}$ as soon as the predictor classifies the current state as **falling**. However, this would make the control policy brittle, over-reactive and often incorrect, with a single false positive causing an unnecessary deployment of $C^{fall}$. Such an event is avoided by maintaining a finite history of the atomic predictions made by the learned classifier, and only predicting **falling** when the list has consistently predicted **falling** over all states in the history window. Figure 5 depicts this adaptation, which also has the effect of decreasing FPR.

The parameter $\tau_{his}$ corresponds to the temporal length of the history window that is maintained. Whereas $\tau_+$ is used in generating the data for training, $\tau_{his}$ only comes into play *after* the predictor has been learned. A positive quantity, $\tau_{his}$ effectively smooths out predictions, weeding out stray, short-lived predictions of **falling**. In so doing, it also has the effect of delaying correct predictions of **falling**, thereby decreasing $\tau_{lead}$. Together, $\tau_+$ and $\tau_{his}$ provide handles to control the tradeoff between FPR and $\tau_{lead}$: they can be provided by the user as inputs to the learning algorithm.

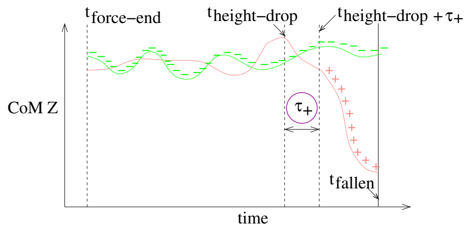


Figure 4: While *preparing* the training data, trajectories in the **falling** class are sampled based on the parameter $\tau_+$. The instant $t_{height-drop}$ is when the height of the CoM begins to monotonically decrease until reaching the **fallen** class (at time $t_{fallen}$). Points from **falling** (marked "+") are sampled in the interval $[t_{height-drop} + \tau_+, t_{fallen}]$.

## 2.5 Features

Feature engineering can make a significant difference in the performance of a learning algorithm. It is desirable for the chosen set of features to be small, to provide all the information necessary to classify a state, while at the same time able to generalize well to nearby states. For our task, we arrive at a set of 16 features through successive refinement, trial and error. Of these, 15 are real-valued numbers corresponding to five physical quantities: CoM displacement, linear momentum and its time derivative, angular momentum about the CoM and its time derivative. We find it best to reference the chosen vectors to a Cartesian coordinate system located at the robot's CoP, with the $x$ and $y$ axes along the ground in the robot's sagittal and frontal planes respectively, and the $z$ axis vertical. Note that the CoP predominantly resides at an edge or vertex of the robot's support base when a fall occurs. Correspondingly, an additional discrete feature we consider is the robot's "foot contact mode," which describes the position of the CoP relative to the robot's feet.

Every state of the robot maps to a foot contact mode, which identifies whether the left and right feet are touching the ground, and if they are, the position of the CoP within the support base. We consider three modes when both feet are touching the ground: LR-INSIDE, LR-FRONT, and LR-BACK. In LR-INSIDE, the CoP lies inside the support base, while in LR-FRONT and LR-BACK, it lies at the front or back edge, respectively. Other modes for single support (both left and right) are defined similarly, and one mode describes the case in which neither foot touches the ground. In total we define 16 foot contact modes; as a result, our feature

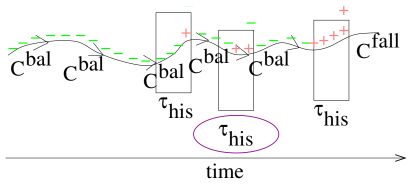


Figure 5: While *using* a fall predictor, a history of the predictions made in the past duration of $\tau_{his}$ is maintained. At time $t$, **falling** is predicted only if *all* individual predictions made in the interval $[t - \tau_{his}, t]$ are **falling**.

vector comprises 16 variables: 15 real-valued variables and one discrete variable that takes 16 values.

Although techniques such as referencing vectors with respect to the CoP result in more discriminative features, we note that a targeted study involving aspects such as the geometry, inertial properties, and controllers used by a specific robot could likely yield more substantial improvements in fall prediction. For example, Höhn and Gerth (2009) find the foot tilting angle and velocity to be effective features on the BARt-UH robot.

## 2.6 Learned Representation

Having prepared sets of positive (**falling**) and negative (**balanced**) training instances (16-dimensional), we experiment with several supervised learning methods from the WEKA machine learning library (Hall et al. 2009). We obtain the best results for our task with rule-based systems such as decision trees and lists, which marginally outperform regression-based approaches such as neural networks and radial basis functions. The trends identified in Section 2.4 apply to all these methods; we posit that the slight difference in their performance is only a consequence of the extent to which these qualitatively different approaches require tuning. We adopt decision list learning (or "rule learning") for our experiments. Rule-based systems find applications in a variety of applications (Langley and Simon 1995). Like decision trees, decision lists are grown by splitting nodes recursively, guided by a heuristic such as information gain (Hall et al. 2009). We observe that better predictions are made when a separate decision list is learned for each foot contact mode, rather than when a single decision list is learned, in which foot contact mode is used as a feature to split nodes. This observation suggests that foot contact modes separate the robot's states into homogeneous regions where decision boundaries are more regular.

## 3. Results

We run 10-fold cross-validation tests on a set of 1000 trajectories generated as described in Section 2.1. Every combination of $\tau_+$ and $\tau_{his}$ yields a separate fall predictor, which registers different values of FPR and $\tau_{lead}$. From Figure 6(a), which is similar to a receiver operating characteristic (ROC) curve for classification problems, it is clear that decreasing FPR in a learned predictor comes at the cost of decreasing $\tau_{lead}$. The dependence of FPR and $\tau_{lead}$ on the parameters of the learning algorithm, $\tau_+$ and $\tau_{his}$, is shown in Figures 6(b) and 6(c). In keeping with intuition, we find that lower (higher) values of $\tau_+$ and $\tau_{his}$ increase (decrease) FPR and $\tau_{lead}$. Note that if we did not withhold any positive instances ($\tau_+ \approx -1.5s$) or use a history window ($\tau_{his} = 0$), FPR would be nearly 100%; $\tau_+$ and $\tau_{his}$ are *necessary* to deliver the benefits of supervised learning.

Table 1 summarizes performance statistics of three representative fall predictors illustrating that FPR and $\tau_{lead}$ can be traded off by adjusting training parameters $\tau_+$ and $\tau_{his}$. $FP^{L1}$ is conservative, with high $\tau_{lead}$ but high FPR. At the other extreme, $FP^{L3}$ has near-zero FPR, but less than half the $\tau_{lead}$ value of $FP^{L1}$. In between, $FP^{L2}$ enjoys a relatively low value of FPR, but also a reasonably high value of $\tau_{lead}$. Indeed $FP^{L2}$ and several other learned predictors improve both the FPR and $\tau_{lead}$ values of manually designed solutions for fall prediction. The best tradeoff we could obtain in a handcrafted solution (by thresholding both X and Y components of the linear momentum) is an FPR of 0.11 with corresponding $\tau_{lead}$ of $0.60s$. While the results in Table 1 are based on training with 900 trajectories (cross-validated), we find that roughly 250 trajectories suffice for obtaining FPR $< 10\%$ and $\tau_{lead} >$ 0.70s.

Table 1: Comparison of three representative fall predictors.

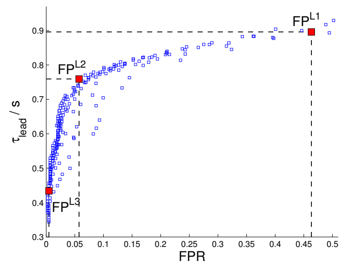
| Fall predictor: | $FP^{L1}$ | $FP^{L2}$ | $FP^{L3}$ |
|---|---|---|---|
| $\tau_+$ / s | -0.3 | -0.25 | 0.3 |
| $\tau_{his}$ / s | 0.016 | 0.060 | 0.044 |
| FPR | 0.46 | 0.06 | 0.005 |
| $\tau_{lead}$ / s | 0.90 | 0.76 | 0.43 |
| Mean Rule Size | 61.6 | 51.4 | 5.3 |

We define the "mean rule size" of a predictor as the average number of comparison operators in its 16 component decision lists. From the last row of Table 1, we notice that with low values of the $\tau_+$ input parameter, $FP^{L1}$ and $FP^{L2}$ have fairly high mean rule sizes, indicating that finer distinctions are learned as we train on states early along **falling** trajectories. The "depths" of the learned lists are typically between 5 and 10. In comparison, we expect typical hand-coded rules for fall prediction to contain no more than 4-6 comparisons, over depths of 2–3. Visualizing projections of trajectories on axes corresponding to different features, we see no clear separation between the classes. Figure 6(d) revisits the illustrative example from Figure 2, and affirms the expected prediction trend among $FP^{L1}$, $FP^{L2}$, and $FP^{L3}$.

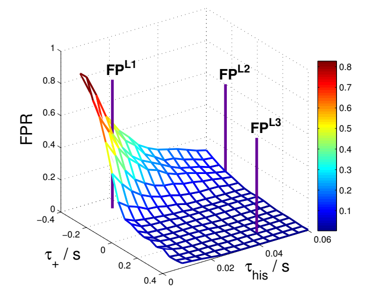
## 4. Summary and Discussion

A humanoid robot cannot completely avoid falling, and so it requires a concerted strategy for controlling a fall when it occurs. The precursor to fall control is fall prediction, which is the subject of this paper. In particular we are concerned with predicting fall when a large robot is subjected to relatively strong disturbances, and the resulting dynamics can be quite complex. We adopt a machine learning approach for the purpose. Rather than our choice of supervised learning method – decision lists – the more general contribution of our work is a method to control the tradeoff between FPR and $\tau_{lead}$, which are fundamental desiderata to the problem of fall prediction. By discarding positive training instances on the basis of the parameter $\tau_+$, we commit to reduce the incidence of false positives, even if it reduces the lead time to fall. Likewise $\tau_{his}$ is used to weed out stray false positives by averaging over a time window, which again has the compensatory effect of reducing $\tau_{lead}$. An important direction for future work is to consider learning methods such as Hidden Markov Models, which are naturally equipped to make predictions over temporal sequences. Höhn and Gerth (2009) implement a related approach for fall prediction and control on a simulated version of a smaller humanoid robot.

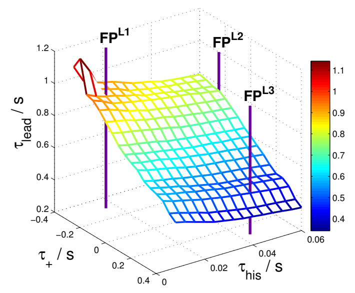
Our results demonstrate that a machine learning-based approach can significantly improve both the FPR and $\tau_{lead}$ values of hand-coded solutions for fall prediction. A learning



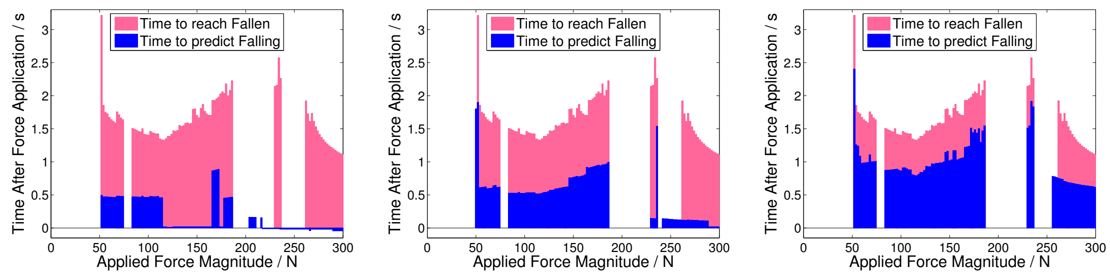
(a)
$\mathbf{FP^{L1}}$



(b)
$\mathbf{FP^{L2}}$



(c)
$\mathbf{FP^{L3}}$



(d)

Figure 6: (a) A range of fall predictors achieving different tradeoffs between FPR and $\tau_{lead}$. (b) FPR and (c) $\tau_{lead}$ values shown as a function of the *inputs* to the learning process: $\tau_+$ and $\tau_{his}$. Three representative predictors, $FP^{L1}$, $FP^{L2}$ and $FP^{L3}$, showcase the conflict between FPR and $\tau_{lead}$. (d) The predictors are compared on the example discussed in Section 1 (Figure 2): one set of bars (taller, background) mark the time after force application to reach a **fallen** state (if at all), while another set of bars (shorter, foreground) show the time elapsed for predicting **falling** (if at all). Foreground bars which do not overlap with background bars represent false positives.

approach indeed appears promising for fall prediction on a real robot, too. The pushing regimen described in our work would be tedious if a human agent is involved in the generation of each trajectory on the robot, but it is conceivable to design an automated apparatus to push the robot and arrest its falls. The robot could then generate data autonomously through a repetitive process. A similar strategy is adopted by Kohl and Stone (2004) to optimize the forward walking speed of four-legged Aibo robots.

In this early work in the area of adaptive fall prediction, we generate and process data as a batch, leveraging the strength of established supervised learning methods to surmount the highly irregular robot dynamics. For improved autonomy, it would be necessary for humanoid robots to learn with less supervision in an on-line, incremental manner as they encounter new situations.

## Acknowledgements

Shivaram Kalyanakrishnan was supported by the summer internship program at the Honda Research Institute, USA. We thank the anonymous reviewers of the current and earlier versions of this paper for providing useful comments.

## References

Fujiwara, K.; Kajita, S.; Harada, K.; Kaneko, K.; Morisawa, M.; Kanehiro, F.; Nakaoka, S.; and Hirukawa, H. 2007. An optimal planning of falling motions of a humanoid robot. In *Proc. IROS 2007*, 456–462. IEEE.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The WEKA data mining software: An update. *SIGKDD Explorations* 11(1):10–18.

Höhn, O., and Gerth, W. 2009. Probabilistic balance monitoring for bipedal robots. *International Journal of Robotics Research* 28(2):245–256.

Kohl, N., and Stone, P. 2004. Machine learning for fast quadrupedal locomotion. In *Proc. AAAI 2004*, 611–616. AAAI Press.

Langley, P., and Simon, H. A. 1995. Applications of machine learning and rule induction. *Communications of the ACM* 38(11):54–64.

Michel, O. 2004. Webots$^{TM}$: Professional mobile robot simulation. *Journal of Advanced Robotics Systems* 1(1):39–42.

Renner, R., and Behnke, S. 2006. Instability detection and fall avoidance for a humanoid using attitude sensors and reflexes. In *Proc. IROS 2006*, 2967–2973. IEEE.

Wieber, P.-B. 2008. Viability and predictive control for safe locomotion. In *Proc. IROS 2008*, 1103–1108. IEEE.

Yun, S.-k.; Goswami, A.; and Sakagami, Y. 2009. Safe fall: Humanoid robot fall direction change through intelligent stepping and inertia shaping. In *Proc. ICRA 2009*, 781–787. IEEE.