In *International Journal of Humanoid Robotics*,
8(2):245--273, 2011.

International Journal of Humanoid Robotics

LEARNING TO PREDICT HUMANOID FALL

SHIVARAM KALYANAKRISHNAN
Department of Computer Science, The University of Texas at Austin
Austin TX 78701 USA
shivaram@cs.utexas.edu

AMBARISH GOSWAMI
Honda Research Institute
Mountain View CA 94043 USA
agoswami@honda-ri.com

Received Day Month Year
Revised Day Month Year
Accepted Day Month Year

Falls are undesirable in humanoid robots, but also inevitable, especially as robots get deployed in physically interactive human environments. We consider the problem of *fall prediction*: to predict if the balance controller of a robot can prevent a fall from the robot's current state. A trigger from the fall predictor is used to switch the robot from a balance maintenance mode to a fall control mode. 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 exhibits complex dynamics. We contribute a novel approach to engineer fall data such that existing supervised learning methods can be exploited to achieve reliable prediction. Our method provides parameters to control the tradeoff between the false positive rate and lead time. Several combinations of parameters yield solutions that improve both the false positive rate and the lead time of hand-coded solutions. Learned solutions 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.

Keywords: Humanoid robots; fall prediction; supervised learning; machine learning.

1. Introduction

Fall is a severe failure mode for humanoid robots, which can be triggered by several factors, including unexpected external forces; power, component or communication failures; and foot slippage. Falls are undesirable not only because they signal abrupt breaks in a robot's normal activities, but also because they can cause catastrophic physical damage to the robot and its surroundings, which may also include people. Robots can be shielded from falls through external support or by being contained within controlled environments that involve very little physical contact. However, as humanoid robots gain autonomy and enter more realistic, possibly unforeseen

environments, fall will be an inevitable occurrence. Coping with falls requires an integrated strategy that incorporates fall avoidance, prediction and control.

1.1. *Fall Avoidance and Control*

Fall *avoidance* deals with developing better strategies for robots to reduce the incidence of fall. Numerous techniques have been proposed in the literature to safeguard a robot's balance. For example, Kagami *et al.* propose a balance compensation scheme.[1] Nishio *et al.* use the "reaction null space" to derive a set of feasible control laws.[2] Controlling the position of the Zero Moment Point (ZMP) of a robot is useful in averting potential falls;[3] correspondingly Ogata *et al.* devise a stepping motion in the predicted direction of the ZMP to extend the robot's support base and maintain balance.[4] Researchers have also proposed other stepping motions and reflex actions for fall avoidance.[5–8] However, any avoidance scheme can only reduce, and not eliminate, the incidence of fall. When fall does occur, fall *control* methods can mitigate its adverse effects. In so doing, such methods address two primary goals:[a] (i) self-damage minimization and (ii) fall direction change.

Uncontrolled fall of a robot can result in high-impact collisions with surrounding objects and the ground. Fujiwara *et al.* propose "UKEMI" motion to control a robot's fall such that its impact points with the ground are in the hip, knees, and elbows, which can all be protected with shock absorbing mechanisms.[10] Subsequent work proposes a parameterized planning strategy for falling that can be optimized to reduce the impact forces at collision and to increase stability after landing.[11,12] Ruiz-del-Solar *et al.* apply a similar optimization-based method in simulation, wherein a human designer participates in setting joint angles at key frames in the fall sequence.[13]

As robots enter environments with stationary and moving objects (including people), preventing damage to surrounding objects during a fall could be more important that minimizing self-damage. Our earlier work in fall control shows that the direction of a robot's fall can be changed significantly through calculated stepping and inertia shaping.[14,15] As a result, collisions with objects on the ground can be avoided. In general there can be mixed objectives during fall control, such as to first avoid hitting a surrounding object during fall, and then to minimize self-damage.

1.2. *Fall Prediction*

The success of a fall control strategy, whether its objective is to minimize self-damage or damage to surrounding objects, depends critically on the time available

[a]Throughout this paper we consider only accidental falls which result from unexpected disturbances or system malfunction. By contrast, a robot may perform an *intentional* fall, which is a strategic action, such as in the case of a robot soccer goalie diving to stop a ball.[9] In an intentional fall, the robot typically has a longer time to plan, and can significantly minimize fall-related damage when compared to an accidental fall.

for the fall controller to act. Thus, an important prerequisite for effective fall control is early and reliable fall *prediction*, which is the subject of this paper. Specifically our objective is to develop a "fall predictor" that continuously monitors the state[b] of the robot, and raises a flag as soon as it detects an *imminent* fall. 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 strategy. Figure 1 pictorially depicts the function of a fall predictor.

While it is essential to predict fall early and maximize 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, and also sometimes precipitate 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 trade off the conflicting objectives of high lead time and low false positive rate.

1.3. *Why Machine Learning?*

One may attempt to predict fall through intuitive rules such as thresholding relevant physical 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. Effective analytical approaches indeed offer greater flexibility in dynamic settings such as when the robot carries a known mass. Unfortunately, manually designed rules and analytical methods do not scale well to large robots with complex

[b]We use the term "state" to generically imply all the variables that describe a system, and not its specific technical meaning as the minimal variable set.

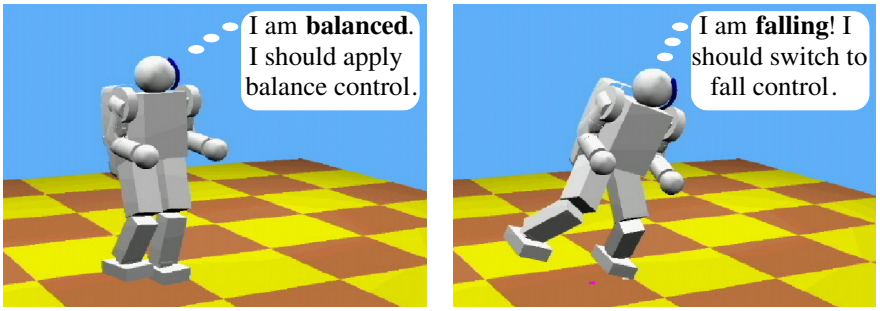


Fig. 1. Screenshots of a simulated ASIMO-like robot. A fall predictor classifies the current state of the robot as balanced or falling to determine whether balance control or fall control should be applied.

geometries that are subjected to strong disturbances: the resulting dynamics is characterized by several degrees of freedom (our robot has 26), variable friction, different contact configurations with one or both feet, slippage and underactuation. 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-intuitive 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. Further, 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 entail fresh calibration and revised modeling. Also note that a machine learning-

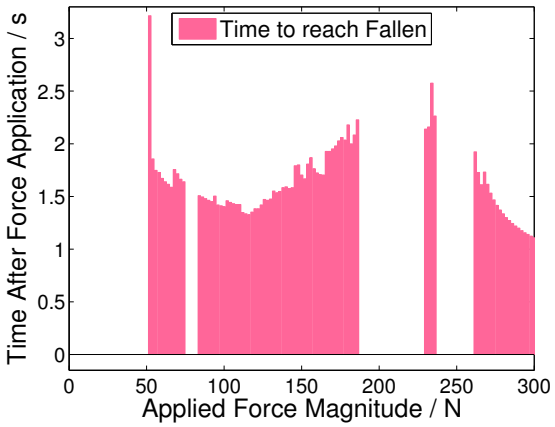


Fig. 2. In this experiment, sideways horizontal pushes of increasing magnitude are applied to an upright humanoid robot from a *fixed* state in its walk cycle. Bars mark the time taken for the robot to fall (as defined in Section 3). Gaps in the plot imply no fall. The fall pattern is non-monotonic.

based solution results in a *reactive* strategy, under which predictions can be made almost instantaneously when deployed on a robot.

1.4. *Contribution and Scope*

Supervised learning provides tools to infer patterns among seemingly irregular data. The main contribution of this paper is a principled approach to train a fall predictor using supervised learning. Specifically we expect our contribution to be relevant in cases like the one illustrated when large, high-DoF (degree of freedom) robots are subjected to strong disturbances. Our solution complements existing model-based strategies that perform well on simpler robots with relatively smaller disturbances.

We conduct simulation experiments on an ASIMO-like robot.[16] First, we collect a batch of "balanced" and "falling" trajectories, which are then processed off-line to learn a fall predictor. Once learned, the fall predictor is used *on-line*, making predictions in real time. Our learning algorithm is designed to accept parameters to control the tradeoff between the early prediction of fall and the minimization of false alarms. Consequently we obtain a family of fall predictors, ranging from ones that signal fall early but raise several false alarms, to ones that provide a smaller reaction time but make reliable predictions of fall. Cross-validation testing shows that several parameter values yield fall predictors that can detect fall earlier, yet raise fewer false alarms than our best manually designed predictors.

Our proposed architecture has a number of replaceable modules. We use a simple balance control mechanism and a standard supervised learning algorithm. Our experiments show that the fall predictors learned are not very sensitive to the supervised learning algorithm used, but currently we do not undertake detailed sensitivity testing with respect to variations in the robot's specifications and its balance controller. These components could be changed appropriately for learning a fall predictor on a different robot. It would be equally important to consider the features used for prediction, which should ideally make the best use of the sensors and measurements available on a given robot.

Ultimately fall is a problem faced by real robots. We do not expect a fall predictor learned in simulation to register identical performance when deployed on a real robot, as our simulation does not incorporate aspects such as action noise, variable friction and complex foot geometry. However, our results show the promise of applying machine learning to data collected from a real robot, and are to be treated as a starting point. A solution learned off-line in simulation could initialize on-line learning on a real robot.

The rest of this paper is organized as follows. After surveying related literature in Section 2, we present a framework for defining fall and evaluating a fall predictor in Section 3. In Section 4 we inspect data obtained from a simulated version of an ASIMO-like humanoid robot, which reveals an irregular falling pattern. In Section 5 we describe our machine learning architecture. Detailed results are presented in Section 6. We conclude with a discussion in Section 7.

2. Related Work

In this section we discuss related work in humanoid fall prediction (Section 2.1), and briefly highlight connections to machine learning (Section 2.2), *human* fall research in the biomechanics literature (Section 2.3), and the general problem of failure warning (Section 2.4).

2.1. *Humanoid Fall Prediction*

Wieber defines the "viability kernel" for a robot and balance controller as the set of states from which the robot can escape fall by applying the balance controller.[17,18] He concludes that for reasonably complex dynamical systems such as bipedal humanoids, it is numerically intractable and often impossible to compute a viability kernel of states. In subsequent work Wieber derives lower bounds for the viability kernel in simpler systems such as a cart pole and an inverted pendulum.[19] The absence of analytical bounds for large, multi-DoF humanoid robots motivates our choice of an empirical approach to predict their fall.

Renner and Behnke employ a model-based scheme for predicting fall on a kid-size humanoid robot (in simulation).[20] A model of the gait, represented using Fourier coefficients and attitude sensor values, is used to predict future states, given the current state and controls. Disturbances are detected by thresholding the deviation of the robot's state from the model's prediction, and actions are taken depending on the magnitude of the deviation. A similar strategy based on deviation from the model's prediction is used by Karssen and Wisse, who conduct experiments on the 6-DoF 'Meta' robot.[21] In our experiments we subject the robot to relatively large disturbances: whereas Renner and Behnke apply impulses of 0.15Ns to a robot weighing 2.3kg, we apply impulses of up to 50Ns to a robot weighing 42.1kg.

Machine learning-based solutions result in direct mappings between states and (probability distributions over) classes. While we use decision lists for classifying states as "balanced" or "falling", Höhn and Gerth conflate fall prediction and control, assigning as classes "reflex" motions such as crouching or stretching in some direction.[22] The predictors learned are Gaussian Mixture Models and Hidden Markov Models, which are applied to a simulated version of the 'BARt' robot. Among the features used by Höhn and Gerth to represent states are force sensors in the feet, inertial measurements of the torso, attitude and rotational velocities of the stance foot, and the time instant during the robot's gait that the disturbance occurs. In an integrated fall management strategy Ogata *et al.* implement a predictor that thresholds the Euclidean distance between the robot's sensory readings with their time-averaged mean.[23] Discriminant analysis is used to improve the quality of the predictions. In subsequent work Ogata *et al.* use feedback from ZMP control to predict fall.[4]

2.2. *Machine Learning*

Our learning architecture admits any supervised learning algorithm and corresponding representation for the fall predictor. Of the several variants we test empirically, we obtain the best results with rule-based systems such as decision trees and lists. Rule-based systems find application in a variety of applications, and are attractive for their easy interpretability.[24] Whereas we use decision lists to *predict* imminent fall, a failure mode, Chen *et al.* apply decision trees to *diagnose* failure once it has occurred in large computer networks.[25] For two-class supervised learning problems, metrics such as the F-measure and area under the receiver operating characteristic (ROC) curve have been proposed to evaluate the tradeoff between the correct prediction of positive and negative instances.[26] For our fall prediction problem we formalize two relevant metrics: false positive rate and lead time.

2.3. *Biomechanics of Human Fall*

Age, sickness and weakness make human beings susceptible to fall. A fallen individual is often unable to recover immediately. To provide a remedy, one line of research in biomechanics addresses the design of wearable devices, such as wristwatches, which trigger alarms when the wearer falls.[27] As with humanoid fall prediction, it becomes necessary to trade off the accuracy of detecting human falls with the false alarm rate.[28,29] The chief difference between fall prediction on a robot and fall detection in humans is that the former *anticipates an imminent fall* on-line, while the latter's objective is to reliably detect fall *once it has occurred*.

2.4. *Failure Warning Systems*

Humanoid fall prediction may be viewed as an instance of the more general problem of failure warning, in which an alarm must be raised when a system enters, or becomes destined to enter, a failure state. In an early survey of methods used for failure detection, Gertler focuses on methods in which a model of the system is available.[30] To test for a failure, a "residual" difference between analytically predicted and observed values of quantities are subjected to statistical testing to estimate if the deviations are due to noise or due to systematic failure. Mattone and De Luca use residuals to detect sensor and actuator faults on a robot manipulator, and indeed this paradigm is closely mirrored in the approach adopted by Renner and Behnke for fall prediction.[20,31] In a starkly contrasting application, Zou *et al.* model and forecast the spread of an Internet worm using a Kalman filter.[32] Vehicular collision warning is another area of active research.[33,34] Reliable failure is also critical in domains that operate at longer time scales, such as tsunami warning systems (minutes/hours) and bank failure prediction (days/months).[35,36]

3. Problem Description

In this section we provide precise descriptions of fall and fall prediction, from which we derive quantitative evaluation metrics.

3.1. *State Classes*

Consider a multi-dimensional feature space (state space) containing all the potentially useful variables that can be measured or computed from the robot's sensors, such as its joint angles and velocities, inertia and momentum. With time the robot traces a trajectory in this space, which we partition 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)

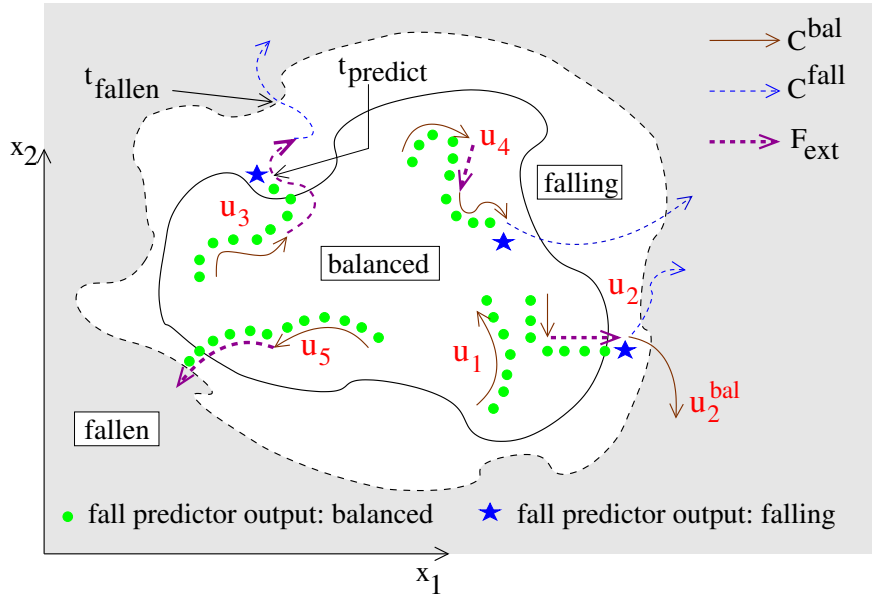


Fig. 3. Conceptual diagram showing the multi-dimensional feature space describing the configuration of the robot, partitioned into balanced, falling and fallen classes. In general the classes need not occupy contiguous regions of the feature space. External forces are necessary to take the robot from balanced to falling. We assume that from a falling state, a trajectory that results from applying a fall controller (C^{fall}) is preferred over a trajectory that results from applying the balance controller (C^{bal}). At every decision cycle, a fall predictor maps the robot's current state to either balanced (solid dot) or falling (star). The robot switches from C^{bal} to C^{fall} as soon as falling is predicted. Descriptions of trajectories u_1 through u_5 are provided in Sections 3.2 and 3.3.

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. In addition we find it convenient to include as F_{ext} all other events that may trigger fall, such as motor failure and random disturbances. 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. When external forces do cause trajectories inside balanced to reach falling, the robot is certain to reach a fallen state if it continues to apply C^{bal} and there are no further external forces to oppose the fall.

3.2. *Fall Predictor*

To mitigate the adverse effects of a fall, the robot invokes its fall controller C^{fall} as soon as it believes to be in a falling state. Not knowing the true class of its state, the robot uses a fall predictor to estimate the class. Figure 4 presents a schematic view of a fall predictor. At every decision cycle the predictor can employ any of the measured or computed variables in the feature space. The predictor maps every such input feature vector to either balanced or falling, determining which controller is to be applied. In Figure 3 we mark five trajectories — u_1 through u_5 — to illustrate typical scenarios arising from the use of a fall predictor. At every state encountered along a trajectory, either balanced (shown as solid dot) or falling (shown as star) is predicted; correspondingly the robot applies C^{bal} or C^{fall}.

The trajectory u_1 resides completely inside balanced, and the fall predictor correctly predicts balanced all along the trajectory. However, in general we do not expect the predictions to all be correct. Trajectories u_2 and u_3 are likely scenarios during fall prediction, in which balanced is predicted inside the balanced class, but continues to be predicted (incorrectly) for some time even after falling is reached. Forking from u_2 at the instant falling is predicted is a hypothetical branch u_2^{bal} that would occur if C^{bal} is continued to be applied rather than switching to C^{fall}: the primary motivation for predicting falling early is that u_2^{bal} is an undesirable trajectory. Yet another scenario to be avoided involves a false positive (trajectory u_4), in which falling is predicted incorrectly *inside* balanced. Needlessly the robot switches to C^{fall} as a response, and this may precipitate a fall. In direct contrast, along a false negative case (trajectory u_5), falling is never predicted until fallen is reached.

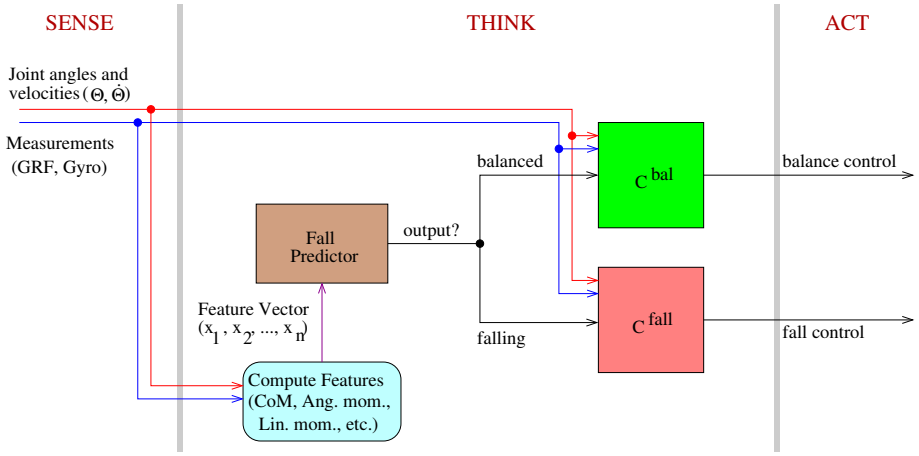


Fig. 4. High-level control flow: sensor readings and measurements are processed into features used by the fall predictor, whose output determines whether C^{bal} or C^{fall} should be applied.

3.3. *Objectives of Fall Prediction*

False negatives can be weeded out effectively by adding a rule to predict falling if the CoM drops below some vertical height threshold (0.48m for our robot). On the other hand, 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** ($\tau_{\mathbf{lead}}$) 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 τ_{lead} imply that C^{fall} gets more time on average to respond to a fall; thus τ_{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 τ_{lead} (zero). At the opposite extreme, a predictor that always predicts falling has maximal τ_{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 τ_{lead}. In our experiments we compute FPR and τ_{lead} with respect to a set of 1000 trajectories recorded on the robot. The next section describes the process of recording the trajectories.

4. Generating Trajectories

We use the commercial robotics simulation software Webots^{TM} to simulate an ASIMO-like robot with 26 degrees of freedom.[37] 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 forward-backward or left-right axes exceeds some threshold, the robot widens its stance, thereby increasing the area of its support polygon and lowering its CoM. This action is effective in thwarting falls caused by impulses of up to $40Ns$, as observed from our simulation results in Section 6. Under the fall controller C^{fall} the robot remains frozen in the target pose reached by the balance controller.

The robot's support polygon at any instant of time during its gait is the convex hull of all the points of contact of the robot's feet with the ground. The Center of Pressure (CoP) is the point inside the support polygon through which the resultant contact force between the robot and the ground acts. It is well known that when the robot starts tipping, the CoP resides at some edge or corner of the support polygon. Taking this fact into consideration, we posit that measurements such as the displacement and velocity of the CoM taken *relative* to the CoP can directly help discriminate falling states from balanced states. Correspondingly we define a Cartesian coordinate system with its origin at the CoP, with x and y axes along the ground in the robot's sagittal and frontal planes respectively, and z axis vertical (Figure 5).

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

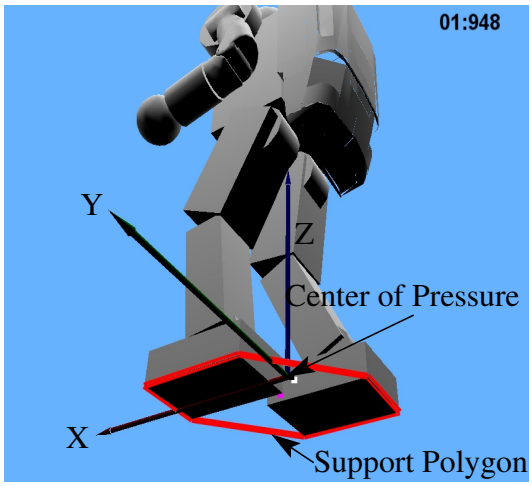


Fig. 5. Cartesian coordinate system, with its origin at the CoP.

trajectories. Each "push" comprises a constant force application for $0.1s$; the force magnitude is drawn uniformly at random 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 at random from $[-180°, 180°]$, at a height above its CoM drawn uniformly at random from $[-0.05m, 0.25m]$. The ranges mentioned are chosen such that roughly half of the trajectories result in fall. Indeed, out of the data set of 1000 trajectories we collect, 554 result in a fall. Along each trajectory that enters fallen, we label states following our push — an external force — but preceding the entry into fallen, as falling. Trajectories *not* ending in a fallen state have all their states labeled balanced.

It is worth mentioning that the external forces causing fall under natural operation of the robot could assume values outside the ranges chosen in our training regimen, impinge parts of the robot's body other than its torso, and act for different durations. Nonetheless, we hypothesize that states resulting from our wide range of impinging impulses will be similar to the states arising when the robot is subjected to more realistic disturbances. Indeed a fall predictor aims to categorize the states resulting from disturbances, and not the source of a disturbance. A related point is that falls are likely to occur significantly less frequently during the robot's normal routines. Reflecting the requirement that fall must be predicted accurately and reliably, even if it is a rare occurrence, we intentionally generate a high fraction of trajectories going through the falling class. The FPR values reported in Section 6 are therefore significantly higher than what might be encountered in practice during the robot's normal operation.

Figure 6 shows examples of trajectories taken by the robot after the force is removed, projected on different axes. Figures 6(a) and 6(b) show the evolution of height of the robot's CoM with time and horizontal components of its linear momentum, respectively. It is apparent that there is no clear separation between balanced and falling states in these projections. Specifically we observe that it takes some time after the impact force has been removed for the balanced and falling trajectories to begin separating. In Figure 6(a) we see a predictor that predicts falling if the CoM height falls below some threshold, and balanced otherwise. Fixing the threshold to obtain an FPR of 0 yields a very low value of τ_{lead} ($0.14s$); however, increasing τ_{lead} even to $0.42s$, FPR is raised to a very high value (0.92). Figure 6(b) shows similar results obtained by thresholding X and Y components of the linear momentum. Among the best tradeoffs obtained here is an FPR of 0.11 with corresponding τ_{lead} of $0.60s$. Thresholding approaches tried with other variables, not shown in Figure 6, achieve no better. In Section 6 we present results indicating that a machine learning solution can indeed reduce FPR further and increase τ_{lead} simultaneously. The next section presents our learning framework.

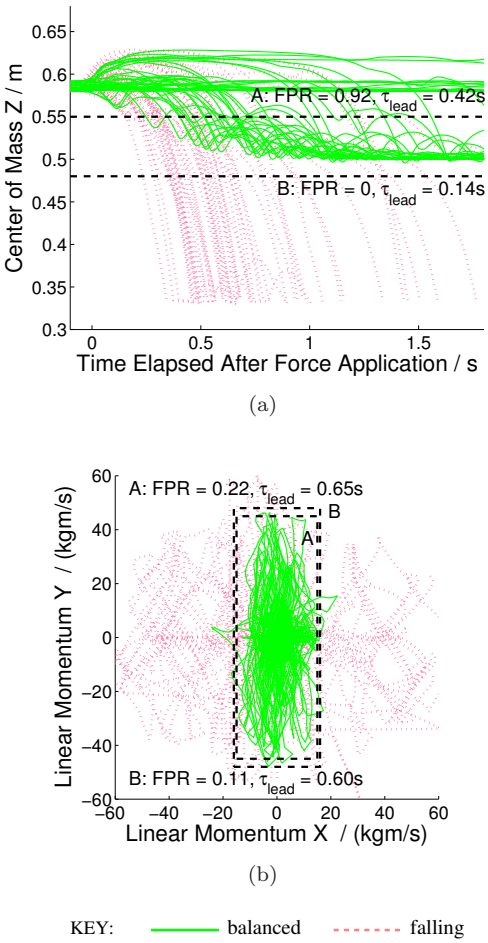


Fig. 6. 100 random trajectories resulting from pushes, projected on different axes. Points on solid lines lie in the balanced class, and points on dashed lines lie in the falling class. In (a) the CoM height above the ground is shown as a function of the time elapsed after the application of the impulse (which is applied in the period [-0.1s, 0]). In (b) the variables plotted are X and Y components of the robot's linear momentum. FPR and τ_{lead} values of fall predictors thresholding the CoM height or horizontal components of the linear momentum are shown.

5. Learning Framework

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 describe our learning process.

5.1. *Features*

The first step in devising a learning solution is selecting features to describe the robot's state. 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 being able to generalize well to nearby states. The sensors and measurements available on a specific robot restrict the scope of the features that can be derived for prediction. For our task, we arrive at a set of 16 features through successive refinement, trial and error. These features are listed in Table 1.

Of the 16 features, 15 are real-valued numbers corresponding to physical quantities such as displacement and momentum of the robot. Recall that these vector quantities are measured in a Cartesian coordinate system centered at the robot's CoP. In general the CoP can dart quickly between points inside the robot's support base, which can itself change dynamically. As a result we do not expect to obtain smooth trajectories in the robot's feature space. Sudden swings in trajectories are quite characteristic of fall: for example, what *appears* to be a configuration certain to result in fall could become perfectly safe as soon as the robot gets an additional foot on the ground. Indeed we find it beneficial to use a discrete feature called the robot's "foot contact mode", which describes the position of the robot's CoP relative to its feet. We now take a closer look at foot contact modes.

Every state of the robot maps to a foot contact mode, which identifies whether its left and/or right feet are touching the ground, and if so, the position of the CoP within the support polygon. Thus, the foot contact mode "LR-INSIDE" corresponds to states in which both left and right feet touch the ground, and the CoP is *inside* the support polygon. Now, suppose the CoP moves to the front edge of the support polygon, the resulting foot contact mode is "LR-FRONT". Other modes are defined similarly; Figure 7 provides a complete list.

Interestingly several foot contact modes that can be imagined in theory seldom occur in practice. For example, "L-RIGHT", under which only the left foot touches the ground, with the CoP on its *right* edge, is hardly encountered in typical trajectories traced by the robot. In total we only find it necessary to consider 16 foot contact modes. Of these, the mode "OUTSIDE" occurs when neither foot touches

Table 1. Features

Physical Quantity	**Type**	**#Features**
CoM displacement	Real-valued	3
Linear momentum	Real-valued	3
Angular momentum about CoM	Real-valued	3
Rate of change of linear momentum	Real-valued	3
Rate of change of angular momentum about CoM	Real-valued	3
Foot contact mode	Discrete, 16 values	1

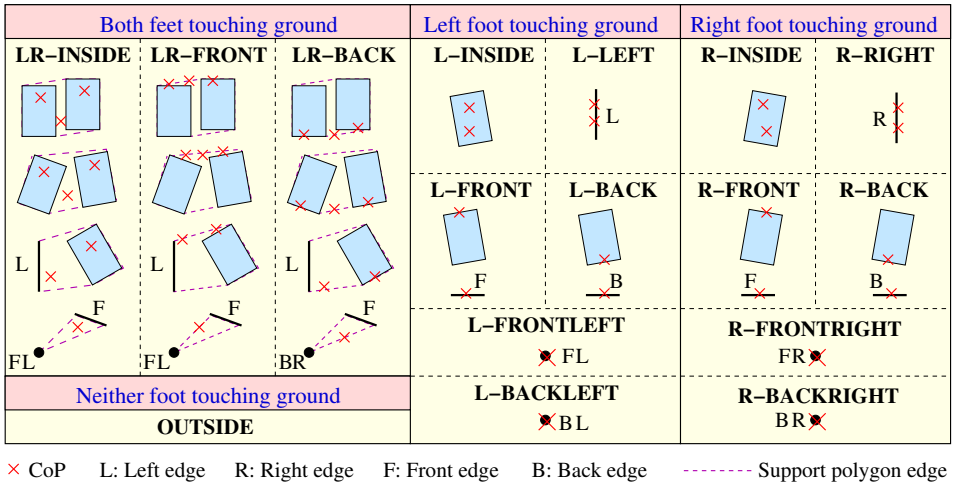


Fig. 7. Foot contact modes. We consider three modes when both feet are touching the ground: LR-INSIDE, LR-FRONT, LR-BACK. Four *representative examples* of feet configurations are shown under each, as a top view with the left foot on the left. The contact region between a foot and the ground may be a rectangle, a line (edge), or a point (corner). In each example several possible positions of the CoP are marked with crosses; where relevant, the support polygon is marked with dotted lines. In LR-INSIDE, the CoP lies inside the support polygon, while in LR-FRONT and LR-BACK, it lies on a front or back edge respectively. When both feet lose contact with the ground, we denote the resulting foot contact mode OUTSIDE. Six modes are defined when the left foot alone touches the ground, which are prefixed with "L-". Likewise, six corresponding modes are defined when the the right foot alone touches the ground, which are prefixed with "R-". A total of 16 foot contact modes are defined. In principle other configurations are possible (such as LR-LEFT and L-RIGHT), but these do not occur frequently in the robot's state trajectories.

the ground, and there is no CoP. In this case, we move the origin of our Cartesian coordinate system to the projection of the CoM on the ground, with the x and y axes still aligned respectively with the robot's sagittal and frontal planes.

5.2. *Learned Representation*

We pose the problem of constructing the fall predictor as a supervised learning problem and experiment with several applicable methods from the WEKA machine learning library.[38] Informal experimentation with a suite of algorithms shows that the best results for our task are achieved with rule-based systems such as decision trees and lists, which perform marginally better than regression-based approaches such as neural networks and radial basis functions. We adopt decision list learning (or "rule learning") for our experiments. Like decision trees, decision lists are grown recursively by splitting nodes, guided by some heuristic such as information gain. The resulting classifier partitions the input space into regions with axis-aligned edges, with one class matched to each region.

We observe that more accurate prediction results when a separate decision list is

learned for each foot contact mode, rather than when a single decision list is learned using the foot contact mode as a feature. A possible explanation for this observation is that the foot contact modes divide the robot's states into homogeneous regions with more regular decision boundaries. Figure 8 shows an example of a learned decision list for the foot contact mode "L-LEFT". A learned fall predictor comprises 16 such lists. From the list shown in Figure 8 it is apparent that learned solutions can be far more complex than the thresholding rules devised in Section 4, which correspond to decision lists with just two or four comparisons. To get a rough idea of the complexity of learned rules, we define the "rule size" of a decision list to be the number of comparison operators it contains: the rule size is 15 in the example in Figure 8. In Section 6 we compare rule sizes of lists learned from differently prepared sets of training data.

5.3. *Parameters to Control Tradeoff between τ_{lead} and FPR*

We are left to explain the process of preparing training data for the supervised learning algorithm, which is a key aspect in our exercise. Typically supervised learning methods for classification seek to minimize a loss function such as the misclassification rate or squared error of the associated regression problem over the input data distribution. However, such loss functions do not completely align with the specific requirements of our application, in which the objectives to optimize are FPR and τ_{lead}. With a prediction accuracy of 99% over all balanced states, a fall predictor could still result in very high FPR if its few incorrect predictions — of predicting falling instead of balanced — are distributed over a large number of balanced tra-

```
if footContactMode = L-LEFT then
  if d-lin-mom-y ≥ -64.50 and d-ang-mom-x ≤ 15.95 and lin-mom-z ≤ -1.16 and d-lin-
  mom-x ≥ -106.14 then
    class ← falling.
  else if ang-mom-y ≤ -14.18 and lin-mom-x ≤ -24.54 then
    class ← falling.
  else if d-lin-mom-y ≥ -19.36 and com-y ≥ -0.04 and lin-mom-y ≥ 5.87 then
    class ← falling.
  else if ang-mom-y ≥ 11.77 and ang-mom-y ≥ 18.32 then
    class ← falling.
  else if com-z ≤ 0.35 then
    class ← falling.
  else if ang-mom-y ≤ -14.18 and lin-mom-y ≤ -11.45 then
    class ← falling.
  else if com-z ≥ 0.62 then
    class ← falling.
  else
    class ← balanced.
```

Fig. 8. Example of a learned decision list, which is invoked when the foot contact mode is "L-LEFT". The number of comparison clauses, or the "rule size", is 15. Units are usual metric.

jectories, rather than contained to a few. On the other hand, a fall predictor with a low accuracy in identifying falling states correctly might still give rise to a high τ_{lead} if its correct predictions occur early in the falling trajectories: once falling is predicted along a trajectory, subsequent predictions are immaterial. In short, prediction accuracy over all the recorded states does not necessarily yield lower FPR and higher τ_{lead}. We describe two techniques that we employ to explicitly promote the learning of fall predictors that minimize FPR and maximize τ_{lead}.

Consider 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, as we observe from Figure 6 (Section 4). Indeed we verify 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. On the other hand, since a falling trajectory will end in a fallen state, states close to this extreme can be easily separated by a simple rule, such as one that thresholds the CoM height.

Figure 9(a) schematically depicts for one balanced and one falling trajectory the CoM height as a function of the time elapsed after the application of the impulse (i.e., after $t_{force\text{-}end}$). 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 time at which the CoM height begins to drop monotonically until a fallen state is reached.

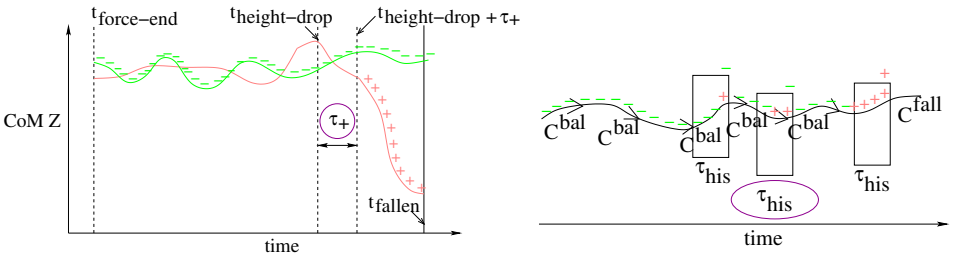


Fig. 9. (a) While *preparing* the training data, trajectories in the falling class are sampled based on the parameter τ_+. The instant $t_{height\text{-}drop}$ is when the height of the CoM begins to monotonically decrease until reaching the fallen class (at time t_{fallen}). Points from falling (marked "+") are sampled in the interval $[t_{height\text{-}drop} + \tau_+, t_{fallen}]$. (b) While *using* a fall predictor, a history of the predictions made in the past duration of τ_{his} is maintained. At time t, falling is predicted only if *all* atomic predictions made in the interval $[t - \tau_{his}, t]$ are fallen.

We define a parameter τ_+, 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 τ_+ is increased, the learned predictor will have lower FPR, but also a lower value of τ_{lead}. Decreasing τ_+ (note that τ_+ can be negative) will likely increase both τ_{lead} and FPR. We still use all the available negative (balanced) examples for training.

Apart from filtering falling trajectories based on τ_+ as described above, we find that the learned performance can be improved by re-weighting the positive and negative training data to provide the learning algorithm. In principle the ratio of weights for positive and negative training instances can also be treated as a parameter to the learning algorithm, but through empirical verification, we observe that weighting balanced instances 4 times the weight of falling instances achieves close to the best values for FPR and τ_{lead} across a broad range of τ_+ values. Thus we do not vary the weight ratio as a parameter.

In attempting to identify a "working range" for τ_+ based on evaluating its relationship with FPR and τ_{lead}, we discover the need to formulate a second parameter, τ_{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 even a single false positive causing the unnecessary deployment of C^{fall}. This is avoided by maintaining a finite history of the predictions made by the decision list, and only predicting falling when the list has consistently predicted falling over all states in the history window. Thus, the atomic predictions made by the learned decision list within the history window are combined to make a single prediction, which is falling only if all atomic predictions are falling. Figure 9(b) depicts this adaptation, which has the effect of decreasing FPR.

The parameter τ_{his} corresponds to the temporal length of the history window maintained. While τ_+ is used while generating data for training, τ_{his} only comes into play *after* the predictor has been learned. Note that under the use of τ_{his}, the fall predictor is no longer a mapping from a feature vector to {balanced, falling}, but a mapping from a **feature vector** and a **history of predictions** to {balanced, falling}. A positive quantity, τ_{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 τ_{lead}. Together, τ_+ and τ_{his} provide handles to control the tradeoff between FPR and τ_{lead}: they can be set by the user as inputs to the learning algorithm.

6. Results

In this section we report experimental results. Since we have two parameters, τ_+ and τ_{lead}, as inputs to the learning process, we obtain a family of solutions. Every combination of τ_+ and τ_{his} yields a separate fall predictor, which registers different

values of FPR and τ_{lead}. Figure 10 plots τ_{lead} values of learned fall predictors against their FPR values. Reported values are obtained through 10-fold cross-validation on a set of 1000 trajectories, as described in Section 4.

The plot in Figure 10 is similar to an ROC curve for classification problems, in which true positive rate is plotted against FPR. In fall prediction it is required that falling states be detected *early* along trajectories reaching fallen: τ_{lead} quantifies the fulfilment of this requirement. Similar to the pattern in a typical ROC curve, we find that decreasing FPR comes at the cost of decreasing τ_{lead}. We pick three learned solutions, FP^{L1}, FP^{L2}, and FP^{L3}, to summarize the observed tradeoff between FPR and τ_{lead}. FP^{L1} is conservative, with high τ_{lead} ($0.90s$) but high FPR (0.46). At the other extreme, FP^{L3} has near-zero FPR, but less than half the τ_{lead} value of FP^{L1}. In between them, FP^{L2} enjoys a relatively low value FPR (0.06), but also a reasonably high value of τ_{lead} ($0.76s$). Indeed FP^{L2} and several other learned predictors compare favorably with the manually designed predictors considered in Section 4 both in terms of FPR and τ_{lead}. Further, having a range of solutions enables predictors to be chosen for specific needs: on a real robot, whose falls could be disastrous, a high reaction time might be favored over a low false positive rate.

The dependence of FPR and τ_{lead} on the parameters of the learning algorithm, τ_+ and τ_{his}, is shown in Figure 11 (plots on left). In keeping with intuition, we find that lower (higher) values of τ_+ and τ_{his} increase (decrease) FPR and τ_{lead}. Note that if we did not withhold any positive instances ($\tau_+ \approx -1.5s$) or use a history

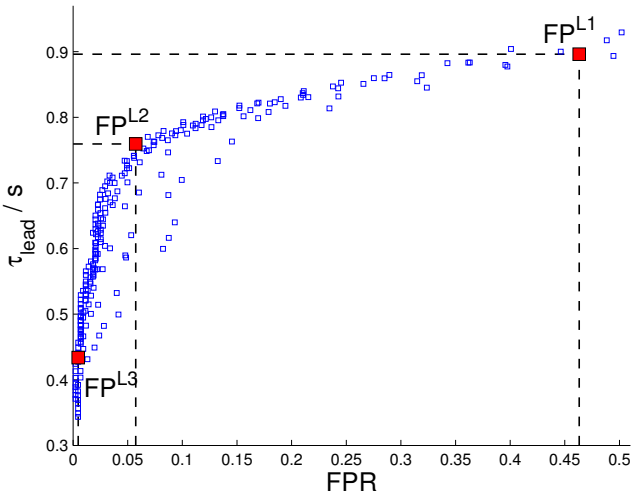


Fig. 10. Tradeoff between FPR and τ_{lead}. Each point corresponds to a trained fall predictor, and marks its τ_{lead} and FPR values. Three predictors — FP^{L1}, FP^{L2}, and FP^{L3} — showcase contrasting configurations, proceeding from high τ_{lead} and high FPR to low τ_{lead} and low FPR.

window ($\tau_{his} = 0$), FPR would be nearly 100%; τ_+ and τ_{his} are *necessary* to deliver the benefits of supervised learning. While the graphs on the left in Figure 11 are based on training with 900 trajectories (cross-validated), we find that roughly 250 trajectories suffice for obtaining FPR $< 10\%$ and $\tau_{lead} > 0.70$s (plots on right).

Recall that separate decision lists are learned for each of 16 foot contact modes: the mean rule size (bottom row in Figure 11) is the average of the rule sizes (number

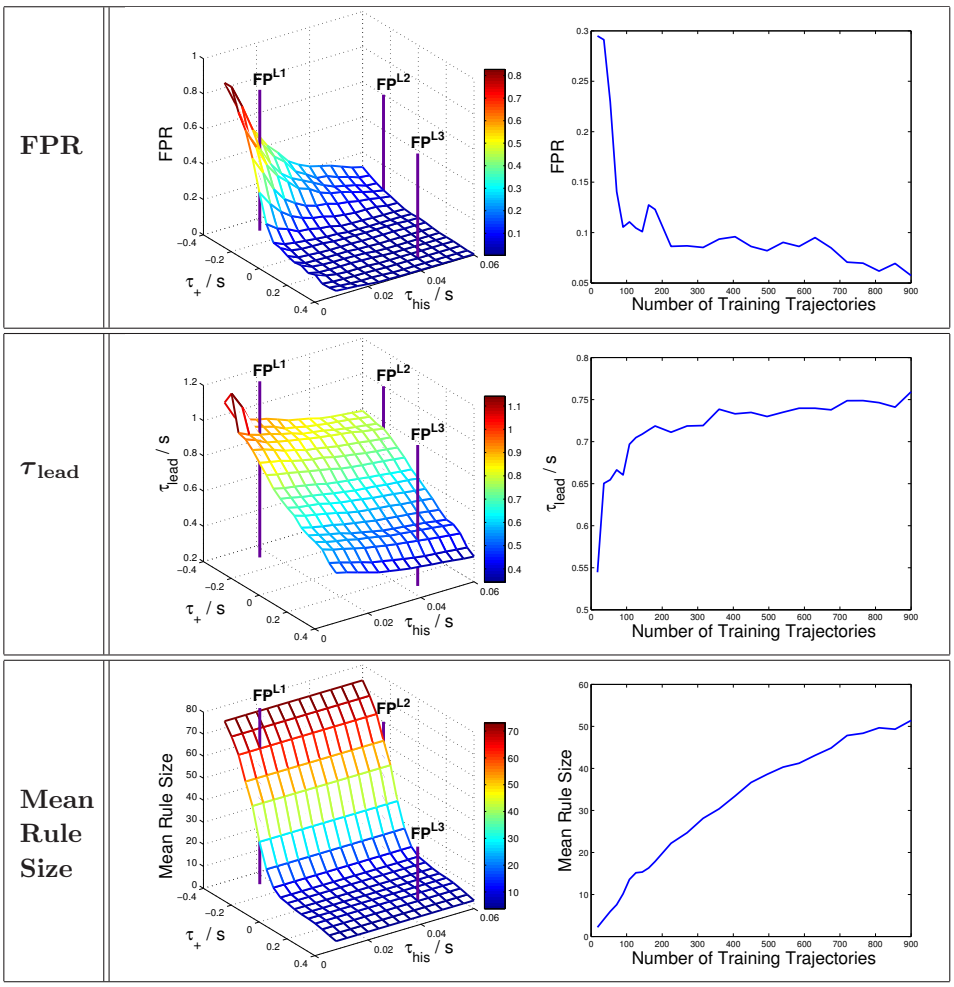


Fig. 11. FPR, τ_{lead}, and the mean rule size of learned fall predictors. The left plot in each row shows the value of the corresponding statistic at different settings of τ_+ and τ_{his}. Three representative predictors, FP^{L1}, FP^{L2} and FP^{L3}, are marked. Note that mean rule size does not depend on τ_{his}. The right plot in each row records the effect of the number of training trajectories in learning the predictor. FPR, τ_{lead}, and mean rule size are plotted against the number of trajectories used in training, under settings corresponding to FP^{L2} ($\tau_+ = -0.25s$, $\tau_{his} = 0.06s$). All reported values are averages of 10 folds of cross-validation.

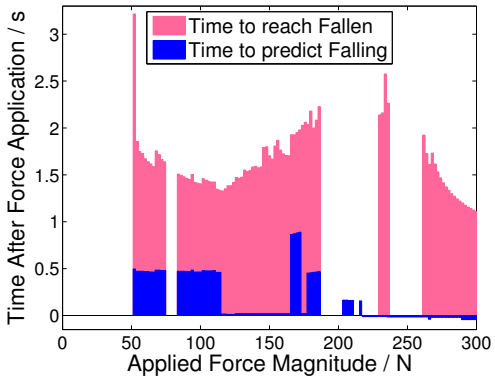
of comparison operators) of these decision lists. Note that the mean rule size does not depend on τ_{his}, since τ_{his} is only used in making predictions *after* the decision list has been learned, and the rule size already determined. The mean rule size increases sharply as τ_+ is decreased, i.e., as points earlier in falling trajectories are included in the training data. This trend is explained by the fact that the boundary between the balanced and falling classes is irregular: rules for predicting falling early have to distinguish fine distinctions 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.

Table 2 shows a comparison of the three learned predictors, FP^{L1}, FP^{L2} and FP^{L3}, summarizing their input parameters and performance statistics. In Figure 12 we revisit the illustrative example introduced in Section 4. In this example, from the same point in the robot's configuration space, it is subjected to forces with increasing magnitudes. From the plots it is apparent that as we progress from FP^{L1} to FP^{L3} nearly every fall is predicted later; within the pockets of "no fall" FP^{L1} makes several incorrect predictions of fall. Many of its predictions (at force magnitudes beyond 220N) are made even while the 100ms impulse is being applied.

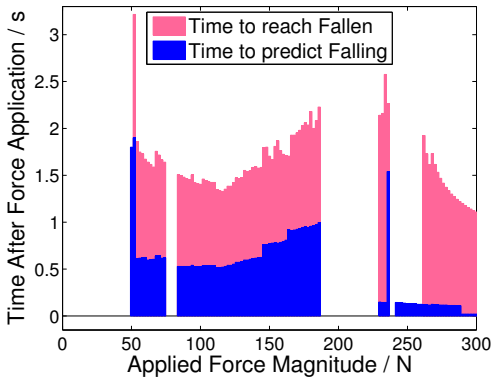
Figures 13 and 14 relate to Figure 6, which shows 100 trajectories projected along axes corresponding to CoM displacement (Figure 13) and linear momentum (Figure 14). The projections are annotated with the points at which FP^{L1}, FP^{L2} and FP^{L3} make their predictions of falling, and indeed indicate whether the predictions are correct. In the plots in Figure 13, the x axis marks the time elapsed after the termination of the 100ms impulse. This projection offers visible evidence that falling predictions are made later as we proceed from FP^{L1} to FP^{L3}. Correspondingly, fewer predictions are false positives. We observe in Figure 14 that FP^{L3}, a careful predictor, predicts falling less often at low magnitudes of linear momentum.

Table 2. A tabular comparison of learned fall predictors FP^{L1}, FP^{L2}, and FP^{L3}. Rows 1 and 2 show input parameters; rows 3, 4, and 5 report performance statistics. Each reported statistic is the average of 10 folds of cross-validation, carried out over a data set with 1000 trajectories.

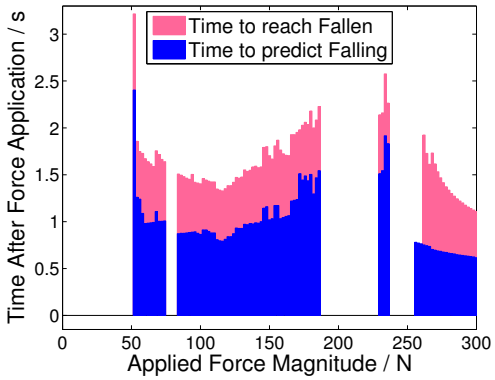
Predictor	$\mathbf{FP^{L1}}$	$\mathbf{FP^{L2}}$	$\mathbf{FP^{L3}}$
τ_+ (/s)	-0.3	-0.25	0.3
τ_{his} (/s)	0.016	0.060	0.044
FPR	0.46	0.06	0.005
τ_{lead} (/s)	0.90	0.76	0.43
Mean rule size	61.6	51.4	5.3



(a) FP^{L1}

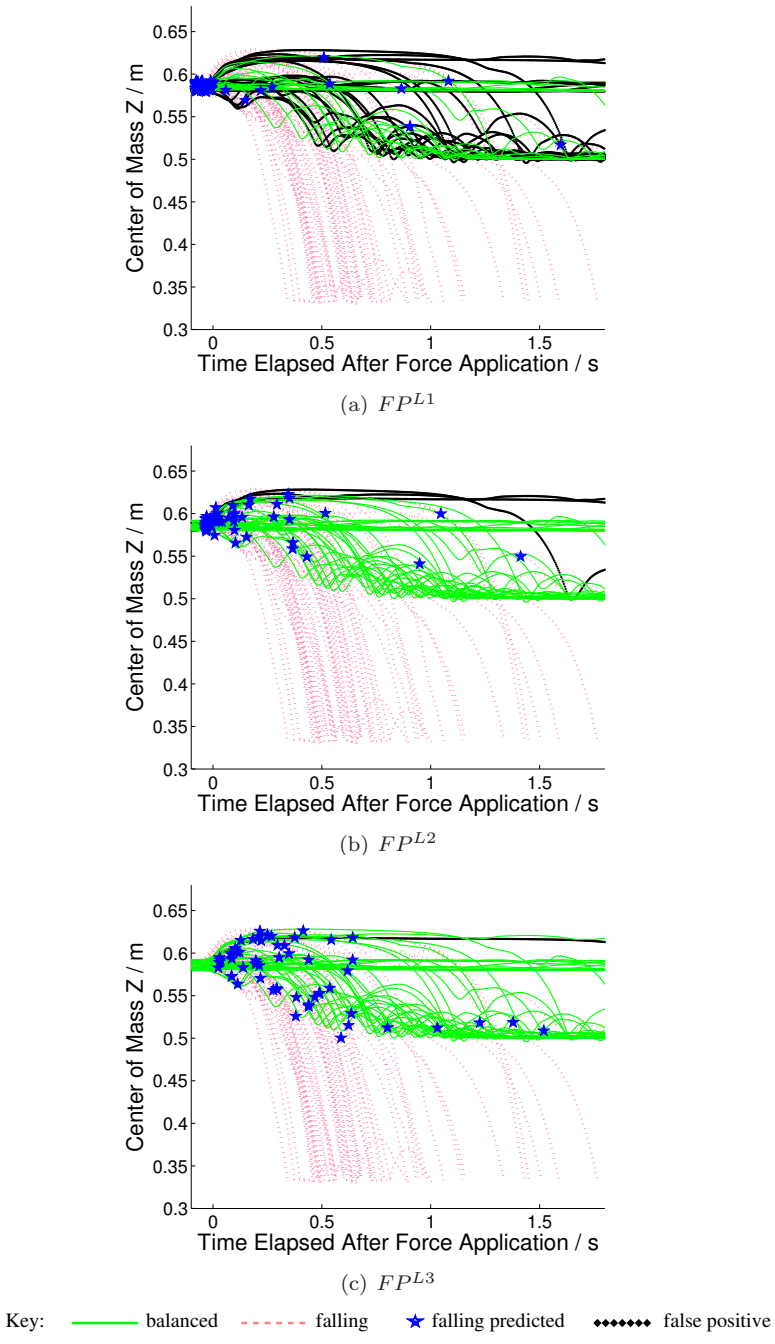


(b) FP^{L2}



(c) FP^{L3}

Fig. 12. Plots corresponding to the example discussed in Section 1 (Figure 2). Each plot corresponds to a different predictor: one set of bars (light) mark the time after force application to reach a fallen state (if at all), while the other set of bars (dark) show the time elapsed for *predicting* falling (if at all). Dark bars that do not overlap with light bars represent false positives.



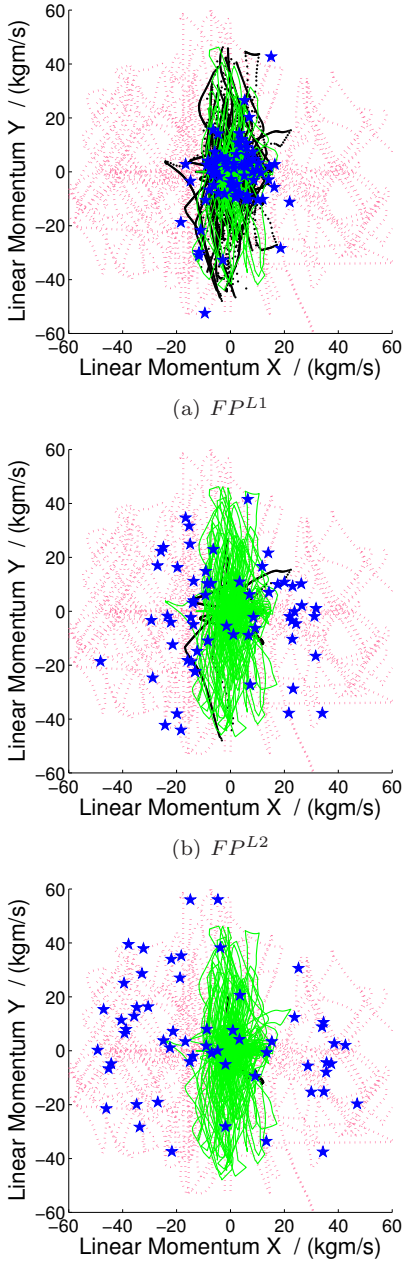
(a) FP^{L1}

(b) FP^{L2}

(c) FP^{L3}

Key: balanced falling falling predicted false positive

Fig. 13. Plots corresponding to the the 100 trajectories shown earlier in Figure 6(a). Each plot corresponds to a different predictor: light solid lines mark trajectories within the balanced class, and dashed lines mark trajectories in the falling class. Stars on light lines mark the points that falling is predicted; the lines are shown dark solid if these predictions were incorrect (false positives).



(a) FP^{L1}

(b) FP^{L2}

(c) FP^{L3}

Key: balanced falling falling predicted false positive

Fig. 14. Plots corresponding to the the 100 trajectories shown earlier in Figure 6(b). Each plot corresponds to a different predictor: light solid lines mark trajectories within the balanced class, and dashed lines mark trajectories in the falling class. Stars on light lines mark the points that falling is predicted; the lines are shown dark solid if these predictions were incorrect (false positives).

7. Summary and Discussion

A humanoid robot cannot completely avoid falling, and so it requires a concerted strategy to control a fall when it occurs. The precursor to fall control is fall prediction, which is the subject of our paper. In particular we are concerned with predicting fall when a human-size robot is subjected to relatively strong disturbances, resulting in quite complex dynamics. We adopt a machine learning approach.

The fall pattern of a robot depends on its mass, inertia, geometry, dimensions, material composition, motor torque limits, and controls. By not explicitly modeling any of these aspects, but rather depending on the information implicit in recorded trajectories, our method can be applied with little change on different robots. A different robot model and/or controller could benefit from specifically designed features; however, whereas a model-based method would require recalibration or a complete revision of the model, our approach primarily requires data generation and training.

We find it beneficial to learn a separate fall predictor under every foot contact mode. 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 τ_{lead}, which are fundamental desiderata while performing fall prediction. By discarding positive training instances on the basis of the parameter τ_+, we commit to reduce the incidence of false positives, even if it reduces the lead time to fall. Likewise τ_{his} is used to weed out stray false positives by averaging over a time window, which again has the compensatory effect of reducing τ_{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, and have been used successfully for fall prediction.[22] Such models have the additional advantage of providing probabilistic (or "soft") classifications of state, rather than the categorical mapping learned by decision lists.

Our current approach ignores aspects such as variable ground friction, sensor and actuation noise, and complex foot geometries. We do not expect fall predictors learned in simulation to register the same FPR and τ_{lead} values when deployed on a real ASIMO robot. At best, we expect running our learning algorithm on data recorded on the real robot to yield results with similar trends. Note that reasonable results are obtained by training on roughly 250 trajectories. The pushing regimen described in our work would be tedious if a human agent is involved in the generation of trajectories 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. Kohl and Stone adopt a similar strategy to optimize the forward walking speed of four-legged Aibo robots.[39]

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. As humanoid robots gain autonomy and begin exploring unknown environments, it might not be feasible to gather

data from all possible situations beforehand and learn off-line. Although off-line learning could initialize the robot's fall predictor, it would have to be refined in an on-line, incremental manner as the robot encounters novel situations. In real-world systems, failures tend to occur less frequently, and so it becomes necessary if learning on-line to extract the most information from the few — in this case falls — that occur.[40] A more worrying concern for an autonomous robot is a lack of reliable supervisory signals. If the robot falls, how should it decide at which points in its trajectory a force was acting, and at which precise instant it crossed from balanced to falling? In our simulation we have the benefit of hindsight after a fall to identify such details for training.

We have focused on learning the high-level decision of choosing between a single balance controller and a fall controller. A natural extension is to further divide falling states based on direction, speed, etc., and design specific fall controllers for each category. More generally, a significant body of work addresses the challenge of adapting the low-level controllers themselves, on humanoid and other robots.[41–43] To learn a fall management strategy, large negative rewards could be provided to learn to avert falls, with graded rewards to prompt the minimization of fall-related damage. In such an integrated framework, fall prediction would become an *implicit* step, determined by the utilities of applying different controls from a state. Our promising results from treating fall prediction as an isolated supervised learning problem encourage the pursuit of a general learning architecture for humanoid fall management.

Acknowledgments

Shivaram Kalyanakrishnan was supported by the summer internship program at the Honda Research Institute, USA. The authors thank the anonymous reviewers of this article and those of earlier versions of this work.

References

1. Satoshi Kagami, Fumio Kanehiro, Yukiharu Tamiya, Masayuki Inaba, and Hirochika Inoue. Autobalancer: An online dynamic balance compensation scheme for humanoid robots. In Bruce Randall Donald, Kevin M. Lynch, and Daniela Rus, editors, *Algorithmic and Computational Robotics: New Directions : the Fourth Workshop on the Algorithmic Foundations of Robotics*, pages 329–339. A K Peters, Ltd., 2001.
2. Akinori Nishio, Kentaro Takahashi, and Dragomir N. Nenchev. Balance control of a humanoid robot based on the reaction null space method. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*, pages 1996–2001. IEEE, 2006.
3. Napoleon, Shigeki Nakaura, and Mitsuji Sampei. Balance control analysis of humanoid robot based on ZMP feedback control. In *Proceedings of the IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS 2002), volume 3, pages 2437–2442. IEEE, 2002.

4. Kunihiro Ogata, Koji Terada, and Yasuo Kuniyoshi. Real-time selection and generation of fall damage reduction actions for humanoid robots. In *Proceedings of the 8th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2007)*, pages 233–238. IEEE, 2008.
5. Qiang Huang and Yoshihiko Nakamura. Sensory reflex control for humanoid walking. *IEEE Transactions on Robotics*, 21(5):977–984, 2005.
6. Shunsuke Kudoh, Taku Komura, and Katsushi Ikeuchi. Stepping motion for a human-like character to maintain balance against large perturbations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2006)*, pages 2661–2666. IEEE, 2006.
7. Jerry Pratt, John Carff, Sergey Drakunov, and Ambarish Goswami. Capture point: A step toward humanoid push recovery. In *Proceedings of the Sixth IEEE-RAS International Conference on Humanoid Robots (Humanoids 2006)*, pages 200–207. IEEE, 2006.
8. Riadh Zaier and Shinji Kanda. Piecewise-linear pattern generator and reflex system for humanoid robots. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA 2007)*, pages 2188–2195. IEEE, 2007.
9. Tobias Wilken, Marcell Missura, and Sven Behnke. Designing falling motions for a humanoid soccer goalie. In *The 4th Workshop on Humanoid Soccer Robots at the 9th International Conference on Humanoid Robots (Humanoids 2009)*, 2009.
10. Kiyoshi Fujiwara, Fumio Kanehiro, Shuji Kajita, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. UKEMI: Falling motion control to minimize damage to biped humanoid robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, volume 3, pages 2521–2526. IEEE, 2002.
11. Kiyoshi Fujiwara, Shuuji Kajita, Kensuke Harada, Kenji Kaneko, Mitsuharu Morisawa, Fumio Kanehiro, Shinichiro Nakaoka, and Hirohisa Hirukawa. Towards an optimal falling motion for a humanoid robot. In *Proceedings of the 6th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2006)*, pages 524–529. IEEE, 2006.
12. Kiyoshi Fujiwara, Shuuji Kajita, Kensuke Harada, Kenji Kaneko, Mitsuharu Morisawa, Fumio Kanehiro, Shinichiro Nakaoka, and Hirohisa Hirukawa. An optimal planning of falling motions of a humanoid robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, pages 456–462. IEEE, 2007.
13. J. Ruiz-del-Solar, R. Palma-Amestoy, R. Marchant, I. Parra-Tsunekawa, and P. Zegers. Learning to fall: Designing low damage fall sequences for humanoid soccer robots. *Robotics and Autonomous Systems*, 57(8):796–807, 2009.

14. Seung-kook Yun, Ambarish Goswami, and Yoshiaki Sakagami. Safe fall: Humanoid robot fall direction change through intelligent stepping and inertia shaping. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009)*, pages 781–787. IEEE, 2009.
15. Umashankar Nagarajan and Ambarish Goswami. Generalized direction changing fall control of humanoid robots among multiple objects. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA 2010)*. IEEE, May 2010 2010. To appear.
16. Yoshiaki Sakagami, Ryujin Watanabe, Chiaki Aoyama, Shinichi Matsunaga, Nobuo Higaki, and Kikuo Fujimura. The intelligent ASIMO: system overview and integration. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, volume 3, pages 2478–2483. IEEE, 2002.
17. Pierre-Brice Wieber. On the stability of walking systems. In *The Third IARP International Workshop on Humanoid and Human Friendly Robotics*, 2002.
18. Jean-Pierre Aubin. *Viability Theory*. Birkhäuser Boston, 1991.
19. Pierre-Brice Wieber. Viability and predictive control for safe locomotion. In *Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1103–1108. IEEE, 2008.
20. Reimund Renner and Sven Behnke. Instability detection and fall avoidance for a humanoid using attitude sensors and reflexes. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*, pages 2967–2973. IEEE, 2006.
21. J. G. Daniël Karssen and Martijn Wisse. Fall detection of two-legged walking robots using multi-way principal components analysis. *International Journal of Humanoid Robots*, 7(1):73–93, 2010.
22. O. Höhn and W. Gerth. Probabilistic balance monitoring for bipedal robots. *International Journal of Robotics Research*, 28(2):245–256, 2009.
23. Kunihiro Ogata, Koji Terada, and Yasuo Kuniyoshi. Falling motion control for humanoid robots while walking. In *Proceedings of the 7th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2007)*, pages 306–311. IEEE, 2007.
24. Pat Langley and Herbert A. Simon. Applications of machine learning and rule induction. *Communications of the ACM*, 38(11):54–64, 1995.
25. Mike Chen, Alice X. Zheng, Jim Lloyd, Michael I. Jordan, and Eric Brewer. Failure diagnosis using decision trees. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC 2004)*, pages 36–43. IEEE, 2004.
26. David L. Olson and Dursun Delen. *Advanced Data Mining Techniques*. Springer-Verlag, 2008.
27. Thomas Degen, Heinz Jaeckel, Michael Rufer, and Stefan Wyss. SPEEDY: A fall detector in a wrist watch. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers (ISWC 2003)*, pages 184– 187. IEEE, 2003.
28. Alan K. Bourke, Karol J. O'Donovan, and Gearóid M. ÓLaighin. Distinguishing

falls from normal ADL using vertical velocity profiles. In *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS 2007)*, pages 3176–3179. IEEE, 2007.

29. N. Noury, A. Fleury, P. Rumeau, A. K. Bourke, G. ÓLaighin, V. Rialle, and J.E. Lundy. Fall detection - principles and methods. In *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS 2007)*, pages 1663–1666. IEEE, 2007.
30. J. J. Gertler. Survey of model-based failure detection and isolation in complex plants. *IEEE Control Systems Magazine*, 8(6):3–11, December 1988.
31. Raffaella Mattone and Alessandro De Luca. Relaxed fault detection and isolation: An application to a nonlinear case study. *Automatica*, 42(1):109–116, 2006.
32. Cliff Changchun Zou, Lixin Gao, Weibo Gong, and Don Towsley. Monitoring and early warning for Internet worms. In *Proceedings of the 10th ACM conference on computer and communications security (CCS 2003)*, pages 190–199. ACM, 2003.
33. R. Parasuraman, P. A. Hancock, and O. Olofinboba. Alarm effectiveness in driver-centred collision-warning systems. *Ergonomics*, 40(3):390–399, 1997.
34. Nate Kohl, Kenneth Stanley, Risto Miikkulainen, Michael Samples, and Rini Sherony. Evolving a real-world vehicle warning system. In Mike Cattolico, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)*, pages 1681–1688. ACM, 2006.
35. Jacques Talandier and Emile A. Okal. An algorithm for automated tsunami warning in French Polynesia based on mantle magnitudes. *Bulletin of the Seismological Society of America*, 79(4):1177–1193, August 1989.
36. W. L. Tung, C. Quek, and P. Cheng. GenSo-EWS: a novel neural-fuzzy based early warning system for predicting bank failures. *Neural Networks*, 17(4):567–587, 2004.
37. Olivier Michel. Webots^{TM}: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.
38. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.
39. Nate Kohl and Peter Stone. Machine learning for fast quadrupedal locomotion. In Deborah L. McGuinness and George Ferguson, editors, *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616. AAAI Press, 2004.
40. Christian Plagemann, Dieter Fox, and Wolfram Burgard. Efficient failure detection on mobile robots using particle filters with Gaussian process proposals. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 2185–2190. IJCAI, 2007.
41. Duško Katić and Miomir Vukobratović. Survey of intelligent control techniques for humanoid robots. *Journal of Intelligent Robotic Systems*, 37(2):117–141, 2003.

42. Kentarou Hitomi, Tomohiro Shibata, Yutaka Nakamura, and Shin Ishii. Reinforcement learning for quasi-passive dynamic walking of an unstable biped robot. *Robotics and Autonomous Systems*, 54(12):982–988, 2006.
43. Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

Shivaram Kalyanakrishnan is a Ph.D. candidate in the Department of Computer Science at the University of Texas at Austin, where he is advised by Peter Stone. Earlier he obtained a B.Tech. degree from the Department of Computer Science and Engineering at the Indian Institute of Technology Madras.

Shivaram Kalyanakrishnan's primary research interest is in the application of reinforcement learning algorithms to complex sequential decision making problems. His work has made both algorithmic and engineering contributions to the area, and has been tested in domains such as robot soccer, humanoid robotics, and Tetris. He is an active participant in the annual RoboCup competitions, where he has won two *Best Student Paper* awards. He undertook an internship at the Honda Research Institute in California, USA, where he collaborated with Ambarish Goswami on the problem of humanoid fall prediction. He has contributed as an organizer and a reviewer to several conferences and competitions related to artificial intelligence and machine learning.

Ambarish Goswami has been with Honda Research Institute USA, Inc., in California, for the past eight years, where he is currently a Principal Scientist. His field is dynamics and control, and currently his main research is in balance maintenance and fall for the Honda humanoid robot ASIMO. He received the Bachelor's degree from Jadavpur University, India, the Master's degree from Drexel University, and the Ph.D. degree from Northwestern University, all in Mechanical Engineering.

Ambarish Goswami's Ph.D. work, under Michael Peshkin, was in the area of automated assembly and robot-assisted surgery. For four years following his graduation he worked at the INRIA Laboratory in Grenoble, France, as a member of the permanent scientific staff. He became interested in human walking and in biomechanics while working on the first anthropomorphic biped robot "BIP" in France. This interest in gait study subsequently brought him to the Center for Human Modeling and Simulation (as an IRCS Fellow) of the University of Pennsylvania, Philadelphia, and a three year position as a core animation software developer for 3D Studio Max at Discreet (Autodesk). Ambarish Goswami has held visiting researcher positions at the Ohio State University and the University of Illinois at Urbana-Champaign for short periods.