Learning to Prevent Failure States for a Dynamically Balancing Robot

Jeremy Searock ad Brett Browning
Carnegie Mellon University
School of Computer Science
5000 Forbes Avenue, Pittsburgh, Pennsylvania, USA
jsearock@andrew.cmu.edu, brettb@cs.cmu.edu*

Abstract

To achieve robust autonomy, robots must avoid getting stuck in states from which they cannot recover without external aid. While this is the role of the robot's control algorithms, these are often imperfect. We examine how to detect failures by observing the robot's internal sensors over time. For such cases, triggering a response when detecting the onset of a failure can increase the operational range of the robot. Concretely, we explore the use of supervised learning techniques to create a classifier that can detect a potential failure and trigger a response for a dynamically balancing robot. We present a fully implemented system, where the results clearly demonstrate an improved safety margin for the robot.

Introduction

Many potential robotics applications require a robot to operate autonomously in complex environments where human intervention is either not possible or expensive. Most robot control algorithms, however, are imperfect and can drive the robot into states from which it cannot recover by itself. Typically, such situations arise from imperfect external sensing, perception, or modeling algorithms. For some robots, these situations can be detected through observation of the robot's internal, or kinesthetic sensors. Information derived from wheel encoders, pitch sensors, motor current sensors, and so on, can indicate that the robot is about to fail.

In this paper, we explore how internal sensors can be exploited to prevent failures. We seek to develop a safety mechanism that allows a robot to perform its normal behavior, except when a failure is imminent as indicated by its internal sensors. In such situations, the safety mechanism can trigger a conservative control policy with the primary goal of removing the robot from danger. We believe that such a mechanism can greatly improve the operational range of a robot, and thereby reduce the rate for human intervention.

*This work was supported by United States Department of the Interior under Grant No. NBCH-1040007. The content of the information in this publication does not necessarily reflect the position or policy of the Defense Advanced Research Projects Agency (DARPA), US Department of Interior, US Government, and no official endorsement should be inferred.

In this paper, we contribute an approach that makes use of a trained classifier to detect potential failure states from the robot's internal sensors. Upon detecting dangerous situations, the classifier triggers a conservative safety response behavior. We have fully implemented this approach on a dynamically balancing robot, the Segway RMP. Due to its dynamic balancing, it is relatively easy for the robot to fall over should it collide with obstacles. Our results show that the safety response mechanism, which requires relatively small amounts of training data and developer time, enable the robot to safely detect collisions with obstacles and to extricate itself from all but the worst high-speed collisions. Thus, the approach greatly increases the robustness, and therefore the operational range of the robot.

The paper is structured as follows. In the following section, we introduce the concepts that underlie our approach using a Markov Decision Process (MDP) framework. Then, we present an implementation on the Segway RMP, followed by the performance results of this implementation under various conditions. Finally, we present the relevant related work and then conclude the paper.

An Approach to Failure Prevention

For an autonomous robot to operate effectively, it must avoid unrecoverable failures. That is, it must avoid situations from which it cannot recover without external aid. For a dynamically balancing robot for example, this means it must avoid collisions that lead to falling over (e.g. figure 3). While this is possible with the right control policy, finding this policy is often hard and imperfections in perception compound the problem. In this paper, we propose an alternative method, whereby the robot executes its nominal policy which is augmented by a fixed, simple failure prevention policy that is executed only when a failure is considered likely. We assume the underlying model of the robot is a Markov Decision Process(MDP) (Puterman 1994).

Definitions

We phrase the problem within an MDP framework. Let, S be the state space of the robot, which the robot transitions through over time as $s_1, \ldots, s_t$. Let A be the set of possible actions the robot can choose to execute and P be the stochastic transition function $P : S \times A \times S \to [0, 1]$, that describes how the robot transitions from state to state given its action choices. Finally, we assume the robot has a control policy $\pi_{norm} : S \rightarrow A$ for selecting actions given the state. We call this the *normal* control policy.

For the Segway RMP, the state space can be observed by the internal sensors of the robot (e.g. pitch, pitch rate, wheel speeds, and so on), the actions are the forward and turn velocity commands. The control policy is the algorithm for commanding the robot to achieve the robot's goals, be it to kick a ball or some other task.

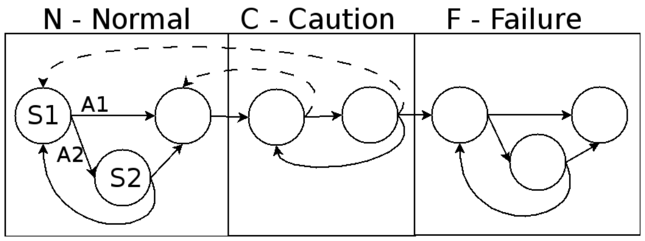


Figure 1: A schematic of the three types of states and the transition relationship between them. Dashed lines indicate lower probabilities.

A failure, in the context of this paper, is when the robot encounters a situation from which it cannot extricate itself without external aid. We concentrate on preventing one Segway RMP failure, falling over. Other failures could be motor failures or other mechanical faults. Using this as a starting point, we now categorize states into three separate types, as:

- Failure F: No control policy the robot can execute will enable it to recover from a failure without external aid.
- Caution C: The robot has a high probability of reaching failure state by following π_{norm}.
- Normal N: The robot can follow its normal control policy π_{norm} with relative safety.

Thus, we partition the state space into F, C, N, where F are the failure states, C are the caution states, and N are normal states which consists of the remainder of the state space (i.e. $N = S - (F \cup C)$). We will refer to $(C \cup F)$ as abnormal states. For the Segway RMP using only internal sensors, F corresponds to when the robot is irretrievably falling over, C to the states when it has collided with an object and cannot continue its prior behavior, and N to the normal driving conditions. By definition once in F the robot cannot return to C or N. States in C are able to transition to N but have a high probability of transitioning to F given the control policy(see figure 1).

We define the caution states C as those states that are *near* F, in terms of the probability of a failure in the near-term future when acting under the control policy $\pi_{norm}(s)$. Additionally, we require that C is defined such that there is no direct transition from N to F. In our approach, we do not explicitly identify C, but learn a classifier from labeled data that identifies between N and and abnormal states, $(C \cup F)$. If an abnormal state is identified quickly enough, the robot will be in a caution state.

Our goal therefore is to detect when the robot is in a caution state by quick detection of abnormal sensor data. When it is, the robot should activate its safety reflex ($\pi_{safety}(s)$) to recover from the potential failure.

Overview of Failure Prevention Method

As sold by Segway LLC, the RMP's control systems and dynamics are unknown. As a result, an accurate model is not readily attainable. Therefore, we decided to use a non model-based method. We propose that learning methods such as C4.5 can be used to obtain classifiers capable of quickly distinguishing between normal (N) and abnormal ($C \cup F$) states. With such a classifier, caution states can be identified, allowing time for a robot to prevent a failure.

Our approach contains several steps that lead to robust failure prevention. These are:

1. Identify Failure States: First, the types of failure $F_1 \ldots F_n$ must be identified. Through the course of experimentation with a robot, certain unrecoverable failure states will occur. When they do, the type of failure and situation which lead to the failure can be categorized.

2. Label Synchronized Internal Sensor Data: With the dangerous situations known, we record the internal sensor data of the robot as it transitions from a normal state, through a caution state, to a failure state. This data originates from a state observation vector z sent from the robot reporting each sensors current reading. A log can be collected recording the state vector over time :$z_1 \ldots z_t$. Two different methods can be used to identify within the log when a transition begins from a normal state to a caution state. While recording the log, we can watch the robot and mark when the robot enters a dangerous situation. Alternatively, we can observe the state transitions by plotting each time-series log of the state vector and marking the time steps in which a transition took place between a normal to abnormal state.

3. Develop a Classifier: With labeled data, methods including learning techniques can be used to obtain classifiers that are capable of autonomously detecting abnormal states, i.e. caution states and failure states. Since no actions will result in a transition from a failure state to a normal state, the classifiers must be able to detect abnormal data quickly so that a caution state is identified. Caution state detection allows time for actions to prevent failure.

4. Take Preventative Actions: Once a caution state is detected, the robot needs to change its control policy to prevent failure. These policies will vary across different platforms and environments. The success of the policy also depends on the speed of the detection.

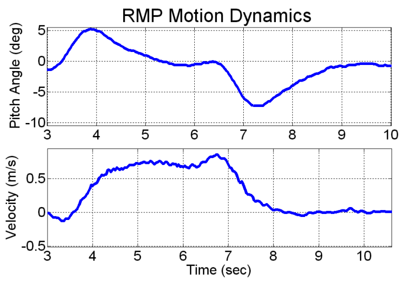
Preventing Failures for a Segway RMP

The Segway RMP is a large dynamically balancing robot. Its weight of just over $100kg$ means that preventing failures, in this case preventing it from falling over, considerably extends the operational capabilities of the robot and reduces the need for human intervention. Therefore, it is an ideal test case for failure prevention techniques. In this section, we describe an implementation of our approach on a Segway RMP. We begin our discussion by describing the RMP platform.

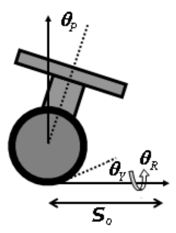
The Segway Robot Mobility Platform (RMP)

The Segway RMP, with its zero turning radius, speeds of up to $12.8 km.h^{-1}$, and indoor-outdoor capabilities, is a versatile platform for building a robot base (Nguyen *et al.* 2004). A computer can communicate with the Segway RMP's on-board controllers via a CAN Bus interface. This communication takes two forms: commands can be sent to the Segway RMP, and up-to-date state information can be read back from the Segway. The on-board controllers maintain the balance of the platform while attempting to achieve the forward and rotational velocity commands v_{cmd}, ω_{cmd}, which may be modified on our robot at $30Hz$. The Segway state information may be read at the same rate. The internal sensors, as relevant to this paper, consist of: Pitch angle and rate $(\theta_p, \dot{\theta}_p)$, wheel velocities (v_L, v_R), motor currents (I_L, I_R), forward displacement (S_o), and the sent velocity command (v_{cmd}, ω_{cmd}). The displacement is the path length travelled by the robot since it was started ie. by integrating the average wheel rotations. Thus, in the framework discussed in approach section, the sensor state is given by $z(\theta_p, \dot{\theta}_p, v_L, v_R, I_L, I_R, S_o, v_{cmd})^T$.

Dynamic balancing control means that the dynamics of the RMP are unique. To accelerate in a direction, the robot must tilt its center of gravity relative to the wheel axles towards that direction. It must then 'catch up' to itself to prevent falling over. The more extreme the acceleration the greater the tilt angle. As a result, when accelerating from standstill requires that the wheels first roll *backwards* to create the tilt, before rolling forwards to catch up (see figure 2).



(a)



(b)

Figure 2: (a) The RMP's velocity and pitch angle for starting and stopping (b) The RMP's pose definitions.

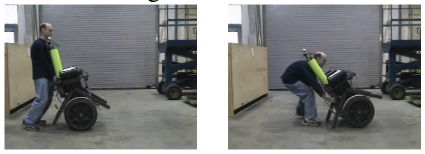
While the Segway RMP's controllers are capable, they assume that the robot is free to move, meaning there is sufficient free space to drive and maintain balance. If the robot hits an obstacle (e.g. because it fails to detect it), a crash usually results. When driving into an obstacle, the robot's velocity drops causing the controllers to command increased torque to correct the increased error between the commanded velocity and the actual velocity. This becomes a positive feedback loop with more torque causing more error and greater tilt angles. If the obstacle does not give way, the robot will eventually reach its angle limits of about $\pm30°$ and fall over. The force and weight of the robot make the resulting crash spectacular and can cause damage to the robot and anything in the surrounding area.

Identifying Failure States

We focus on the failures caused by running into obstacles. In our work with the Segway RMP, we have noticed that obstacles can be categorized into two classes based on their size. Large obstacles, such as walls, humans, tables, and other robots, cause the robot to pitch forward eventually reaching its angle limits as described above. We will call these type of failures, F_L.

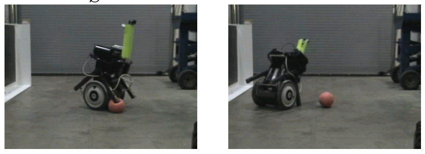
Small obstacles, on the other hand, tend to interfere with just one wheel or side of the robot. Typically, the robot will wedge itself up on top of the obstacle whereupon it spins out of control as the rotational velocity error builds up. As our main use of the Segway RMP is for robot soccer (Browning *et al.* 2004), this can cause serious difficulties. We will call these type of failures, F_S. Figure 3 shows two sequences of both F_S and F_L failures.

F_L Large Obstacle Failure



(a) (b)

F_S Small Obstacle Failure



(a) (b)

Figure 3: Images gathered from live video showing (a) Caution state (b) Failure state

Labeling the RMP's Sensor Data

We have the ability to record a stream of the Segway RMP's sensor data, $z_1, \ldots, z_T$, into a log file using our control software. This allows a user to record the sensor data stream while teleoperating the RMP into a particular failure state. The recording/teleoperating procedure can be repeated as desired to obtain data for describing the failure states. Figure 4 shows an example data stream for an F_L failure.

With these plots, a human operator can label where the data stream transitions from normal behavior, through a caution state to a failure.

The key in labeling when the transition began is the ability to observe sensor data from before, during, and after the caution state. The entire sequence is observable and picking out the transition is made simple. For example, when the RMP runs into a wall or other large obstacle, its overall displacement stops increasing. This one sensor stream alone is sufficient in identifying when the transition began. When capturing the logs, we knew that we would be moving forward before driving into the obstacle. This prior knowledge

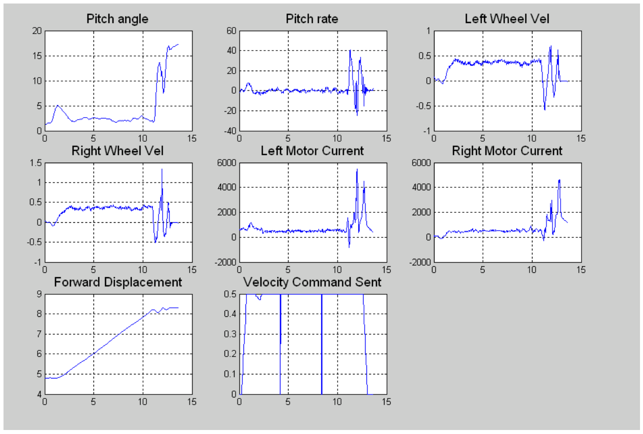


Figure 4: Plots of 8 RMP Internal Sensor Streams of an F_L Failure

and the limited time length of the log allows hand-labeling to become simpler.

A method used to reduce error in hand labeling was identifying two time steps corresponding to the transition. The first time step, t_N, signifies that all data before this time was definitively normal data. The second time step, t_F, signifies that all data after this time should be labeled as abnormal. It is difficult to know the exact time at which a caution state begins, but it is much easier to identify a small window where the transition began. This gives the labeler flexibility to choose a range which helps to ensure that the labeled data is correct.

Learning a Classifier

With hand labeled sensor data, it is possible to develop a decision tree by hand to identify caution states. However, hand-coding a decision tree for vectors of many dimensions is both a time consuming and error prone process. Indeed, we found that developing a hand coded decision tree produces a low-quality classifier at best. The difficulty arises in determining the right combination of sensors and exact threshold values needed to identify a caution state while not falsely classifying normal data as abnormal. This is where hand coded rules fail. They identify abnormal situations correctly, but they tend to falsely identify normal driving as abnormal at a much higher rate than the rules generated from learning. It is simply very difficult for a human to consider all the factors that go into recognizing a caution state.

There is a huge body of literature on classifiers with a plethora of different techniques available that can be learned from labeled data. We chose to use decision trees trained using the C4.5 algorithm (Quinlan 1993), an extension of ID3. C4.5 is able to learn compact pruned decision trees accounting for unavailable values, continuous attribute inputs, and avoids over fitting. We chose decision trees due to their fast run-time execution. This enables the use of the classifier at the lowest control levels of the robot running at $30Hz$ with minimal overhead costs. Secondly, C4.5 is able to learn useful trees from relatively small amounts of training data. Given the effort required to capture logs of robot failures, this is a very desirable feature.

From our knowledge of the domain and dynamics of the RMP, we modified the sensor state used for the classifier to be: $z = (\theta_p, \dot{\theta}_p, v_{avg}, I_L, I_R, v_{cmd})^T$. Here, the pitch angle and rate $(\theta_p, \dot{\theta}_p)$, motor currents (I_L, I_R), and sent velocity command (v_{cmd}), are as before. Instead of the individual wheel velocities we use the average forward speed (i.e. $v_{avg} = 1/2(v_L + v_R)$) as this reflects the information that is useful for detecting collisions. To avoid false-positives due to noise, a simple first order filter is applied to the data stream to smooth it out. Concretely, this filter is given as:

$$\hat{z}_t = \alpha * z_{t-1} + (1 - \alpha)\, \hat{z}_{t-1} \tag{1}$$

Where $\alpha = 0.25$ was the nominal value used. During operation, the filter output $\hat{z}_t$ is fed into the classifier and the result determines if the robot is in a dangerous situation. This filter is applied to the data used in training and to the the real-time data being classified. The filter introduces a lag in the response of the system. This can be described by a first order pole with a time constant of $T_{delay} = \frac{T}{\alpha} = 0.133s$. Where T is the time period between updates. For the segway updates are run at $30Hz$.

To create the training data for the classifier, the data stream was filtered into definitely normal, and definitely failure data sets. That is, the definitely normal data set was labeled as *normal* $D_N = z_1 \dots z_{t_N}$ while the definitely failed data set $D_F = z_{t_F} \dots z_T$ was labeled as *abnormal*. The remaining data, where labeling accuracy is uncertain, was not used for training. This data was then provided to the C4.5 learning algorithm to produce a pruned decision tree. The resulting pruned decision tree is converted into C++ code and incorporated into the robot's control loop.

Twenty logs of the RMP running into a large obstacle and falling over were used along with approximately three minutes of normal driving data to obtain the large obstacle classifier. Fourteen logs of the RMP running into a soccer ball and falling over along with approximately 3 minutes of normal driving data were used to obtain the small obstacle classifier. Furthermore, the amount of time spent collecting this data and labeling was only approximately two hours. The resulting classifier was well worth the minimal time commitment.

Taking Preventative Actions

When an impending failure is detected a safety response mechanism must be triggered. As this paper considers failures when the Segway RMP runs into an obstacle, the safety response mechanism reverses the direction of travel for a short time, before waiting for further commands. This response mechanism is simplistic, and certainly not foolproof, but is sufficient for the experiments described below. To be precise, the response behavior sends a command speed of $2m.s^{-1}$ with the sign set to the opposite of the last command sent to the robot before the failure. The robot reverses for a duration of $0.5s$ and then halts for $5s$ before switching back to normal operation. This speed command causes the robot to pitch backwards aggressively, pulling itself off the obstacle, but without traveling very far.

The failure detection mechanism was implemented at the low-level of the control architecture between the interface with the Segway RMP itself. In this way it can effectively block off commands when the robot is in danger and needs to execute the recovery. In the next section, we examine the performance of this approach on the Segway RMP.

Experimental Results

The following are the results of the implementation of our method on the Segway RMP for both large and small obstacle failures.

Classification Performance

We started with classifying the failure in which the RMP runs into a wall or other large obstacle, i.e. F_L. We hypothesized that this would be the easier of the two failures to classify. We thought with successful results on this failure, the smaller obstacle failure would be possible to prevent with the same procedure.

We determined that most of the error resulted from sensor data that was not explicitly classified as either normal or abnormal by the decision tree. This data by default is classified as normal. As a result, some of the data that was more extreme than the training data was classified as normal. This error is fixed by training on more data which contains most of the situations that will be experienced. Using this information, we obtained more logs and trained a final working classifier. The results of this classifier using 10-fold cross validation are shown in Figure 5 under Final F_L Results.

	Normal Correct	Abnormal Correct
F_L Results	92.6%	97.7%
F_S Results	91.5%	97.9%

Figure 5: Percentage correct classification for normal and abnormal cases for large and small obstacles.

With successful detection of F_L, we used the same approach to classify F_S. Figure 5 shows the results of the F_S classifier using 10-fold cross validation.

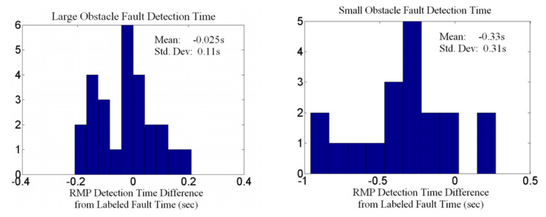
Failure Prevention Results

In implementation, we used the trained classifiers described in the previous section to continuously identify whether the robot was in a normal state or not.

If abnormal data was recognized a caution state was identified and the Segway RMP triggers the safety response behavior described earlier. Additionally, it became necessary in practice to filter the output of the classifier to require a small number of sequential abnormal classifications before triggering the safety behavior. Five sequential abnormal classifications were required to trigger the response. This final filtering was required to prevent undue misfires of the safety mechanism from single false-positive classifications. For example when the robot accelerates quickly from rest, the classifier can sometimes report one or two frames of abnormal behavior but not a long stream of abnormal behavior. The final number was obtained through experimentation to eliminate false positives in extensive testing. With this filtering mechanism no misfires of the safety behavior were observed as reported below.

For a dynamically balancing robot, speed of detection is critical to being able to avoid the impending failure. As explained earlier, it is very difficult to identify exactly when the RMP transitions from a normal state to an abnormal state. In hand labeling, two transitions were identified, the first signifies that all data before was normal while the second signifies that all data afterwards is abnormal. We conducted tests to characterize the speed of the detection by determining the time in which the classifier identifies an abnormal state relative to the second hand-labeled step (i.e. to t_F).

We performed 10-fold cross validations on both failure types. Figure 6 shows a histogram of the detection speeds for each type of failure, F_L and F_S respectively. The results show the distribution over time of the autonomous caution state detection relative to the second label. A negative detection time corresponds to the classifier detecting the impending failure faster than compared to the human-labeled transition. As seen, both classifiers on average perform better than human labeling.



(a) (b)

Figure 6: (a)Large Obstacle Detection Time Results, F_L (b) Small Obstacle Detection Time Results, F_S

Tests were also conducted to evaluate the overall effectiveness of failure prevention for the Segway RMP driving in a flat, but cluttered environment, as shown in figures 3 and 7. We teleoperated the robot, with the failure prevention system running, into a soccer ball obstacle 30 times and a large obstacle an additional 30 times. This test also included considerable driving in free space to observe in any false-positives misfires would occur. For the total 60 collisions, the safety response behavior triggered correctly and prevented the robot from falling in each case. The robot would hit the obstacle, the failure mechanism would activate and drive the robot backwards out of the situation. Additionally, no false-positive misfires were observed during driving in free space. As a result, the robot could be driven around at random without fear of falling over. Figure 7 shows an image sequence of the working RMP failure prevention. The RMP drives into an obstacle, recognizes the obstacle, and safely backs away.

Related Work

Failure detection is a rich field (e.g. (Gertler 1998)). For mobile robots, most approaches focus on model-based methods where a model of the robot is used to predict how it

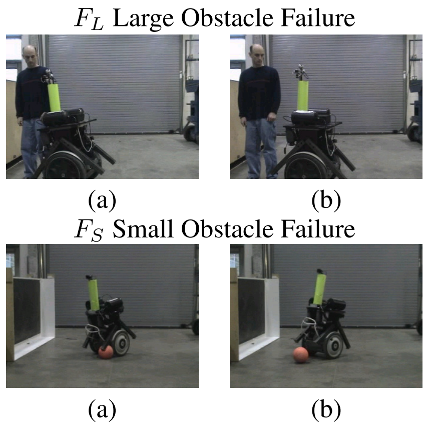


Figure 7: Images gathered from live video showing (a) Caution state (b) Successful failure prevention

transitions in the world (i.e. $P(s', a, s)$ in the MDP – see the approach section). Using this model, predictions can be made about behavior of the robot which can be compared against sensor readings to determine a residual. The residual can then be classified to determine if the robot is failing.

Generally, four methods for residual generation have been explored in the literature. These include: parameter estimation, Kalman filters, diagnostic observer, and parity relation (Gertler 1998; Yan 2003). Parameter estimation techniques vary but include the use of neural networks (Yan 2003) as classifiers and function approximation methods (Polycarpou & Vemuri 1995), additionally particle filter based methods have become recently popular (Dearden *et al.* 2004; Dearden & Clancy 2002).

Model-based approaches differ from our method, where no model is required, although one is assumed to exist. The requirement for a detailed model can be limiting when no such models are available. For many commercial robots, such as the Segway RMP and Sony AIBO, no detailed model exists. Additionally, as the models are a function of the environment, changes in the environment that are poorly accounted for in the model degrades its accuracy and usefulness.

There are a number of related model-free techniques in the literature. In particular, (Vail & Veloso 2004) makes use of C4.5 to learn state classifications for the Sony AIBO robot using accelerometer data. This work uses a classifier to determine the walking surface of the robot and whether or not the AIBO has fallen over (Vail & Veloso 2004). Thus, this work is an extension of this idea to include failure prevention rather than recovery. State classification by other means is also possible, as in (Lenser & Veloso 2004), and this constitutes a future direction for our research.

Future Work

Future work will concentrate on eliminating the need for a human to label the internal sensor data. The data can be continually logged and data relating to a bad situation such as running into a wall can be identified when compared to normal driving. This can then be used to obtain a classifier to identify the onset of future failure states. With this infrastructure, the classifier can be continually updated leading to a better classifier over time.

Conclusions

This paper has contributed a novel approach to detect failures from internal sensor information for an autonomous robot. Our approach utilizes a decision-rule classifier, trained from a small set of hand labeled data. The classifier detects states that are potentially dangerous and triggers a failure prevention mechanism. We fully implemented this approach on a dynamically balancing Segway RMP robot to prevent it from falling when it collides with external obstacles. The procedure takes minimal time to implement and the results clearly demonstrate the usefulness of the approach in improving the robustness of the robot.

References

Browning, B.; Rybski, P.; Searock, J.; and Veloso, M. 2004. Development of a soccer-playing dynamically balancing mobile robot. In *Proceedings of ICRA*.

Dearden, R., and Clancy, D. 2002. Particle filters for real-time fault detection in planetary rovers. In *Proceedings of the International Workshop on Principles of Diagnosis*.

Dearden, R.; Huttner, F.; Simmons, R.; Verma, V.; Thurn, S.; and Willeke, T. 2004. Real-time fault detection and situational awareness for rovers: Report on the mars technology program task. In *Proceedings of IEEE Aerospace Conference*.

Gertler, J. 1998. *Fault Detection and Diagnosis in Engineering Systems*. NY: Marcel Dekker, Inc.

Lenser, S., and Veloso, M. 2004. State identification from robot sensors using non-parametric statistics. In *Proceedings of the International Conference on Intelligent Robots and Systems*.

Nguyen, H.; Morrell, J.; Mullens, K.; Burmeister, A.; Miles, S.; Farrington, N.; Thomas, K.; and Gage, D. 2004. Segway robotic mobility platform. In *SPIE Proc. 5609: Mobile Robots XVII*.

Polycarpou, M., and Vemuri, A. 1995. Learning methodology for failure detection and accommodation. *Control Systems Magazine, IEEE* 15(3):16–24.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Interscience.

Quinlan, J. 1993. *Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Vail, D., and Veloso, M. 2004. Learning from accelerometer data on a legged robot. In *Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*.

Yan, W. 2003. Fault detection and multi-classifier fusion for unmanned aerial vehicles (uavs). In *GE Global Research Technical Report*.