Fall Prediction for New Sequences of Motions

Junyun Tay*[1,2], I-Ming Chen[1], and Manuela Veloso[2]

[1] Nanyang Technological University, Singapore 639798, SINGAPORE
[2] Carnegie Mellon University, Pittsburgh, PA 15213, USA
junyunt@andrew.cmu.edu
michen@ntu.edu.sg
veloso@cs.cmu.edu

Abstract. Motions reinforce meanings in human-robot communication, when they are relevant and initiated at the right times. Given a task of using motions for an autonomous humanoid robot to communicate, different sequences of relevant motions are generated from the motion library. Each motion in the motion library is stable, but a sequence may cause the robot to be unstable and fall. We are interested in predicting if a sequence of motions will result in a fall, without executing the sequence on the robot. We contribute a novel algorithm, ProFeaSM, that uses only body angles collected during the execution of single motions and interpolations between pairs of motions, to predict whether a sequence will cause the robot to fall. We demonstrate the efficacy of ProFeaSM on the NAO humanoid robot in a real-time simulator, Webots, and on a real NAO and explore the trade-off between precision and recall.

Keywords: fall prediction, sequence of motions, humanoid robot

1 Introduction

Research showed that as much as 70% of meanings in communication is derived from non-verbal behavior such as gestures and other body movements [1]. To enable effective human-robot interaction, we leverage on the use of motions for an autonomous task-performing humanoid robot to complement the traditional form of communication, e.g., text-to-speech. In our previous work, we contribute algorithms to enable humanoid robots to autonomously dance to the beats and emotions of music [2] and animate speech with gestures [3]. Motions are meaningful when they are relevant and initiated at the right times: a switch to a new motion is cued by a change in a task. For example, an autonomous dancing humanoid robot plans for a switch in the dance movement according to the change in emotions of the music [2], and a storytelling robot plans to switch to a pointing motion to refer to a specific object [3] given the cue of the object.

From our previous experiences, we observe that several sequences of motions are generated to express the same meaning but only one sequence of motion can

* Junyun Tay is in the Nanyang Technological University-Carnegie Mellon University (NTU-CMU) Dual Degree PhD Programme in Engineering (Robotics).

be executed by the robot at a time. However, not all the sequences of motions are stable. We are interested in predicting the stability of a sequence of motions before executing it on the robot. As a motivating example, we consider the task of an autonomous humanoid robot playing a game of charades. In our library of motions, each motion is labeled with meanings expressed by the motion. For example, a motion where the robot waves its arms in the air is labeled with the word "happy." Each label can be mapped to multiple motions, e.g., two motions are labeled with the word "sad." Likewise, each motion can be mapped to multiple labels. There is a sequence of words to play in the game of charades, and for every word, several motions express its meaning. Hence, multiple sequences of motions that are relevant and synchronized to the task are generated. We need to determine the sequences that will cause the robot to fall. In particular, we consider the case where each motion in the library is stable by itself for execution on the robot, but a sequence of motions may result in a fall.

We contribute an algorithm, ProFeaSM, that predicts if a sequence of motions will fall. Our approach is novel as conventional methods to check for stability require an accurate model of the robot. We do not require any model of the robot to perform the prediction. Motion planning methods for stability modify the sequence of motions (e.g., timings and motion trajectories), but these methods are not applicable for our problem as the motions have to be synchronized with the task and reflect the meanings. Existing fall avoidance approaches also cannot be applied as they require training instances of the sequences for their algorithm, whereas ProFeaSM does not require execution of any sequence.

Our algorithm, ProFeaSM, is based on the concept of momentum and inertia. We predict how the body angles change as the sequence of motions is executed. The body angles are computed using the gyroscope and accelerometer sensors, which are typically found in humanoid robots. ProFeaSM uses the body angle values from the executions of single motions and interpolations between pairs of motions to derive the velocity and acceleration of the body angles. ProFeaSM then uses these velocities and accelerations to determine how the body angles will change during a sequence of motions. We design ProFeaSM to include a parameter we can vary to explore the trade-off between precision and recall. With a threshold of the body angle, we can determine if the robot will fall from executing the sequence of motions. We show the efficacy of ProFeaSM using a complex real-time simulator, Webots, and also on a real NAO humanoid robot.

We discuss related work and highlight the differences of this work in Section 2. We explain our algorithm in Section 3. We describe our experimental setup and analyze the results in Section 4. Lastly, we conclude our work in Section 5.

2 Related Work

We want to prevent falls for a humanoid robot as it takes time to get up and may cause the robot to break or cause more serious wear and tear. To predict a fall, we can determine the dynamic stability with a model of the robot and its environment and test if a sequence of motions is stable in simulation. However,

it is often difficult to have an accurate model to predict stability reliably given that it is difficult to model variables such as friction, slippage and wear and tear.

Conventional motion planners for stability require an accurate model of the robot and its environment and generating dynamically balanced motion for humanoid robots is challenging with the high number of degrees of freedom and "the size of the space to explore is augmented with the robot velocity and footprint positions." [4] Geometric paths can be planned by approximating a dynamic trajectory [4]. However, the drawback of this method is that "some feasible dynamic motions are inherently impossible to compute with this approach." [4] Though there are planners that will compute dynamically stable motion trajectories offline [5],[6], these planners require an accurate robot model and change the desired motion in terms of the timings and the trajectory. Changing the timings of the desired motion causes the motion to be no longer synchronized to the task and changing the motion trajectory changes the meaning expressed.

Falls that can cause the robot to break or serious wear and tear can be predicted and is generally done through online monitoring methods. These online monitoring methods predict falls by thresholding relevant physical quantities (e.g., angular momenta) or determine stability by tracking the position of the center of pressure (CoP) such that the CoP stays within the support polygon. Others analytically model the robot's dynamics to determine if a fall will occur. However, these methods do not scale well to humanoid robots with complex geometries and high degrees of freedom. Since it is difficult to model real world variables such as friction and wear and tear etc, a data-driven approach is used to collect sensor data of stable and unstable trajectories and classified to determine if a fall will occur [7],[8]. The prediction is done during the execution before a fall occurs. For instance, the robot's internal sensors are monitored over time to detect the onset of a robot's failure by using supervised learning techniques and create a classifier to determine failures for a dynamically balancing Segway RMP [11]. Our approach is to use sensor values from previous executions of motions and predict if a sequence is stable offline, without execution or monitoring on the robot. We also do not require a model of the robot or its environment.

Falls can be avoided using reflex motions [9][10] or to execute a controlled falling motion [7]. However, the desired motion trajectory is changed by executing reflex motions or a controlled falling motion, which also results in the change of the meanings expressed. We want to execute sequences of motions on the robot without falling. Fall avoidance methods are also triggered at a time where the motions are in the midst of execution and only slightly before the fall occurs. Though they can reduce the probability of the robot breaking, these methods cannot predict falls before execution and require training data of instances of the robot falling. Fall prediction methods can also be falsely triggered if the training data is insufficient or not executing controlled falling motions in time to prevent bad consequences. Hence, offline predictions of falls without execution will be better than trying to reduce the impact of a fall. Fall avoidance should only be used as a last resort to avoid significant damage to the robot when a fall occurs.

3 Technical Approach

We formalize the problem of predicting the stability of a sequence of motions in Section 3.1 and describe our algorithm in Section 3.2. We define a motion as a motion primitive and describe the components of a motion primitive and how a sequence of motion primitives is determined. We also explain how the list of possible combinations of sequences is derived. We use the task of an autonomous humanoid robot playing a game of charades as a motivating example.

3.1 Formalization

Let D denote the number of generalized coordinates or degrees of freedom (DOF) of the robot R and let C be the D-dimensional configuration space of R. A keyframe $k \in C$ is a vector of D real numbers specifying values for each of the joint angles for the respective degrees of freedom (joint) of R. A keyframe $k \in C$ is valid if it is collision-free and the joint angles stay within joint limits.

A motion primitive contains a series of keyframes (static poses) and durations [3]. Each duration is the time to interpolate (move) between two keyframes. A sequence of motion primitives starts with an initial pose and contains a list of motion primitives that expresses the meanings required of the task.

Definition 1. *A **motion primitive** P is a tuple of N primitives and is defined as $P = (M_1, \ldots, M_N)$ and $N \in \mathbb{Z}^+$. The primitive M_n is a tuple of 2 keyframes, k_{n-1} and k_n, and the time to interpolate between these two keyframes, $t_{n-1,n}$, where $M_n = (k_{n-1}, t_{n-1,n}, k_n)$. k_0, the first keyframe in M_1 is the initial pose of the robot, R, which contains all the joint angles for D degrees of freedom. Let P be the set of all motion primitives in the motion primitive library.*

To interpolate between two keyframes, the interpolation method, e.g., linear interpolation or bezier interpolation, is defined. We assume the trajectories to interpolate between the keyframe are generated by a motion planner and will fulfill the following conditions:

1. be collision-free
2. be within physical capabilities (joint angular and velocity limits)

The time to interpolate between two keyframes, k_n and k_{n+1}, is computed by the interpolation time computation function $T : C \times C \to \mathbb{R}^+$. $t_{n,n+1}$ specifies the minimum duration required to interpolate from the angles of the respective joints in k_n to the angles defined in k_{n+1} or is pre-defined by the motion choreographer. If the minimum duration required to interpolate from one keyframe to another is longer than the time pre-defined by the motion choreographer, the minimum duration will be used for $t_{n,n+1}$. The minimum duration depends on the interpolation method used and is determined using the maximum joint angular velocities. Therefore, to compute $t_{n,n+1}$, we use $t_{n,n+1} = T(k_n, k_{n+1})$.

Definition 2. *A **sequence of motion primitives** $\mathbb{S}$ consists of L sequence primitives, where $\mathbb{S} = (\mathbb{P}_1, \ldots, \mathbb{P}_L)$, $L \in \mathbb{Z}^+$. Let S be the set of all possible sequences of motion primitives. $\mathbb{P}_l$ is a tuple containing 2 motion primitives, P_{l-1} and P_l; $t_{l-1,l}$ denotes the time to interpolate between P_{l-1} and P_l (time to interpolate between the last keyframe of P_{l-1} and the first keyframe of P_l).*

Similarly, $t_{l-1,l}$, the minimum time to interpolate between P_{l-1} and P_l, is computed by the interpolation time computation function T. The joint angles of the initial pose of the robot R is defined by the first keyframe k_0 of the first primitive M_1, in the first motion primitive P_0, in the sequence primitive $\mathbb{P}_1$, in the **sequence of motion primitives** $\mathbb{S}$.

We aim to predict if a sequence of motion primitives $\mathbb{S}$ will cause the robot to fall without the robot executing the sequence, and without having a model of the robot and the environment. Every sequence in the list of possible combinations of sequences starts with the same initial pose.

3.2 Algorithm

Before the prediction of whether a sequence of motion primitives will fall, we record the body angles of the robot whilst executing each motion primitive in the library and the interpolations between pairs of motion primitives. We explain how our algorithm works using the body angles recorded to determine the stability of a sequence of motion primitives.

Body angles are recorded at a regular frequency f. For example, we execute a motion primitive and the b body angles are $(t_0, s_0), \ldots, (t_b, s_b)$, where t_i is the timestamp, s_i consists of the body angles at time t_i, and s_0 are the body angles of the initial pose. s_i comprises of the body angle X and Y readings per time step. b is calculated using the duration of the execution, d, using the equation $b = (d \times f) + 1$ as we also record the body angles for one time step before the motion primitive is executed.

We collect the body angles of three different groups of executions:

1. single: Each motion primitive is executed individually with the initial pose of the start of every sequence as the first keyframe, followed by the keyframes in the motion primitive. We denote single_{P_i} as the body angles collected for the motion primitive P_i.
2. startSingle: Each motion primitive is executed individually with its first keyframe (typically different from the initial pose of Fig. 1a). We denote startSingle_{P_i} as the body angles collected for the motion primitive P_i.
3. interpolation: We find all possible pairs of motion primitives, P_i and P_j, where $i \neq j$ and execute the interpolation between each pair of motion primitive. The interpolation between two motion primitives, P_i and P_j, is done with the execution of the last keyframe K_n of the first motion primitive P_i, and the first keyframe K_1 of the second motion primitive P_j. We denote $\text{interpolation}_{P_i,P_j}$ as the body angles collected for the interpolation between the two motion primitives, P_i and P_j.

We collect m iterations of each of the three groups of executions. With these three groups of body angles, we use our algorithm ProFeaSM to predict if a particular sequence of motion primitives, $\mathbb{S}$, will cause the robot to fall. With m iterations of each of the three groups and $|\mathsf{P}|$ number of motion primitives, we collect a total of $m \times (|\mathsf{P}| + |\mathsf{P}| + |\mathsf{P}|(|\mathsf{P}| - 1)) = m \times |\mathsf{P}|(|\mathsf{P}| + 1)$ executions.

Our algorithm ProFeaSM is made up of four algorithms, namely Process, Feasibility, Stitch and Multiplier (Algorithms 2-5). We process the body angles of the three groups of executions if we collect more than 1 iteration of the three groups of executions using Algorithm 2.

Algorithm 1 ProFeaSM: Process-Feasibility-Stitch-Multiplier

ProFeaSM($\mathbb{S}$, inertialMultiplier, single, startSingle, interpolation)

$(m, n) \leftarrow size$(single)
if m = 1 **then**
  hasFallen $\leftarrow$ Feasibility($\mathbb{S}$, inertialMultiplier, single, startSingle, interpolation)
else
  single $\leftarrow$ Process(single)
  startSingle $\leftarrow$ Process(startSingle)
  interpolation $\leftarrow$ Process(interpolation)
  hasFallen $\leftarrow$ Feasibility($\mathbb{S}$, inertialMultiplier)
end if
return hasFallen

Next, we process the m iterations collected of each of the three different groups of executions by determining the median for body angle X, $\mathcal{M}_{\text{bax}}$, and body angle Y, $\mathcal{M}_{\text{bay}}$. Process (Algorithm 2) returns the median of a trajectory of body angles (X or Y), given ba, a list of m body angle trajectories. For example, $\mathcal{M}_{\text{bax}} =$ Process(bax), where bax contains the m trajectories of body angle X. We use these median body angles to predict if a sequence of motion

Algorithm 2 Process m iterations of b time steps

Process(ba)

{ba is a $m \times b$ matrix, containing m iterations with b time steps}
for $i = 1$ **to** b **do**
  medianAtEachStep(i) = median$_{j=1}^{m}$(ba(j, i)) {finds median at time step i}
end for
medianBA = argmin$_{j=1}^{m}$($\sum_{i=1}^{b}$ |ba(j, i) $-$ medianAtEachStep(i)|)
return medianBA

primitives, $\mathbb{S}$, will fall using Algorithm 3, Feasibility. Algorithm 3 requires two parameters, $\mathbb{S}$ and inertialMultiplier.

Definition 3. *The algorithm Feasibility* : $\mathsf{S} \times \mathbb{R} \rightarrow \{0, 1\}$ *computes the* ***feasibility*** *of a sequence of motion primitives, where a sequence of motion primitives is* ***feasible*** *if and only if the robot is able to execute the keyframes whilst being* ***stable****. Hence, Feasibility*($\mathbb{S}$, *inertialMultiplier*) $= 1$ *when* $\mathbb{S}$ *is feasible.*

Algorithm 3 Predict whether a sequence of motion primitives is feasible

Feasibility($\mathbb{S}$, inertialMultiplier, single, startSingle, interpolation)

1: {Indices start from 1}
2: data $\leftarrow$ Stitch($\mathbb{S}$, single, startSingle, interpolation)
3: $v \leftarrow (0, \text{data}(2) - \text{data}(1), \text{data}(3) - \text{data}(2), \ldots)$
4: $a \leftarrow (0, v(2) - v(1), v(3) - v(2), \ldots)$
5: predictTraj $\leftarrow \text{single}_{P_1}$
6: stepMultiplier $\leftarrow 0$ {initialized as 0 as $e^0 = 1$}
7: **for** $i = 1$ **to** $|\text{single}_{P_1}|$ **do**
8: stepMultiplier $\leftarrow$ Multiplier($a(i)$, stepMultiplier, inertialMultiplier)
9: **end for**
10: hasFallen $\leftarrow$ false
11: **for** $i = |\text{single}_{P_1}| + 1$ **to** |data| **do**
12: predictAngle $= v(i) \times \exp(\text{stepMultiplier}) + \text{predictTraj}(i-1)$
13: stepMultiplier $\leftarrow$ Multiplier($a(i)$, stepMultiplier, inertialMultiplier)
14: predictTraj $\leftarrow$ append(predictTraj, predictAngle)
15: **if** |predictAngle| > fallenThresh **then**
16: hasFallen $\leftarrow$ true
17: **end if**
18: **end for**
19: **return** hasFallen

Algorithm 3 is based on the concept that as the acceleration of the body angles increases, the velocity increases and vice versa. As the acceleration of the body angles approaches zero, the velocity of the body angles reaches a constant. The body angle indicates the angle of the robot's torso with respect to the ground. Therefore, the higher the body angle, the higher the possibility that the robot is going to fall. To determine the velocity, v, we determine the change in body angles at each time step. To determine the acceleration, a, we calculate the change in velocity at each time step. Since the velocity will not increase linearly due to the effect of gravity, inertia and momentum, we model the velocity as an exponential curve in Algorithm 3. The acceleration will affect how far we are along the exponential velocity curve, hence changing the x-value of the exponential curve, which we term as stepMultiplier.

Algorithm 4 (the Stitch function called in Line 2 of Algorithm 3) explains how we stitch up the body angle values collected. We always begin with the original body angle trajectory of the first motion primitive in the sequence, single_{P_1}. Next, we determine the change in body angle and add the change to the last known body angle for the rest of the sequence using the change in the body angles of interpolation and startSingle.

Lines 3 and 4 in Algorithm 3 shows how we determine v, the velocity of the body angles and a, the acceleration. To predict the body angle trajectory, predictTraj, during the execution of the sequence of motion primitives, we begin with the body angles collected from single_{P_1} since the body angles should be similar to executing the motion primitive single_{P_1} that starts from the initial pose. However, we still have to determine the stepMultiplier using Algorithm 5

Algorithm 4 Stitch collected data into a trajectory

Stitch($\mathbb{S}$, single, startSingle, interpolation)

1: data $\leftarrow$ single$_{P_1}$
2: lastAngle $\leftarrow$ single$_{P_1}$(|single$_{P_1}$|)
3: **for** $l = 2$ **to** L **do**
4: **for** $i = 2$ **to** |interpolation$_{P_{l-1},P_l}$| **do**
5: lastAngle $\leftarrow$ lastAngle + (interpolation$_{P_{l-1},P_l}(i)$ − interpolation$_{P_{l-1},P_l}(i-1)$)
6: data $\leftarrow$ append(data, lastAngle)
7: **end for**
8: **for** $i = 2$ **to** |startSingle$_{P_l}$| **do**
9: lastAngle $\leftarrow$ lastAngle + (startSingle$_{P_l}(i)$ − startSingle$_{P_l}(i-1)$)
10: data $\leftarrow$ append(data, lastAngle)
11: **end for**
12: **end for**
13: **return** data

since the inertial and momentum are changing in Lines 7-9 (Algorithm 3). As we predict the body angle trajectory, predictTraj, we predict the velocity using the exponential velocity curve and stepMultiplier so as to determine the change to the previous body angle in Line 12 (Algorithm 3). Next, we continue to change stepMultiplier in Line 13 (Algorithm 3) and append the predicted body angle, predictAngle, to predictTraj. We check if predictAngle exceeds fallenThresh, a threshold to determine if the robot has fallen in Lines 15-17 (Algorithm 3).

Algorithm 5 is used to determine how stepMultiplier varies along the exponential velocity curve. As the computed acceleration of the body angles per time step is very small, inertialMultiplier is used as a multiplier to the acceleration, and varies how stepMultiplier changes in Line 4 of Algorithm 5. accThres is used as a threshold to determine if the acceleration approaches zero and if so, stepMultiplierDec is used to decrease stepMultiplier in Lines 1-2 of Algorithm 5.

Algorithm 5 Determine the step multiplier based on the acceleration

Multiplier(a, stepMultiplier, inertialMultiplier)

1: **if** $|a| <$ accThres **then**
2: stepMultiplier $\leftarrow$ stepMultiplier − stepMultiplierDec
3: **else**
4: stepMultiplier $\leftarrow$ stepMultiplier + ($a \times$ inertialMultiplier)
5: **end if**
6: **if** stepMultiplier < 0 **then**
7: stepMultiplier $\leftarrow 0$ {stepMultiplier will not go below 0}
8: **end if**
9: **return** stepMultiplier

To summarize, ProFeaSM comprises of Algorithm 2–5. With m iterations of body angles recorded, we use Algorithm 2 to determine the median body angle

trajectories. Next, we use Algorithm 3 to predict the stability of a sequence of motion primitives. Algorithm 3 uses Algorithm 4 to stitch up the body angles collected using the body angles collected from the three groups of executions and their respective velocities of the body angles. Algorithm 3 also uses Algorithm 5 to determine the multiplier, stepMultiplier, in the exponential velocity curve.

4 Experiments

We describe the experiments carried out in simulation and on a real NAO humanoid robot in Section 4.1 and discuss our results in Section 4.2.

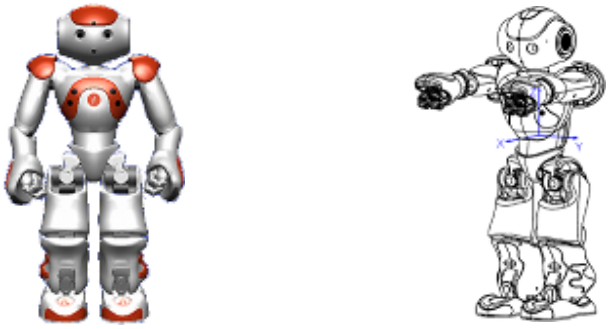
4.1 Experimental Setup

Experiments are conducted in simulation using Webots 7 [12] and on a real NAO humanoid robot. Webots 7 [12] is a real-time simulator that simulates the dynamics of the NAO humanoid robot's whole body executing a sequence of motion primitives. For the experiment in simulation, we simulate a NAO V4.0 H25 humanoid robot and conduct $m = 10$ iterations to collect body angle values for three groups of executions: single, startSingle, interpolation. Lastly, we test if each sequence of motion primitives is stable. We also ran 10 iterations to ensure that each motion primitive in the library is stable. Webots is restarted each time an iteration is ran to ensure that the NAO humanoid robot begins with the same initial pose and position in the environment.

To test if our algorithm, ProFeaSM, can be applied to a real NAO, we use a NAO V3.3 H21 humanoid robot, with a V4.0 head. We use a different model of the NAO from the simulation to enable us to test if ProFeaSM can be applied to different robots. We conduct $m = 1$ iteration to collect the three groups of executions as it is impractical to collect many iterations in real life, so we wanted to evaluate ProFeaSM when $m = 1$. Though the single motion primitives in the motion primitive are stable, we do not guarantee that the interpolations between the pairs will be stable. If we execute multiple iterations of interpolations that fall, it may break the robot. Hence, we allow the robot to fall gently by tying a thread around the robot's torso to prevent the robot from hitting the ground too hard. We also test the stability of every sequence on the real NAO. We stop the execution of the rest of the sequence when the robot has fallen. The collection of the body angles for the three groups of executions is unaffected since only interpolations may fall and the interpolations are short as compared to the entire sequence. The NAO robot's software includes a fall manager [13], that detects a potential fall during execution and prevents itself from breaking by putting its arms before its face before touching the ground. However, we observe that the fall manager is often triggered prematurely though the motion primitive is stable. Hence, we disable the fall manager during the collection of the body angles and the execution of the sequences.

Each sequence of the motion primitive starts with the same initial pose as shown in Fig. 1a. The body (torso) angles are recorded at a frequency of 100

Hz (every 10 milliseconds) using a function provided by the NAO's software [13] and computed using the accelerometer and gyrometer sensors readings from the inertial measurement unit (IMU) [13]. The body angles recorded are body angle X (roll) and Y (pitch) as shown in Fig. 1b.



(a) Initial pose of the NAO (b) Coordinate frame of IMU [13]

Fig. 1: NAO's initial pose and IMU's coordinate frame

For our experiments, fallenThresh in Algorithm 3 is 1.0 based on the body angle we observe when the robot is lying on the ground. accThres in Algorithm 5 is set to 0.005 and stepMultiplierDec to 0.001 respectively as the 0.005 is close to 0, and 0.001 only changed the multiplier slightly. With 1 iteration of body angles collected for the three groups of executions for the real NAO experiment, we skip Algorithm 2 to determine the median body angle trajectory and use the body angles recorded to predict the fall of the sequences. We vary different values of inertialMultiplier in Algorithm 5 from 10 to 100.

The task of the NAO humanoid robot is to play a game of charades to guess emotions. There are three different emotions: angry, sad and surprised. The robot can choose to act out the emotions in any order. In our library of motion primitives, for every emotion, there are two motion primitives that are labeled with the particular emotion. The robot is able to successfully execute each individual motion primitive without any falls. The number of combinations of sequences for three different motions (angry, sad, surprised in any order) is $2 \times 2 \times 2 \times 3! = 48$. For the three groups of executions: single, startSingle, interpolation, we collect a total of $|\text{single}| + |\text{startSingle}| + |\text{interpolation}| = 6 + 6 + (6 \times 4) = 36$ body angle trajectories. We do not need to collect all the $6 \times 5 = 30$ body angle trajectories for interpolation as there are two motion primitives per emotion and we do not use two motion primitives labeled with the same emotion consecutively.

To compute the interpolation time between keyframes, we use the interpolation time computation function T that uses the maximum joint angular velocity. We limit the maximum joint angular velocity in the simulation to be 70 percent of the real maximum joint angular velocity in simulation and 40 percent of the real maximum joint angular velocity on the real NAO so as to ensure that each motion primitive in the library is stable.

4.2 Experimental Results

Table 1 shows two sequences of motion primitives: (a) Sad2, Angry2, Surprised1, (b) Surprised1, Sad2, Angry2. The first row shows the intended sequence of the motion primitives (shown in bold) and the interpolations between motion primitives. "Start-" indicates the interpolation from the initial pose of the robot to the first motion primitive.

We demonstrate that even though each motion primitive in our motion primitive library is stable, a sequence of individual stable motion primitives does not guarantee the robot's stability in Table 1 as the sequence of Surprised1, Sad2 and Angry2 results in a fall. Though the sequence of Surprised1, Sad2 and Angry2 is unstable, a different ordering of the motion primitives, Sad2, Angry2 and Surprised1 is stable. From Table 1, we may deduce that the sequence from Sad2 and the interpolation between Sad2 and Angry2, and Angry2 causes the fall, but the sequence from Sad2 to Angry2 in the sequence of Sad2, Angry2 and Surprised1 is stable. Hence, we cannot predict the fall of the robot based solely on part of the sequence, but we have to consider the entire sequence.

Table 1: Intended and Actual Execution of Two Sequences of Motion Primitives

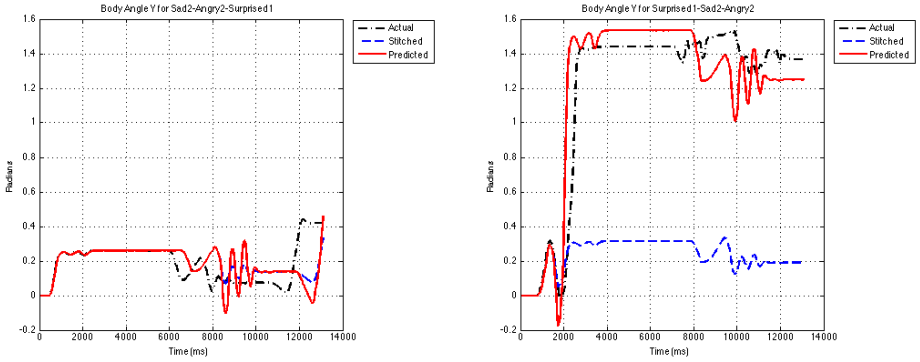
Intended	Start-Sad2	**Sad2**	Sad2-Angry2	**Angry2**	Angry2-Surprised1	**Surprised1**
Actual	Start-Sad2	**Sad2**	Sad2-Angry2	**Angry2**	Angry2-Surprised1	**Surprised1**
Intended	Start-Surprised1	**Surprised1**	Surprised1-Sad2	**Sad2**	Sad2-Angry2	**Angry2**
Actual	Start-Surprised1	**Surprised1**	Surprised1-Sad2	***Fallen***	***Fallen***	***Fallen***

In our experiments, we observed that body angle Y values is sufficient for predicting if the robot will fall, as the robot only falls forward or backward and never sideways. Hence, we present results regarding body angle Y values since we only use body angle Y values to predict if a sequence will fall.

Fig. 2 shows two plots: Fig. 2a shows the body angle Y values of the sequence, Sad2, Angry2 and Surprised1 over time, and Fig. 2b shows the body angle Y values of the sequence, Surprised1, Sad2 and Angry2 over time. Fig. 2a shows a sequence of motion primitives that does not fall whereas the right plot, Fig. 2b, shows a sequence of motion primitives, Surprised1, Sad2 and Angry2 that causes the robot to fall. Both sequences are executed in simulation and the prediction of the body angle trajectories are made from the body angles collected in simulation. For both figures in Fig. 2, we plot three body angle Y trajectories: First, we plot the body angle Y trajectory that was collected during the execution of a sequence in black with a line style of $-\cdot-$ (Actual). Next, we plot a stitched body angle Y trajectory in blue with a line style of $--$ (Stitched) using only Algorithm 4. Lastly, we plot the predicted body angle Y values using Algorithm ProFeaSM in red with a line style of — (Predicted).

In Fig. 2a, the actual, stitched, and predicted body angle Y trajectories are similar. However in Fig. 2b, we show that the predicted body angle Y is similar to the actual body angle Y trajectory, while the stitched body angle Y trajectory is not. Thus, we cannot simply stitch up body angles collected. We demonstrate

that our algorithm works well in predicting the body angle trajectory given that the curvature of the trajectory is similar.



(a) Sad2-Angry2-Surprised1 (b) Surprised1-Sad2-Angry2

Fig. 2: Body Angle Y values for Two Sequences of Motion Primitives

After testing our algorithm, ProFeaSM, on the body angles collected from the single motion primitives and the interpolations between pairs of motion primitives in simulation, we vary the parameter of ProFeaSM, inertialMultiplier, to analyze if the accuracy of our fall prediction improves. We refer to the use of precision and recall for classification tasks [14]. Similar to this task, whereby we want to classify sequences that are unstable as falls. Precision is the number of true positives (sequences that we label as falls and will actually fall during the execution) divided by the sum of true positives and false positives (sequences that we label as falls but will not fall during the execution). A perfect precision score of 1.0 means that every sequence that ProFeaSM labeled as a fall actually did fall during the execution of the sequence. Recall is the number of true positives divided by the total number of sequences that actually fall during the execution. A perfect recall score of 1.0 means that every sequence that actually fell during the execution was labeled as a fall by ProFeaSM, but does not consider sequences that are wrongly labeled as falls. Precision and recall often has an inverse relationship whereby increasing one decreases another. We aim to have as high a recall and precision as possible, but it is often difficult to achieve both at a perfect score.

Fig. 2 is plotted using ProFeaSM and inertialMultiplier = 90. After testing ProFeaSM in simulation, we collect the body angles from the executions of single motion primitives and the interpolations between pairs of motion primtives on the real NAO humanoid robot. We note that the interpolation time is about twice as long on the real robot than in simulation since we use 40 percent of the maximum joint angular velocity for the real NAO robot instead of 70 percent of the maximum joint angular velocity in simulation. Even though the interpolation times are different, we show the efficacy of ProFeaSM by varying inertialMultiplier with the recall and precision values as shown in Fig. 3.

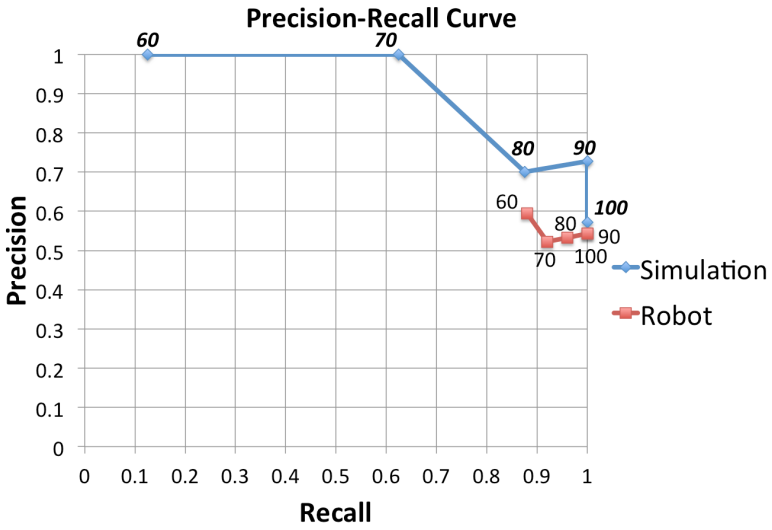


Fig. 3: Precision Recall Curve

Fig. 3 shows two curves, one for simulation and one for the real robot. The two curves are generated by varying different inertialMultiplier values, from 60 to 100, using different sets of body angle Y values – the simulated data and the actual robot data. We do not plot values of 10 to 50 as there were no sequences that were predicted as falls in simulation. Each curve is marked by the value of inertialMultiplier. The blue line for Simulation shows the precision and recall rate for using the body angle Y values collected from the execution of motion primitives in simulation and by varying the different inertialMultiplier, we get a different precision and recall score. From the Simulation results, 90 is a value to be used for the inertialMultiplier if we want to ensure that all sequences that will fall will be predicted as falls (a perfect recall value of 1.0), but we have a low precision of 0.72, which means that we have predicted some false positives (sequences that we predict as falls did not fall). If we use the same value of 90 for the actual robot prediction, we will also achieve a perfect recall value of 1.0, but the precision value is lower at 0.54. This means that we have quite a high number of false positives, which may not be desirable since we have less choices of sequences to execute. Hence, there is a trade-off between precision and recall depending on the requirements. We can require a high precision, where there are as few false positives as possible so that we can have more sequences to choose from. We can also require high recall instead of precision, where there are as many true positives as possible so that we can avoid sequences that fall and can tolerate having less sequences to choose from.

Our algorithm, ProFeaSM, scales quadratically with the number of motion primitives in the motion primitive library. We can reduce the number of times the body angle Y values are recorded for ProFeaSM if the interpolations between

pairs of motion primtives are not unique, e.g., the last keyframe of the motion primitive P_1 and the first keyframe of the next motion primitive P_2, are the same two keyframes for the last keyframe of the motion primitive P_3 and the first keyframe of the next motion primitive P_4.

We arrive at our current approach based on previous failed attempts at predicting the stability of a sequence using algorithms like Hidden Markov Model and Reinforcement Learning. There are several problems that we encounted with these algorithms. We require the executions of many iterations of different sequences to learn the transition probabilities, which does not fulfill our need as we want to predict different sequences without executing them. Moreover, both algorithms possess the Markov property–the next state depends only upon the present state and not on the sequence of events that preceded it. If the Markov property is not violated, we will require a large state space as we create new states based on the ordering of the motion primitives due to the momentum caused by the ordering of the motion primitives in the sequence. Moreover, the transition probability of a fall from each state is low, thus the probability of predicting a fall in a sequence is low too.

5 Conclusion

We contribute an algorithm, ProFeaSM, to predict if a sequence of motions will fall when the robot executes the sequence. We use only body angle Y values collected from the executions of single motions and the interpolations between pairs of single motions since the robot only falls forward or backwards and not sideways. Body angles are computed using accelerometer and gyroscope sensors and these sensors are commonly found in humanoid robots. Moreover, we do not require training instances of body angle Y values collected from sequences of motions to make predictions of the sequences, unlike traditional fall prediction methods that require training data and can only predict possible falls whilst monitoring the execution of the sequence. We make predictions before any sequence of motions is executed on the robot. We also require no model of the robot and the environment to make a prediction.

ProFeaSM includes the parameter, inertialMultiplier, that is varied to achieve different precision and recall values. Firstly, we collect body angles in simulation and test the efficacy of ProFeaSM. We showed that ProFeaSM can achieve a perfect recall value of 1 and a precision value of 0.72 when the value, inertialMultiplier is set to 90 in simulation. For the prediction of sequences that fall for the real robot, we collect body angles for the execution of all the single motion primitives in the motion primitive library of 6 motion primitives and the 24 interpolations between pairs of the motion primitives. We show that by using the same value of inertialMultiplier = 90 using our algorithm, we can achieve the same perfect recall score and predict all the sequences that fall, albeit at a poorer precision value of 0.54. By varying different values of inertialMultiplier, we can achieve different precision and recall values. We need to weigh the trade-off of having a higher recall value versus a higher precision value.

We ran ProFeaSM in simulation and on the real robot. The simulated robot is a NAO V4.0 H25 humanoid robot with 25 joint actuators, and for the real robot, we use a NAO V3.3 H21 humanoid robot with a V4.0 head that has only 21 joint actuators. Despite the differences in the number of joint actuators, the interpolation time between keyframes and the weight of the robot in simulation and of the real robot, we are still able to achieve the same recall value using our algorithm and predict all unstable sequences without executing the sequences.

Acknowledgements

This work was supported by the Singapore Millennium Foundation Research Grant and by award NSF IIS-1218932 of the National Science Foundation. The NTU-CMU Dual Degree PhD Programme in Engineering (Robotics) is funded by the Economic Development Board of Singapore. The views and conclusions contained herein are those of the authors only.

References

1. Engleberg, I., Wynn, D.: Working in Groups: Communication Principles and Strategies. Pearson Education (2006)
2. Xia, G., Tay, J., Dannenberg, R., Veloso, M.: Autonomous robot dancing driven by beats and emotions of music. In: Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS). Volume 1. (2012) 205–212
3. Tay, J., Veloso, M.: Modeling and composing gestures for human-robot interaction. In: Int. Symp. on Robots and Human Interact. Comm. (RO-MAN). (2012) 107–112
4. Dalibard, S., El Khoury, A., Lamiraux, F., Nakhaei, A., Tax, M., Laumond, J.P.: Dynamic walking and whole-body motion planning for humanoid robots: an integrated approach. Int. Journal of Robotics Research **32**(9-10) (2013) 1089–1103
5. Kuffner, J., Nishiwaki, K., Kagami, S., Inaba, M., Inoue, H.: Motion planning for humanoid robots. In: Proc. 11th Int. Symp. of Robotics Research (ISRR). (2003)
6. Kanehiro, F., Suleiman, W., Lamiraux, F., Yoshida, E., Laumond, J.P.: Integrating dynamics into motion planning for humanoid robots. In: Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on. (2008) 660–667
7. Höhn, O., Gerth, W.: Probabilistic balance monitoring for bipedal robots. Int. Journal of Robotics Research **28**(2) (2009) 245–256
8. Kalyanakrishnan, S., Goswami, A.: Learning to predict humanoid fall. International Journal of Humanoid Robotics **08**(02) (2011) 245–273
9. Höhn, O., Ganik, J., Gerth, W.: Detection and classification of posture instabilities of bipedal robots. In Tokhi, M., Virk, G., Hossain, M., eds.: Climbing and Walking Robots. Springer Berlin Heidelberg (2006) 409–416
10. Renner, R., Behnke, S.: Instability detection and fall avoidance for a humanoid using attitude sensors and reflexes. In: IROS. (2006) 2967–2973
11. Searock, J., Browning, B., Veloso, M.: Learning to prevent failure states for a dynamically balancing robot. Technical Report CMU-CS-TR-05-126, School of Computer Science, Carnegie Mellon University (2005)
12. Webots: http://www.cyberbotics.com Commercial Robot Simulation Software.
13. Aldebaran Robotics: https://community.aldebaran-robotics.com/doc/1-14/ NAO Software 1.14.5 documentation.
14. Russell, S., Norvig, P.: AI: A Modern Approach. Prentice Hall (2003)