Hardware Experiments of Humanoid Robot Safe Fall Using Aldebaran NAO

Seung-Kook Yun and Ambarish Goswami

***Abstract*— Although the fall of a humanoid robot is rare in controlled environments, it cannot be avoided in the real world where the robot may physically interact with the environment. Our earlier work [1], [2] introduced the strategy of direction-changing fall, in which the robot attempts to reduce the chance of human injury by changing its default fall direction in real-time and falling in a safer direction. The current paper reports further theoretical developments culminating in a successful hardware implementation of this fall strategy conducted on the Aldebaran NAO robot[3]. This includes new algorithms for humanoid kinematics and Jacobians involving coupled joints and a complete estimation of the body frame attitude using an additional inertial measurement unit. Simulations and experiments are smoothly handled by our platform independent humanoid control software package called *Locomote*. We report experiment scenarios where we demonstrate the effectiveness of the proposed strategies in changing humanoid fall direction.**

I. Introduction

Imagine a scenario in which a group of children approaches a full-size humanoid robot and playfully applies tugs or pushes to it. Serious injuries can result if the heavy robot accidentally loses balance and topples as a consequence. Even though the occurrence of such falls are rare, they cannot be avoided in realistic human surroundings where uncertainties are abundant, errors are frequent, and the chances of significant physical interaction, accidental or otherwise, cannot be fully eliminated. This provides a strong motivation for developing a robust fall management system, and especially for innovating effective fall controllers, that can minimize injury to people in its surroundings.

A body of research has been devoted to the study of humanoid fall control, with most work focusing either on fall prediction ([4], [5], [6]) or on minimizing impact damage to the robot ([7], [8], [9], [10]). We have earlier proposed a direction-changing fall controller which attempts to change the default fall direction of the robot in real-time such that the fall occurs in safer direction where there is no people [1], [2]. This controller, which is concerned with reducing injury *to others*, is composed of two main strategies that can be employed individually, simultaneously or sequentially. The first strategy is to cause a change in the fall direction by appropriately changing the geometry of the support area. This can be achieved by taking a step or by lifting a foot. The second strategy is to execute a coordinated whole-body motion under the scheme of inertia shaping in order to influence the motion of the robots' center of mass (CoM) through the control of its locked inertia [11] about the center of pressure (CoP).

S. Yun and A. Goswami are with Honda Research Institute US., 425 National Ave, Suite 100, Mountain View, CA, USA syun@honda-ri.com, agoswami@honda-ri.com

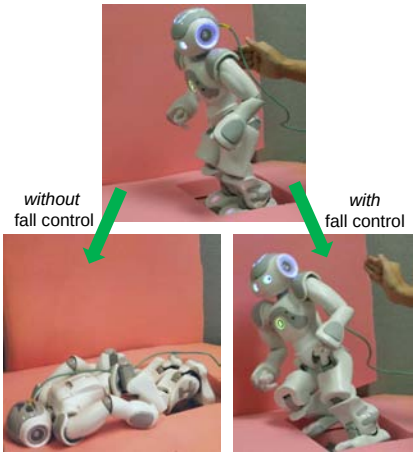


Fig. 1. Consequence of a humanoid fall without and with the proposed fall controller. The NAO robot is initially in upright stance pose and is subjected to a forward push (top figure), Without any fall controller the robot falls directly forward. The fall controller successfully changes the fall direction and the robot is able to avoid falling forward.

Fig. 1 shows two cases of humanoid fall caused by a push from behind when the robot is standing upright (top figure). We assume there is an important object in front of the robot which the robot must avoid hitting. Without any fall control, the robot will fall forward and presumably hit the object (bottom, left). In the second case (bottom, right), the robot recognizes the position of the object and the proposed controller successfully avoids hitting it.

In this paper, we demonstrate that these foot-lifting, stepping and inertia shaping are indeed effective for changing the fall direction in experiments. Though the ultimate target of our research is a full-size humanoid robot, which can actually cause damage to the environment or physical injury to people, we use the smaller Aldebaran NAO robot [3] as a first step such that the damage to the robot by repetitive experiments is minimized.

The implementation of the safe-fall strategy to a hardware platform is far from straightforward. Significant hurdles need to be overcome, most being specific to the NAO hardware. Two main challenges are related to the sensing of global location and attitude of the robot as well as determining its contact points. The robot has limited sensing during toppling when either of its two feet is not firmly planted on the ground. We were required to use an additional inertial measurement unit (IMU) sensor and develop estimation

algorithms to obtain the global location. We also need to update our existing kinematic and inertia shaping (whole-body motion) algorithms to accommodate a shared joint along the two legs, which is a feature of the NAO robot.

For both simulation and experimental control we have used our unique software *Locomote* which is designed to be not tied to any specific humanoid model, either in simulation or experiment. Section III details the main features of Locomote.

II. Fall Direction Change Through Support Area Modification

In this Section, we review the core concepts of our previous work on direction changing fall [1], [2].

A. Background of Support Area Modification

When a robot starts to topple, its CoP rapidly moves towards an edge (or corner) of the support area. As a consequence, the robot rotates about this *leading* edge (or corner). Therefore, *a change in the physical location of the leading edge (corner) of the support area with respect to the robot CoM, exerts influence on the direction of robot rotation i.e. the direction of fall.*

In Fig. 2, a humanoid robot is subjected to a forward push as indicated by the arrow at the top left. If the push is strong enough to topple the robot, the CoP will approach the front edge of the support base and the robot will begin to rotate about this leading edge.

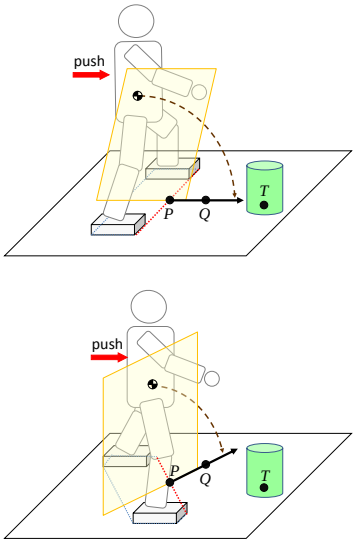


Fig. 2. A schematic diagram showing the basic idea behind fall direction change through support base geometry modification. A forward push is assumed. P denotes the CoP, Q is the reference point to which the robot is falling, and T is a target object that the robot should avoid damaging. The dotted lines show the support area (polygonal convex hull) of the robot while the polygon edge containing CoP is dotted.

The direction and magnitude of the toppling motion is given by PQ where P is the CoP and Q is what we call a *reference point*. The reference point indicates the direction and magnitude of fall. In this paper we use the *capture point*[12] as our reference point. Although PQ may not be initially perpendicular to the leading edge of the support base, it becomes so once the toppling motion sets in.

B. Support Area Modification Controller

Once the fall of a humanoid is determined to be certain, we estimate the time to fall and the default fall direction. Based on this estimation, the controller finds the best position to step and controls the stepping leg accordingly (See [1]).

Assuming that the humanoid is in double support phase, the controller chooses one of the following actions, if necessary (no action may lead to a safe fall direction), to find the optimal support area:

1) Lifting (and not re-planting) left or right foot (2 cases)
2) Taking left or right step (2 cases).

C. Direction-changing Fall through Inertia Shaping

Arm windmilling, bending of the upper body at the hip, and vertically lowering the upper body are natural human responses under external disturbances. Likewise, a humanoid can further change the direction of fall. Since a falling robot is normally underactuated, a direct control of the CoM would not be effective. However, we can indirectly change the fall direction by generating angular momentum away from the direction of the target. Inertia shaping [13] is used to control this whole-body motion by reconfiguring the desired inertia matrix of the robot (See [1]).

III. Locomote: Hardware-Independent Software Architecture

In order to reduce the dependence of our controller on a specific humanoid model, we designed our software package to posses a modular architecture in which the modules for dynamic simulations and hardware can be replaced without requiring to re-code other parts. This is achieved by placing an interface class that separates the controller on one side and the dynamics simulator and the hardware on the other. The interface class defines all the inputs and outputs of the controller on an abstract level, and acts as an interpreter between the controller and the dynamic simulator. It becomes necessary and sufficient for the controller to communicate with this class without the need to directly accessing specific functions in the simulator or the hardware. We have implemented our proposed architecture in our simulator package called Locomote.

A. The Locomote Package

Fig. 3 shows the overall structure of the *Locomote* package.

Locomote is composed of the following modules: (1) *Robot Controller* which includes *Controller* to generate the control command given the sensor data and *Robot Solver* to compute all the kinematic and dynamic data required by *Controller*, such as inverse kinematics/dynamics, Jacobians, CoM and CoP. *Robot Configuration* is shared through *Robot Controller*. (2) *Robot Interface* to connect *Robot Controller* and *Applications* so that they can be independently developed. (3) *Applications* to send out the sensor data and to

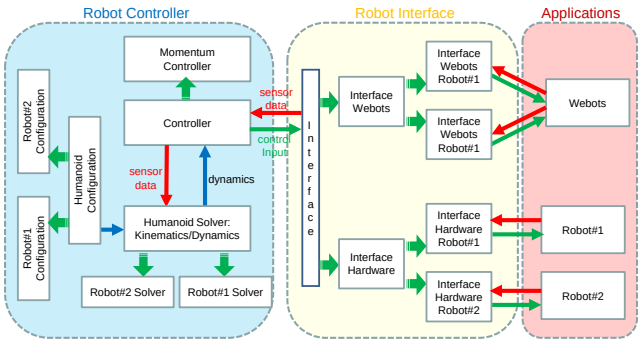
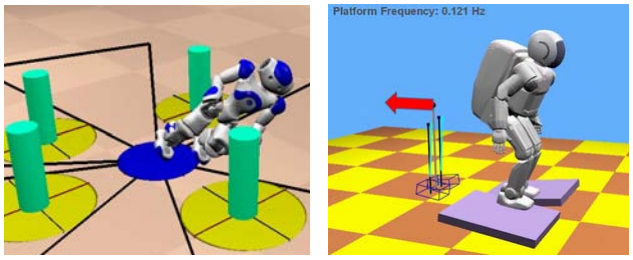


Fig. 3. Architecture of Locomote for simulations and experiments of two humanoid platforms. Sensor data and control inputs are processed through an interface between the robot controller and simulator/hardware. Interface classes for the dynamic simulator and the robot hardware are inherited from the abstract interface class which is accordingly declared in the controller class. Webots[14] is used for the simulations of both platforms. Each base class is derived into corresponding sub-classes with respect to the platform. In addition, the momentum-based controller class is derived from the position controller class.



(a) (b)

Fig. 4. Applications of the Locomote package. (a) Simulation of fall control by the position controller. The humanoid NAO would fall forward on an obstacle without any control, however it ends up with falling towards a safer region and avoids hitting the important objects (green cylinders). (b) Simulation of balance control using a momentum controller. The full-sized humanoid is balancing itself on the moving plates (See the red arrow for velocity of the plates) which are tilted differently. The experiments of this paper are also implemented by Locomote.

receive the control command from *Controller*. This can be dynamic simulator or robot hardware.

The core idea of Locomote is that *Robot Controller* is separated from the application environment by Interface, a class that defines specific types of interactions between *Robot Controller* and *Applications*. At the top level, the interface is defined as an abstract class with pure virtual member functions, which are called by the controllers. The interface defines all the sensor inputs and control outputs of a controller such that a controller needs only to interact with the interface class. At this level, the actual realization of the sensor input and control output is not defined, which is fulfilled by derived classes. The application environment classes are associated with corresponding derived classes that inherit from the abstract interface class and implement the specific member functions in terms of the functions of the particular applications.

This way a controller is prevented from directly calling functions specific to a dynamic simulator or a humanoid hardware, and hence can be independent of particular dynamic simulator or hardware devices.

The inputs and outputs of the interface class must correspond to the capabilities of the target hardware robot. The types of sensory data should be limited to those which can be acquired by the humanoid robot.

B. Applications of Locomote

We have used Locomote to implement dynamic simulations in the Webots environment including (1) Fall controller [2], [1] in which the humanoid adjusts its default fall direction to avoid hitting any important objects around the robot (See Figure 4(a)), (2) Momentum based balance controller [15] in which the humanoid optimize its torque output to balance itself against external disturbance such as a push or a change of ground geometry. (See Figure 4(b)).

We are also using Locomote for hardware experiments of the fall and balance controller in which the environment class communicates with the humanoid hardware via a TCP/IP network (See Figure 1).

IV. Adaptation to the Aldebaran NAO Robot

In order to implement the direction-changing control to the NAO robot, we need to address a few issues specific to this humanoid robot. The NAO robot is 58 cm tall, possesses 22-dof and weighs about 5 kg. The kinematic structure of this robot is unique in that the hip joint is shared by both legs. Consequently, our inverse kinematics and inertia shaping algorithms must be updated. Another main challenge is that due to limitation in the built-in gyro of the robot, a complete attitude of the robot trunk is not available when either foot is not firmly planted on the ground. This makes it impossible to fully obtain the global position of the robot during fall. In this section, we address these two issues.

A. Inverse Kinematics and Inertia Shaping Involving a Shared Joint

Unlike most humanoid robots, the two legs of the NAO robot have different dofs due to the shared joint which connects the body to both legs. One way to treat this shared joint is to imagine an asymmetry in the legs where one leg has 6-dof and fully possesses the pelvic joint while the other leg has 5-dof under the hip. This asymmetry raises a problem in solving the inverse kinematics since most humanoids have two 6-dof legs and the typical inverse kinematics solution for a 6-dof link can be used for both legs. In order to use our fall controller for NAO, we design a Jacobian-based inverse kinematics for this special joint configuration, which enables stepping as well as control of the body posture.

Suppose that both legs of the robot have firm support on the ground. We have the following equations:

$$\dot{\boldsymbol{P}}_L - \dot{\boldsymbol{P}}_{body} = \boldsymbol{J}_L \dot{\boldsymbol{\theta}}_L \quad (1)$$

$$\dot{\boldsymbol{P}}_R - \dot{\boldsymbol{P}}_{body} = \boldsymbol{J}_R \dot{\boldsymbol{\theta}}_R, \quad (2)$$

where $\boldsymbol{P}_L$ and $\boldsymbol{P}_R$ are the positions of the left and right feet, respectively, $\boldsymbol{P}_{body}$ is the location of the body frame, $\boldsymbol{\theta}_R$ is 6×1 joint angle vector of the right leg, $\boldsymbol{\theta}_L$ is 5×1

joint angle vector of the left leg, and $\boldsymbol{J}_L$ and $\boldsymbol{J}_R$ are the leg Jacobian matrices. Note that any one of the legs could be considered 6-dof while the other is 5-dof.

Subtracting Eq.1 from Eq.2, we get:

$$\dot{\boldsymbol{P}}_R - \dot{\boldsymbol{P}}_L = [\boldsymbol{J}_R \quad -\boldsymbol{J}_L] \left[\dot{\boldsymbol{\theta}}_R \; \dot{\boldsymbol{\theta}}_L\right]^T \tag{3}$$

where we define the (6×11) foot-to-foot Jacobian matrix as follows:

$$\boldsymbol{J}_{R-L} = [\boldsymbol{J}_R \quad -\boldsymbol{J}_L] \tag{4}$$

Since we want to control the location of the body frame as well as to take a step[1], we can design a cost function for the inverse kinematics algorithm to minimize:

$$\begin{aligned} min \quad & \|\Delta \boldsymbol{P}_{R-L} - \boldsymbol{J}_{R-L}\Delta\boldsymbol{\theta}\|^2 + \lambda^2 \|\Delta\boldsymbol{\theta}\|^2 + \\ & \epsilon^2 \|\Delta \boldsymbol{P}_{body} - \boldsymbol{J}_R \Delta\boldsymbol{\theta}_R\|^2 , \end{aligned} \tag{5}$$

where $\boldsymbol{P}_{R-L}$ is the displacement between the two feet, $\boldsymbol{P}_{body}$ is the location of the body frame, and λ and ϵ are constants. This cost function pursues the simultaneous control of the body location and the step displacement while minimizing the total joint angle displacements.

Eq. 5 can be re-written as follows:

$$min \quad \left\| \begin{bmatrix} \boldsymbol{J}_{R-L} \\ \lambda \boldsymbol{I} \\ \epsilon \begin{bmatrix} \boldsymbol{J}_R & 0 \end{bmatrix} \end{bmatrix} \Delta\boldsymbol{\theta} - \begin{bmatrix} \Delta \boldsymbol{P}_{R-L} \\ 0 \\ \epsilon \Delta \boldsymbol{P}_{body} \end{bmatrix} \right\|^2 , \tag{6}$$

which leads to the following inverse kinematics solution:

$$\Delta\boldsymbol{\theta} = \left(\hat{\boldsymbol{J}}^T \hat{\boldsymbol{J}} + \lambda^2 \boldsymbol{I}\right)^{-1} \hat{\boldsymbol{J}}^T \begin{bmatrix} \Delta \boldsymbol{P}_{R-L} \\ \Delta \boldsymbol{P}_{body} \end{bmatrix} , \tag{7}$$

where

$$\hat{\boldsymbol{J}} = \begin{bmatrix} \boldsymbol{J}_{R-L} \\ \epsilon \begin{bmatrix} \boldsymbol{J}_R & 0 \end{bmatrix} \end{bmatrix} . \tag{8}$$

The parameter ϵ controls the relative importance between the step displacement and the body displacement. For example, a low ϵ puts higher priority on the step displacement.

Note that this inverse kinematics is another version of the damped least-squares solution [16].

Also in inertia shaping, we adjust the centroidal composite rigid body (CRB) inertia Jacobian $\boldsymbol{J_I}$ accordingly which maps changes in the robot joint angles $\boldsymbol{\theta}$ into corresponding changes in the strung-out CRB inertia matrix $\widehat{\boldsymbol{I}}$:

$$\delta\widehat{\boldsymbol{I}} = \boldsymbol{J_I}\delta\boldsymbol{\theta}. \tag{9}$$

Therefore $\boldsymbol{J_I}$ is a 6×11 matrix rather than a 6×12 matrix used in [1].

B. Estimation of Global Position and Foot/Ground Contact Point

The biggest challenge for the experiment is to estimate the global posture of the body frame while the robot is falling. In simulation, this information was readily available. However, during the experiment we have to estimate it using an inertial measurement unit (IMU: accelerometers and gyros) and force sensitive resistors (FSR) in the feet. The estimation is relatively easy when one foot has a firm contact with the ground since the relative 3D transform between the foot and the body frame results in the global position. However, when the robot is falling, the robot can lose the firm contact, and we need to depend on the IMU.

[1]This is straightforward when each leg has 6-dof

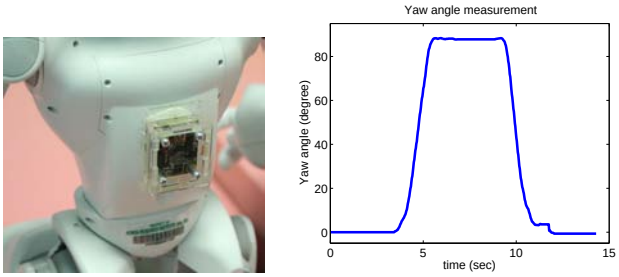


Fig. 5. (a) An additional IMU is attached on the back of the NAO robot. (b) Measured yaw angle when the robot is manually rotated by about 90 degrees and returned to 0 degree.

Unfortunately, the built-in IMU in the NAO robot has only 2 gyros added to 3 accelerometers which can yield only the roll and the pitch angles. This may suffice when the robot has a firm contact on the ground, but not when foot toppling is involved. To estimate the missing yaw angle, we attached an additional IMU with 3 gyros and 3 magnetometers. Fig. 5 shows the attached IMU and its reading of the yaw angle.

Given these estimated orientations, the position of the body is still missing. In order to estimate it, we use the relative pose between the contact point and the body given the assumptions: no slip, no change of the contact point and a rectangular foot area. If the robot does not experience any slip and maintain the contact point during fall, the contact point can be referenced for global position. Fig. 6 shows an example of the falling robot. The point is that we can obtain the estimated orientation of the body frame and the estimated location of the foot contact point. Combining them results in the full posture of the body frame.

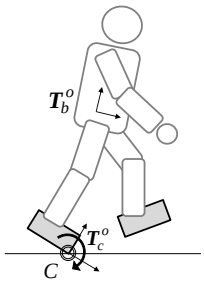


Fig. 6. Coordinates of the body and the anchor foot. C is a contact point. T_f is a frame of the anchor foot, and T_b is the body frame. Note that the transformation from the foot to C with respect to T_f is constant.

The following equation describes the relative posture between the contact point and the body frame:

$$\begin{bmatrix} \boldsymbol{R}_c^0 & \boldsymbol{P}_c^0 \\ 0 & 1 \end{bmatrix} \boldsymbol{T}_b^c = \begin{bmatrix} \boldsymbol{R}_b^0 & \boldsymbol{P}_b^0 \\ 0 & 1 \end{bmatrix} , \tag{10}$$

where $\boldsymbol{T}$ is a transformation matrix and $\boldsymbol{R}$ and $\boldsymbol{P}$ are a rotation matrix and a position vector. 0, c and b in the super and sub-scripts refer to the global frame, the contact point frame and the body frame, respectively. In Eq. 10, the orientations of the contact point $\boldsymbol{R}_c^0$ and the position of the body frame $\boldsymbol{P}_b^0$ are unknown given the joint angles.

Fig. 7. Default case: without direction-changing fall controller, the NAO robot, when pushed from behind by a linear actuator, falls forward.

Rewriting Eq. 10 leads to the solution:

$$\boldsymbol{R}_c^0 = \boldsymbol{R}_b^0 \boldsymbol{R}_c^b \tag{11}$$

$$\boldsymbol{P}_b^0 = \boldsymbol{R}_c^0 \boldsymbol{P}_b^c + \boldsymbol{P}_c^0, \tag{12}$$

since

$$\boldsymbol{T}_b^c = \begin{bmatrix} \boldsymbol{R}_b^c & \boldsymbol{P}_b^c \\ 0 & 1 \end{bmatrix}. \tag{13}$$

In order to estimate the foog/ground contact point during fall, again we assume no slip and non-changing contact point during fall. Unlike during simulations where the contact point information can be directly obtained, we have to estimate it during the experiment. Since we also assume the foot area is a perfect rectangle, the contact point can be either an edge (the robot topples like a 2D inverted pendulum) or a vertex (the robot falls as a 3D inverted pendulum). We determine this from the 4 FSRs of each foot. At every control sampling time the controller checks the values of the FSRs and check states from a tuned threshold. From empirical data, the controller estimates that the contact is over an edge when two adjacent FSRs are *ON* and their values are equivalent while the other two are *OFF*. If one of them has a significantly higher value than the other, the controller interprets this a vertex contact.

V. Fall Direction Change Experiment

We implement our fall controller in hardware experiment and compare the results to the simulation.

A. Challenges for Experiments

In order to implement the fall controller on the NAO robot, we have to consider limited capabilities of sensing and control with respect to what can be measured, implemented, and computed. The main differences between the simulation and the experiment are listed in Table I. The strategies described in Section II-B and II-C are utilized.

B. Experimental Results

In the first experiment, the robot gets a steady push from behind until it switches to fall control mode and the proposed strategies are utilized. For repeatability, we use a linear actuator to give a push to the robot (the machine visible behind the NAO robot in Fig. 7). The Locomote controller runs on an external laptop connected to the robot via wired network. The lean angle of the robot estimated from the IMU is used to trigger the direction-changing fall controller.

Without a fall controller, the robot falls forward as shown in Fig. 7. Figs. 8(a-b) demonstrate that the foot lifting strategy can make a significant change under the same push. The robot lifts the right leg to change the fall direction and falls almost to the right. We tested two foot lifting strategies which lifts the left/right leg respectively, and the resultant CoM trajectories are compared in Fig. 9(a).

According to Fig. 9(b), our fall controller seems to over-predict the resultant fall angles. We think that this difference is caused mainly due to the change of the foot/ground contact point during fall. The prediction comes from considering the robot as a 3D pendulum with a *fixed* contact point, which becomes invalid when the foot/ground contact point moves. For example, in Fig. 8(a), the foot/ground contact point is at the front-right corner of the left foot which our controller correctly estimates and the predicted fall angle is 114 degrees. However somewhere between Fig. 8(a) and Fig. 8(b), the 3D fall motion of the robot causes the right edge of the left foot to become the foot/ground contact edge. This prevents the robot from rotating further backward, and the robot ends up falling to the right (around 90 degrees). In future work, this issue should be addressed in order to obtain better prediction accuracy.

In the next experiment, in order to test the effectiveness of inertia shaping, we performed an experiment to see if through inertia shaping we can cancel the effect of fall direction change, which was originally achieved through foot lifting. As seen in Figs. 8(a-b), lifting of the right foot causes the robot to fall toward its right. In the following example, after foot lifting, we execute inertia shaping using the forward direction fall as the objective. As seen in Figs. 8(c-d), inertia shaping is shown to have successfully canceled the effect of foot lifting and the robot falls forward. Note that the arms are stretched to maximize the effect of inertia shaping. In independent inertia shaping experiment, we can make the robot fall diagonally, under a forward push, as shown in Figs. 8(e-f).

Fig. 10 shows how inertia shaping changes the CoM trajectory.

The third experiment checks the effect of pure inertia shaping without involving any stepping. In this experiment, only inertia shaping is used to change the fall direction. In the experiment described in Figs. 8(c-f), the robot has very short time for inertia shaping because it spends a part of the fall time in lifting up a leg. In order to have more control time dedicated to inertia shaping, in this experiment we start from a single support pose of the robot as shown in Fig. 11(a). The robot is pushed from the left and falls to the right without inertia shaping. Two independent experiment of inertia shaping with 0 degree (forward) and 45 degrees (forward right) desired fall angles are implemented. The success of this experiment is evident in the resultant CoM trajectories (See Fig. 11(b)).

In the fourth experiment, Fig. 12 shows snapshots of the experiment for stepping strategy. A push comes from the left of the robot which is supported by the left foot only. The controller modifies the support area to change the fall direction to 45 degrees (right forward). The support area changes from a rectangle to a line and to a pentagon. The

Simulation	Experiment
• Faster control sampling time (1 kHz)	• Slow control sampling time (≈30 Hz)
• Perfect knowledge of exact global position of the body frame	• Noisy estimation of global position of the body frame
• Perfect sensing of joing angle, velocity and acceleration	• Only joint angles sensed
• Perfect knowledge of foot/ground contact points	• Imperfect estimation of foot/ground contact point
• Perfectly rectangular feet	• Feet perimeter is curved
• Perfect knowledge of exact timing of push	• Timing of push is unknown

TABLE I

DIRECTION-CHANGING FALL: DIFFERENCE BETWEEN SIMULATION [1] AND EXPERIMENT

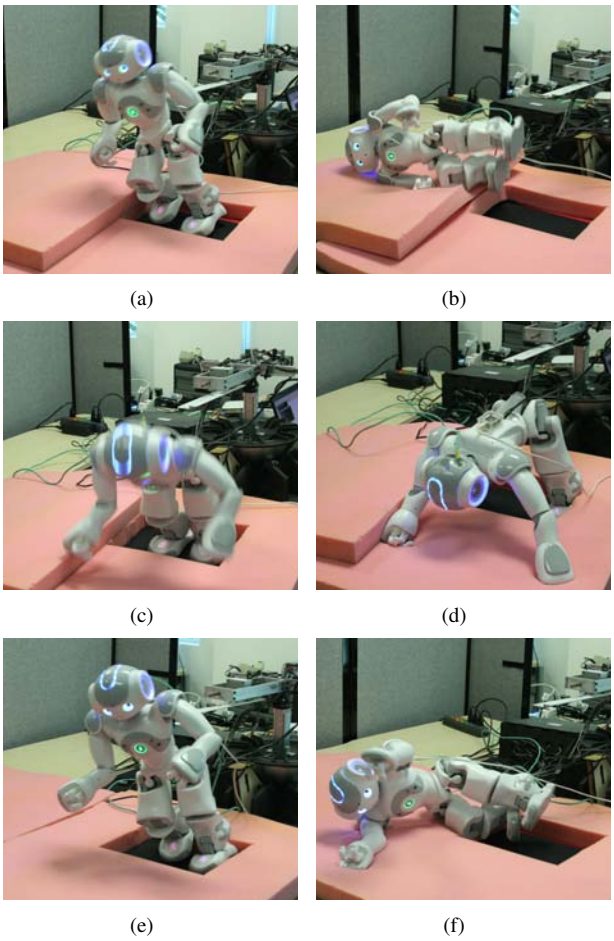


Fig. 8. Snapshots of the fall experiment with foot lifting strategy and inertia shaping. (a) The robot lifts up the right leg. (b) The robot falls almost completely to the right. (c) After lifting up the right leg, the robot starts inertia shaping with the objective of canceling the effect of the foot lifting strategy. (d) Inertia shaping successfully makes the robot fall almost forward. (e) The robot uses inertia shaping to fall diagonally forward after lifting up the right foot, and inertia shaping is reasonably successful (f).

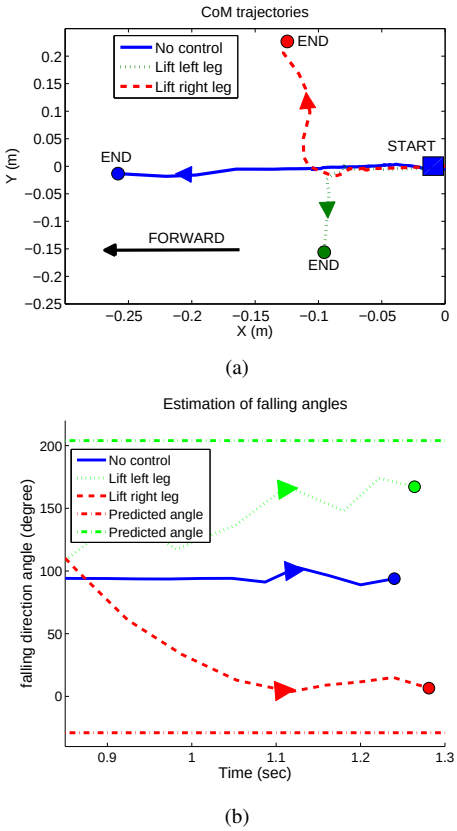
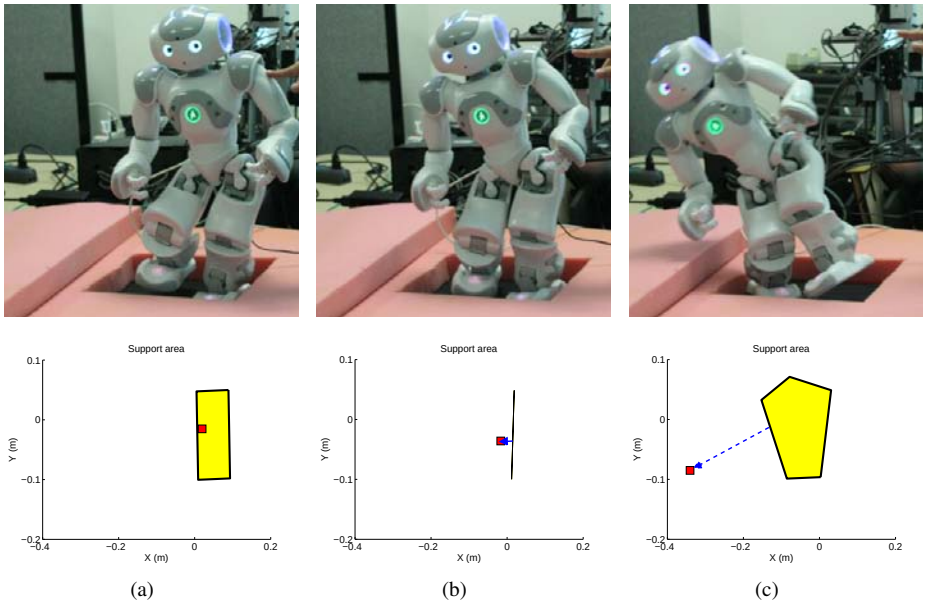


Fig. 9. (a) CoM trajectories of the robot during the foot lifting strategies. The circles denote the end of the trajectories. The solid blue curve is the CoM trajectory of the falling robot without a fall controller, and the dotted green and dashed red curves are trajectories from our fall controller by lifting the left and right leg, respectively. The forward direction is displayed by the black arrow. (b) Measured fall angles with respect to the lift-up strategies and their estimations shown as horizontal lines at the top and bottom. The differences between estimations and experimental results mainly from the fact that the foot/ground contact point changes over time during fall.

direction of fall changes, as expected, according to support area, and the resultant trajectory of the CoM is shown in Fig. 13 in which the robot also takes a step to change the fall direction to -45 degrees (right backward). When the humanoid is on single support in Fig. 12(a), it topples to the right and rotates about the right edge of the support foot as shown in Fig. 12(b). Once the robot takes a step with the right foot rotated by 45 degrees, the support base extends to a pentagon as shown in Fig. 12(c). The direction of fall goes to the right forward since the reference point (Capture point) is at the right forward of the support polygon.

C. Discussion: Comparison with Simulation

Fig. 14 shows motions from simulations corresponding to the experimental results in Fig. 8(c-f). We see that the apparent motions in the experiments match well those in the simulations. However we found lack of the motor power in the experiment. Even though we use the same maximum joint speed and torque as in the simulations, motors with high load often could not follow the desired trajectories and stalled. Due to the property of fall, the robot is likely



(a) (b) (c)

Fig. 12. Upper pictures are snapshots of falling humanoid with changing support polygon. The lower figures show the support polygon and Capture point. The small red square is Capture point. The dashed blue arrow is the estimated fall direction. (a) The support area is a rectangle formed by the left foot. Capture point resides inside the support area. (b) The robot is toppling after the push, and the support area in the inner edge of the left foot. Capture point is at the right, which implies the robot is falling to the right. (c) The robot has taken a step, and the support area is a pentagon formed by the contact points of the two feet. Capture point is out of the support area, and the robot falls diagonally as we intended. The target falling angle of the controller is 45 degree (right forward). The CoM trajectory of this experiment is shown in Fig. 13.

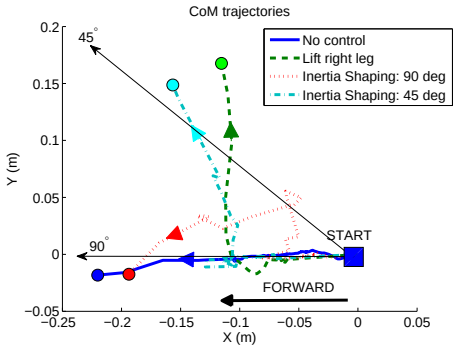
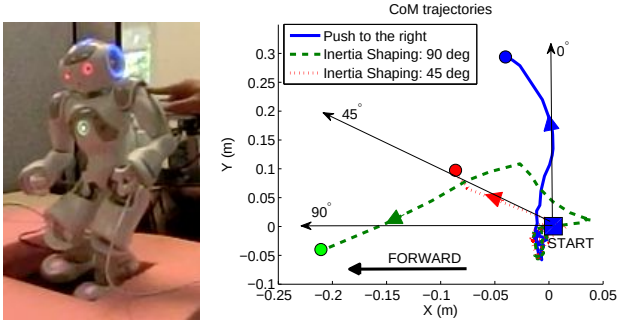


Fig. 10. CoM trajectories. The solid blue curve is the trajectory of the falling robot without any control. The dashed green curve corresponds to the foot-lifting strategy. The dotted red curve is for the controller with inertia shaping to forward. The dot-dashed cyan curve is done by inertia shaping to right forward. The circles denote the end of the trajectories.



(a) (b)

Fig. 11. Effect of inertia shaping during fall. (a) The NAO robot is in single support on the left leg when it is pushed from the right. (b) CoM trajectories with/without inertia shaping. Without inertia shaping, the robot falls to the right (solid blue curve). Two tests of inertia shaping with target angles of 45 (dotted red curve) and 90 (dashed green curve) degrees are used to change the fall direction. The circles denote the end of the trajectories.

unbalanced and some joints would be under very high load mainly due to gravity. Therefore, during fall control, the desired motion cannot be met due to this load since the joints cannot be actuated properly. In this example of Fig. 8(e-f), we found out that the hip roll joint did not follow the desired trajectory, which caused a distorted motion and the fall direction diverged from what was obtained from simulation. This lack of power makes it hard to achieve consistency of the experiments. Given the same initial condition including a push, the robot may take the right action expected in simulation when every joint follows the desired trajectories but may not when any controlled joint is stalled. Thus currently the capability of our fall controller is limited by the hardware specifications.

Also, the maximum rotational speed of the actuators did not match those assumed in the simulation. Lifting-up a leg by the same amount of height takes more time for the experiment, which means we have smaller time for the following motion like inertia shaping. However we can incorporate actual torque and velocity limits in Locomote if we know them beforehand.

Since the current NAO API does not support velocity control used in simulation, we had to modify the velocity controller into a position controller. This controller modification and the slow control sampling time in the experiment (≈30 Hz) sometimes caused jerky motion.

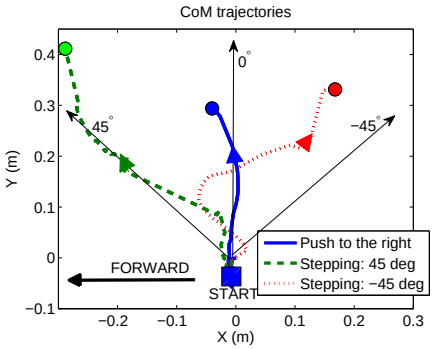
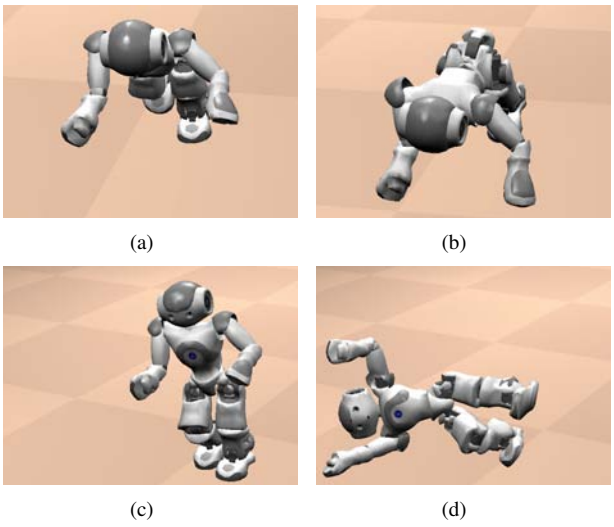


Fig. 13. CoM trajectories when the robot uses the stepping strategy. Two target falling angles (± 45 degrees) are used. The solid blue line is the COM trajectory of the falling robot after a push from the left. The dashed green curve is from the stepping strategy with the 45 degree target angle, and the dotted red curve is for -45 degree target angle. The solid circles denote the end of the trajectories.



(a) (b) (c) (d)

Fig. 14. Snapshots of the fall simulations which have the same goal fall direction as the experiments in Fig. 8(c-f). (a-d) match Fig. 8(c-f) respectively.

VI. Conclusion and Future Work

We implemented direction-changing fall controller on an Aldebaran NAO robot hardware through our own software architecture called Locomote. The core of the architecture is an interface which connects various humanoid robot platforms to the controller using virtual functions which do not explicitly include platform-specific information. This architecture helped the smooth implementation for multiple humanoid platforms.

The low-level sensing and control was also very challenging for the fall experiment due to lack of sensing capabilities for emergency as fall where the robot loses a firm contact with the ground. The global position of the body frame was estimated using filtered orientation from an additional IMU, and foot/ground contact point was estimated from the foot contact sensors.

Fall experiments were performed where the robot received a steady push from behind. Three proposed techniques, lifting-up a leg, stepping and inertia shaping were used and we compared the experimental results to results from simulation.

We believe much room is still left for further development. On-line adjustment should be added to the controller to deal with an unpredicted situation or an error of the estimation, and the controller needs to incorporate a non-rectangular feet area.

References

[1] S. Yun and A. Goswami, "Safe fall: Humanoid robot fall direction change through support base geometry modification," in *IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009, pp. 781 – 787.

[2] U. Nagarajan and A. Goswami, "Generalized direction changing fall control of humanoid robots among multiple objects," in *IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, May 2010, pp. 3316–3322.

[3] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "Mechatronic design of NAO humanoid," in *IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009, pp. 2124–2129.

[4] S. Kalyanakrishnan and A. Goswami, "Learning to predict humanoid fall," *The International Journal of Humanoid Robotics*, vol. 8, no. 2, pp. 245–273, 2011.

[5] R. Renner and S. Behnke, "Instability detection and fall avoidance for a humanoid using attitude sensors and reflexes," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, October 2006, pp. 2967–2973.

[6] J. G. D. Karssen and M. Wisse, "Fall detection in walking robots by multi-way principal component analysis," *Robotica*, vol. 27, no. 2, pp. 249–257, 2009.

[7] K. Fujiwara, F. Kanehiro, S. Kajita, K. Kaneko, K. Yokoi, and H. Hirukawa, "UKEMI: Falling motion control to minimize damage to biped humanoid robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lansanne, Switzerland, Sept. 2002, pp. 2521 – 2526.

[8] K. Fujiwara, S. Kajita, K. Harada, K. Kaneko, M. Morisawa, F. Kanehiro, S. Nakaoka, S. Harada, and H. Hirukawa, "Towards an optimal falling motion for a humanoid robot," in *6th IEEE-RAS International Conference on Humanoid Robot*, Genova, Italy, Dec 2006, pp. 524–529.

[9] K. Ogata, K. Terada, and Y. Kuniyoshi, "Real-time selection and generation of fall damagae reduction actions for humanoid robots," in *International Conference on Humanoid Robots*, Daejeon, Korea, Dec 2008, pp. 233–238.

[10] J. R. 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*, vol. 57, no. 8, pp. 796–807, 2009.

[11] M. W. Walker and D. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 104, pp. 205–211, Sept. 1982.

[12] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, "Capture point: A step toward humanoid push recovery," in *International Conference on Humanoid Robots*, Genova, Italy, Dec 2006, pp. 200 – 207.

[13] S.-H. Lee and A. Goswami, "Reaction mass pendulum (RMP): An explicit model for centroidal angular momentum of humanoid robots," in *IEEE International Conference on Robotics and Automation*, Roma, Italy, April 2007, pp. 4667–4672.

[14] O. Michel, "Webots: Professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.

[15] S.-H. Lee and A. Goswami, "Ground reaction force control at each foot: A momentum-based humanoid balance controller for non-level and non-stationary ground," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, Oct 2010, pp. 3157 – 3162.

[16] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," Department of Mathematics, University of California, San Diego, Tech. Rep., 2004.