Realization of a Real-time Optimal Control Strategy to Stabilize a Falling Humanoid Robot with Hand Contact

Shihao Wang[1] and Kris Hauser[2]

***Abstract*— In this paper, we present a real-time falling robot stabilization system for a humanoid robot in which the robot can prevent falling using hand contact with walls and other surfaces in the environment. Instead of ignoring or avoiding interaction with environmental obstacles, our system uses obstacle geometry to determine a contact point that reduces impact and necessary friction. It uses a planar dynamic model that is appropriate for falling stabilization in the robot's sagittal plane and frontal plane. The hand contact is determined with an optimal control approach, and to make the algorithm run in real-time, a simplified three-link robot model and a pre-computed database of subproblems for the hand contact optimization are adopted. Moreover, if the robot is not leaning too far after stabilization, we employ a heuristic push-up strategy to recover the robot to a standing posture. System integration is performed on the Darwin-Mini robot and validation is conducted in several environments and falling scenarios.**

I. INTRODUCTION

Humanoid robots can exploit legged locomotion to step over obstacles, walk on uneven terrains, and reduce energy consumption by employing passive dynamics [1]. However, they have a high risk of falling due to the natural instability of bipedal locomotion [2]. External factors such as unevenness in the ground, malfunctions of the balancing controller, unexpected pushes, and nontrivial disturbances can cause a robot to fall as well. Because of severe damage may be caused to the robot from falling, many strategies have been proposed to recover the fall at the instant of falling [3], [4], or to minimize the damage when an inevitable fall is triggered [5], [6]. Protective stepping is the most widely used strategy. This method plans future capture footsteps for the robot to stop or slow its rate of fall. However, protective footsteps may not be feasibly taken if the robot were to be used in a cluttered environment with obstacles such as walls and tables. These environmental obstacles can occupy positions of the planned footsteps and preclude the robot from being recovered from fall. However, the existence of environment obstacles provides the robot with a novel strategy for fall recovery. Inspired by human's natural reaction to make hand contact with the nearby wall to arrest the fall after being pushed, our previous work [7] proposes that humanoid robots can adopt hand contact for fall recovery as well. However, this strategy was only evaluated in simulation, and physical robots require handling real-time constraints and integration with other components of fall recovery, such as fall detection.

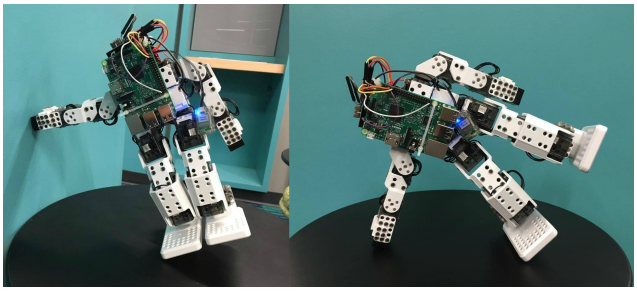


Fig. 1: Illustrating how the hand of a humanoid (ROBOTIS Darwin Mini [8]) can be used to stabilize itself on a wall and to reduce falling damage on flat ground.

This paper presents the implementation of a real-time fall recovery system that uses hand contact with the environment to prevent a humanoid robot from falling (Fig. 1). This system is capable of 1) fall detection and fall direction prediction, 2) using hand contact to stabilize the robot, 3) if physically possible, utilizing a push-up motion to recover to the standing posture. Fall detection is based on the computation of the orbital energy [9] in the falling plane. The decision about where and how to place hand contact is determined via trajectory optimization, which is a modified version of prior work [7] that achieves better real-time performance. Finally, if the robot is not tilted too far, a heuristic push-up recovery mechanism is implemented for the robot to bring itself back to a standing posture.

We implement the system on a small humanoid platform augmented with additional sensors. It consists of four components:

- Base robot — ROBOTIS Darwin Mini
- Microcomputer — Raspberry Pi 3
- IMU — Adafruit BNO055
- Touch Sensor — ROBOTIS TS-10

The implementation of falling stabilization must cope with a number of issues like limited computational power and time delays from communication and optimization. Experiments demonstrate that the stabilization system helps prevent the robot from falling in real-time in environments with and without walls, and with waist-height obstacles.

II. RELATED WORK

The problem of humanoid robot falling has been considered by several authors. Overall, related work can be classified into three main categories: fall detection, push recovery, and fall mitigation.

*This work was supported by NSF grant NRI #1527826

[1]Shihao Wang is with the Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708, USA shihao.wang@duke.edu

[2]Kris Hauser is with the Departments of Electrical and Computer Engineering and Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708, USA kris.hauser@duke.edu

Fall detection plays an important role in stabilization and it is a prerequisite for the proposed system. Kalyanakrishnan and Goswami propose a machine learning based method to conduct the falling classification while satisfying the tradeoff between fast trigger time and high accuracy [10]. Renner and Behnke uses a model-based abnormality detection method with a specific gait [11]. Karssen and Wisse propose a fall detection method based on the magnitude of the deviation of the current robot state from the commanded robot state [12]. Ogata and his colleague propose an abnormal detection method and predicted zero moment point (ZMP) to evaluate the risk of falling [13]. Our fall detection technique embraces the idea of capture point and uses the orbital energy as the falling index [3], [14].

Push recovery strategies aim to balance the robot with the internal joints control such as ankle strategy or hip strategy [15], or the use of swing foot for stepping to reduce the extra momentum, and can be employed in a unified framework with walking [3]. Besides falling in sagittal plane, push recovery strategies have also been studied for omni-directional falling [16]. However, these techniques do not consider environmental constraints, and inside buildings or other crowded environments could run the robot into obstacles. A recent trends in hand contact strategy address the environment obstacles in fall stabilization process [17], [18]. However, these two works oversimplify the contact mode and neglect the friction constraint and the change of the robot state due to the contact impulse.

Fall mitigation strategies reduce the impulse or damage at collision by altering how the robot falls. Fujiwara et. al. [5] allows the robot to land on the robot knees or arms. A similar trajectory planning approach is proposed in [19]. Ha and Liu [20] minimize the falling damage using multiple contact points and optimize a sequence of contacts using a variant of a 2-D inverted pendulum model. Goswami proposes a tripod contact strategy to reduce the robot's kinetic energy during the fall before the impact [6]. However, the issue with each of these techniques is a long computation time, reported to be over 1 s and in some cases 15 s [5], [19], [20], [6]. Our technique uses a simplified 3-link dynamics model and precomputed databases to achieve fast optimization.

Heuristic approaches have been used in past work to achieve real-time performance. Tam and Kottee propose the use of walking sticks to prevent forward falling [21]. Their works uses forward fall detection and a heuristic quadrupedal posture to stop the robot from falling. In Yi et al, a machine learning approach is used to select between predefined push recovery controllers for a small humanoid robot, including use of the ankle, hip, or stepping [4]. A more comprehensive learning approach was taken in Kumar et. al., in which reinforcement learning is used to find a full body fall-recovery policy that can use protective stepping or hands, and this outperforms direct optimization by orders of magnitude [22]. However, all of these approaches work only on flat ground. In our work, the shape of the environment is not constrained and its geometric shape is used by the falling controller to select an appropriate limb and contact point.

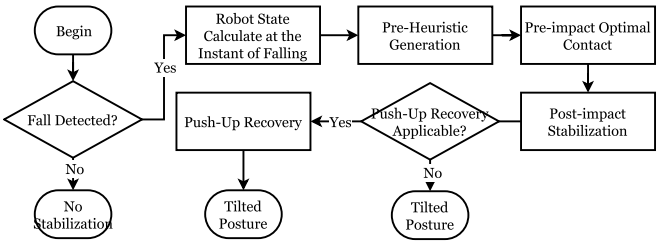


Fig. 2: Flowchart of the proposed system.

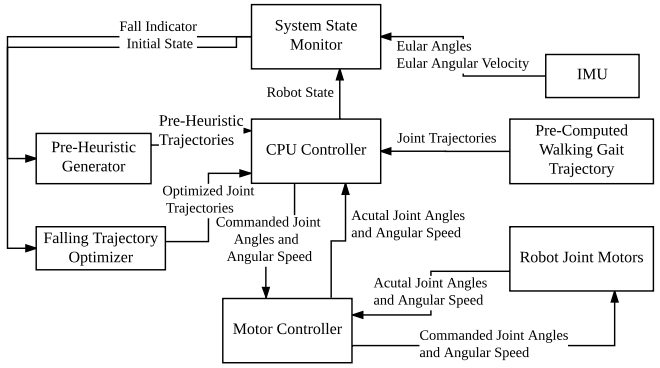


Fig. 3: Diagram of the hardware and control flow of the proposed system.

III. Method

A. Summary

The system that we implement in this paper is illustrated in Fig. 2. Our method is model-based and makes the following assumptions:

1) The *falling plane*, which is the plane containing the center of mass velocity and the gravity vector, is limited either to the sagittal plane or the frontal plane.
2) The shape of the nearby environment in the falling plane is known.
3) Centers of masses and inertia matrices of the robot's links are known.
4) The hand makes contact on the environment at a single point of contact, and this point of contact is inelastic.
5) In the push-up recovery stage, the robot elbow can generate enough momentum to bring the robot away from the wall.

With these assumptions, the proposed system monitors the state of the robot, stabilizes the falling robot after a fall is detected, and allows a push-up mechanism to recover the robot to a standing posture if the robot is not tilted too far. The diagram of the hardware and control flow of the proposed system is in Fig. 3.

B. Hardware Setup and System Integration

1) Darwin Mini Robot: Our experimental platform is the ROBOTIS Darwin Mini, a small humanoid robot with 16 actuated joint motors (Dynamixel XL 320). The motor encode can read the motor's present angle and angular velocity. These motors can be directly controlled by the OpenCM 9.04 C board featuring a 32-bit ARM Cortex-M3 processor. The baudrate between the board and motor

is configured to be 1Mbps. This board supports the serial data transmission via Tx, Rx and Bluetooth (BT210) with a maximum baudrate of 115200bps. Two ROBOTIS touch sensors are connected to this board to monitor the status of the robot hand contact.

2) Raspberry Pi 3: Raspberry Pi 3 (Rpi3) is chosen as the main CPU due to its size, cost and sufficient computational capability. Raspbian Jessie is the OS for Rpi3. To increase the efficiency of the data transmission between the robot and Rpi3, the two microcontroller boards are connected in both Bluetooth (OpenCM sending data to Rpi3) and Tx, Rx (Rpi3 sending data to OpenCM). This central computational unit undertakes all the calculation including fall detection and pre-impact optimal controller computation.

3) IMU-Adafruit BNO055: The IMU provides readings of Euler angles and angular velocities at 100Hz. However, the default connection port between the Rpi3 and BNO055 has been occupied for the data transmission between RPi3 and OpenCM. An additional Arduino UNO board is used to read the IMU data and send the data via Universal asynchronous receiver-transmitter (UART) to Rpi3 with a 115200 bps.

C. Fall Detection

Fall detection is a critical component of fall recovery. The earlier an inevitable fall is accurately detected, the more likely the robot will be stabilized. We use an inverted pendulum model for fall detection due to its simplicity to satisfy the strict real-time requirement. This pendulum is modeled from the foot edge where the robot rotates around to the center of the mass of the robot (Fig. 4). The foot edge location is estimated via kinematics given the current IMU and joint encoder readings.

In the falling plane, if the robot is not falling, the kinetic energy of the robot at each time t is less than the critical kinetic energy T_{crit} which is the minimum kinetic energy that the inverted pendulum needs to move from its current angle position to its unstable equilibrium point.

Therefore,

$$T_{crit} = MgL(1-\cos(\gamma)) \tag{1}$$

where M is the total mass of the robot, g is the gravitational constant, L is distance from the foot edge to the center of mass of the robot and γ is angle between the pendulum and the vertical axis in the falling plane, .

Meanwhile, the kinetic energy T_t at each instant of time t is

$$T_t = \frac{1}{2}ML^2\dot{\zeta}^2 + \frac{1}{2}I_F\dot{\zeta}^2 \tag{2}$$

where ζ is the angle between the pendulum and the horizontal axis, $\dot{\zeta}$ is the derivative of ζ and I_F is the equivalent moment of inertia at the rotation axis in the falling plane. If $T_t > T_{crit}$, then the robot is going to fall.

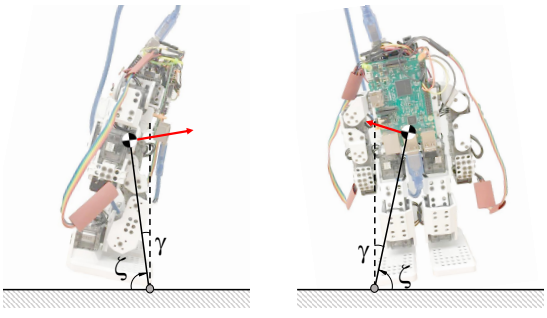
D. Modeling the Falling Trajectory

Following [7], a rigid three-link model is used to approximate the robot dynamics in the falling plane (Fig. 5). This model is chosen for the following reasons:

1) It is computationally light to satisfy the real-time computation requirement,
2) It is complex enough to allow for the use of pre-impact inertia shaping to reduce impact.
3) It models the use of post-impact compliance to achieve closed loop stability.

We divide the falling motion into two phases: 1) *pre-impact*, where the robot is pivoting about an edge of its foot and its hand is brought to touch the environment, and 2) *post-impact*, where the hand is in contact with the environment and any residual velocity is dampened.

After a fall has been detected, the pre-impact optimal controller will be computed to bring the robot hand into contact with environment obstacles. However, this computation requires knowledge of the robot state at the instant of falling. The robot state is the state angles (θ, α, β) and state angular velocities $(\dot{\theta}, \dot{\alpha}, \dot{\beta})$ where θ is the angle between the ground and stance leg, α is the angle between the stance leg link and the torso and β is the angle between the torso and contact arm. For a given falling plane, α and β are associated with actuated robot joint motors so the values of $\alpha, \dot{\alpha}, \beta$ and $\dot{\beta}$ are accessible with motor encoders. However, θ is an underactuated angle and cannot be directly measured. Therefore, an IMU is attached to the torso link to realize the



(a) Falling in Sagittal Plane (b) Falling in Frontal Plane

Fig. 4: Falling detection with inverted pendulum model in sagittal plane and frontal plane. The red arrow denotes the velocity at the center of the mass.

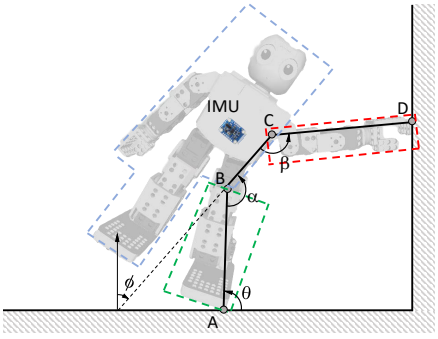


Fig. 5: Three-link model used in this work, illustrated in post-impact. The robot is divided into three blocks, the stance leg, torso, and contact arm, each of which is modeled as a rigid link. The free arm and swing leg are assumed to be fixed to the torso.

computation of θ. Based on the kinematics of the robot,

$$\begin{aligned}\theta &= 270^\circ - (\phi + \alpha) \\ \dot{\theta} &= -(\dot{\phi} + \dot{\alpha})\end{aligned} \tag{3}$$

where ϕ is the roll angle in the frontal plane or the pitch angle in the sagittal plane.

E. Optimal Stabilization with Hand Contact

The optimal stabilization process is divided into two subprocess: *Pre-impact optimal contact* and *Post-impact compliant stabilization*.

For the pre-impact system, our algorithm uses a direct shooting method to find an optimal controller to bring the robot into contact with the environment while minimizing the performance index function. This function describes the probability that at least one of the following three *catastrophic events* occurs:

Event 1: The impact from collision damages the robot.
Event 2: The foot contact point slips.
Event 3: The hand contact point slips.

The first event occurs when the impact I exceeds the critical amount I_{crit} needed to damage the robot. The second and third events occur, respectively, when the *necessary sticking friction* at the hands and feet μ_A and μ_D of a hypothetical trajectory exceed the actual environmental friction coefficients μ_{foot} and μ_{hand}. The terms I, μ_A, and μ_D are assumed deterministic functions of a given trajectory, while I_{crit}, μ_{foot}, and μ_{hand} are unknown, normally distributed random variables.

The performance index is chosen to be

$$J(\boldsymbol{q}_{tr}, \dot{\boldsymbol{q}}_{tr}, \boldsymbol{u}_{tr}) = 1 - P(I \leq I_{crit}) P(\mu_A \leq \mu_{foot}) P(\mu_D \leq \mu_{hand}) \tag{4}$$

where $\boldsymbol{q}_{tr}$ is the configuration trajectory, $\dot{\boldsymbol{q}}_{tr}$ is its time derivative, and the control trajectory is $\boldsymbol{u}_{tr}$. Lower values are better. The probability of each event is calculated using the cumulative distribution function (CDF) of a normal distribution with some given mean and standard deviation. Specifically, for each of the three values X, $P(y \leq X)$ is 1 minus the value of the CDF(y) of a normal distribution model $X \sim N(Avg(x), Std(x)^2)$ with mean $Avg(x)$ and standard deviation $Std(x)$. Here, $Avg(x)$ and $Std(x)$ can be coarsely estimated by a human engineer (lower values lead to a more conservative controller) or measured empirically.

In the shooting method, $\boldsymbol{u}_{tr}$ is taken as a piecewise constant control sequence of length N, with each segment lasting for dt. With given control sequence, an integration of the pre-impact dynamics is conducted until the collision is detected. The Lagrangian dynamics of the pre-impact system is

$$D(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}}) + G(\boldsymbol{q}) = B\boldsymbol{u} \tag{5}$$

where $D(\boldsymbol{q}) \in \mathbb{R}^{3\times3}$ is the generalized inertia matrix. $C(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{3\times1}$ is the centrifugal and coriolis forces matrix. $G(\boldsymbol{q}) \in \mathbb{R}^{3\times1}$ is the generalized gravity matrix. $\boldsymbol{u} \in \mathbb{R}^{2\times1}$ is the joint torque vector for α and β, and $B \in \mathbb{R}^{3\times1}$ is the input matrix mapping the joint torques to generalized coordinates.

An impact mapping function maps the pre-impact state to the post-impact state and enables the computation of the collision impulse whose magnitude I is used to evaluate the probability of robot damage [23].

The post-impact system is in fact a four-bar linkage system with only one degree of freedom and can be stabilized with one control. Equation of motion for post-impact system is

$$D(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}}) + G(\boldsymbol{q}) = B' u_\beta + E(\boldsymbol{q})^T \boldsymbol{\lambda} \tag{6}$$

where $E(\boldsymbol{q})$ is the jacobian matrix of the constraint equations with respect to the state variables and $\boldsymbol{\lambda}$ is a 2×1 vector of Lagrange multipliers. After the feedback linearization, (6) is expressed a linear function of u_β:

$$\ddot{\beta} = f_\beta(\beta, \dot{\beta}) + g_\beta(\beta, \dot{\beta}) u_\beta \tag{7}$$

where u_β is the torque applied at joint C, $g_\beta(\beta, \dot{\beta})$ is the term associated with the control u_β and $f_\beta(\beta, \dot{\beta})$ is the remaining term.

Let

$$\ddot{\beta} = -K\dot{\beta} \tag{8}$$

where K is the derivative gain of this stabilizing controller. This value is chosen such that $\forall|\dot{\beta}| \in [\dot{\beta}_{min}, \dot{\beta}_{max}]$, $|K\dot{\beta}|$ remains within the joint acceleration bound $\ddot{\beta}_{max}$.

This controller yields an analytic expression of how the post-impact system evolves

$$\beta(t) = \beta^+ + \frac{\dot{\beta}^+}{K}(1 - e^{-Kt}) \tag{9}$$

where β^+ is the post-impact angle, $\dot{\beta}^+$ is the post-impact angular velocity.

Then the *necessary sticking friction coefficients* can be computed along the post-impact joint trajectories and used in the performance index to evaluate the probability of contact slipping. We use a precomputed database of the necessary sicking friction coefficients at all possible post-impact state to reduce the computation time and realize the real-time optimal controller computation. For more information about the control sequence initialization and necessary sticking friction coefficients computation, please refer to [7].

F. Pre-Heuristic Trajectory Execution

Based on the empirical observations of the computational time of the optimal controller, it takes around 100 ms to calculate the optimal control sequence and joint trajectories for robot hand and hip with numerical integration. Those trajectories will be used as references for robot joint motors to track to reach an optimal contact. However, for a small size humanoid, the falling time can be as short as 400 ms. It is possible that the joint takes more than 300 ms to move from the falling time angle position to the optimal angle position. As a consequence, the robot would have already fallen to the ground before the optimal contact can be made. Thus, to save the time of tracking optimal joint trajectory, the robot should execute a pre-heuristic trajectory after the fall has been detected. The method to compute this trajectory is using the numerical integration of the pre-impact system with the actuated joint torques determined by a heuristic inverse dynamics approach. The dynamics equations of the

two actuated joints are

$$\begin{bmatrix} \ddot{\alpha} \\ \ddot{\beta} \end{bmatrix} = \begin{bmatrix} -k_{P\alpha}(\alpha - \alpha_{ref}) - k_{D\alpha}\dot{\alpha} \\ -k_{P\beta}(\beta - \beta_{ref}) - k_{D\beta}\dot{\beta} \end{bmatrix} \quad (10)$$

where the values of $k_{P\alpha}, k_{P\beta}, k_{D\alpha}, k_{D\beta}, \alpha_{ref}$ and β_{ref} are determined heuristically.

This pre-heuristic trajectory stretches the robot arm near a possible optimal arm position to reduce the tracking time while the optimal controller is being computed. After the latter has been computed, the robot joint will switch to track the optimal joint trajectory. To transit smoothly from the pre-heuristic trajectory to the optimal joint trajectory, the final state on the pre-heuristic trajectory should be the initial state of the optimal joint trajectory. The whole pre-heuristic trajectory execution time t_{pre} is the maximum time (150 ms) that our algorithm needs to finish computing the optimal controller and its value can be determined empirically. The optimal joint trajectories will not be use until t_{pre} time has elapsed from the instant of falling.

G. Push-up Recovery

After fall stabilization, the robot will remain in a steady state and can either 1) wait to be relocated by human to start a new gait, or 2) recover to an upright position by pushing off of the wall. We use a flexing motion of the elbow to allow the robot to gain sufficient momentum to recover a standing posture.

Suppose the hand contact point remains fixed during the push recovery process. Again we apply a three-link model to analyze this push recovery motion, where the three links are the body link (foot rotational point to the shoulder), upper arm (shoulder to elbow) and lower arm (elbow to hand), θ^* is the angle between the ground and the body link, α^* is the angle between the body link and upper arm and β^* is the angle between the upper arm and the lower arm (Fig. 6). The mass of the arm is assumed negligible compared to the mass of the body so the behavior of the robot can be approximated as an inverted pendulum.

To gain the momentum needed to move away from the environment support, the robot can first bend its elbow to reach an angle β^*_{min} and then stretch the elbow angle to its initial value $\beta^* = 180^\circ$ with a commanded angular velocity $\dot{\beta}^*_{cmd}$. In this paper, we use a heuristic approach to determine this distance range, β^*_{min} and $\dot{\beta}^*_{cmd}$. However, this mechanism works under a constraint of the robot tilt angle in the falling plane. If that angle is beyond a feasible range, the push up will not be able to generate enough momentum for the robot to recover.

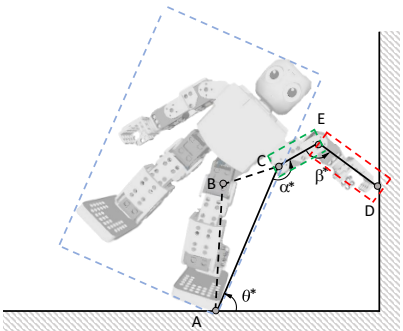


Fig. 6: Three-link model used for push-up recovery. The robot is divided into three blocks, the body, upper arm, and lower arm, each of which is modeled as a rigid link. The upper arm is assumed to be fixed to the body. The two black dashed lines are the stance leg link and torso link from the post-impact model.

IV. Experimental Evaluation

We evaluate our system on 8 scenarios. The robot parameters used are shown in Table I, and Table II shows the weight coefficients and heuristic values used in the experiment. Note that the robot has no visual sensors; its local environment geometry relative to the robot's initial pose is given as input to our system, and the wall shape in the the falling plane is determined once a fall is detected.

The eight scenarios illustrate forward and sideways falling on four environments. **Vertical 1** and **Vertical 2** are vertical walls at distance 0.19 m and 0.22 m, respectively. **Table a** and **Table b** use a flat surface at height of 0.05 m. **Ground a** and **Ground b** use the flat ground surface. **Push-Up a** and **Push-Up b** use a vertical wall at distance 0.19 m and the robot remains stationary in the former case. In all case except **Push Up a**, the robot performs a treadmill like walking gait which is walking without globally moving forward. This gait is chosen to reduce the complexity of the determination of falling direction when a push is given because the forward walking momentum may lead to a false fall detection on sagittal plane given a push in frontal plane. We leave the problem of falling plane detection during navigation for future work.

In all cases except for **Table** and **Ground**, the control is initialized with heuristic target values $\alpha_{ref} = \alpha_{ref1}$ and $\beta_{ref} = \beta_{ref1}$. In **Table** and **Flat**, we use $\alpha_{ref} = \alpha_{ref2}$ and $\beta_{ref} = \beta_{ref2}$ as given in Tab. II. The robot state at the instant of falling for each experiment is in Tab. III. For each robot state, an optimal controller is computed in real-time to bring the robot into contact with the environment walls. Tab. IV shows the time for optimization (Comp. time), the time at which contact is made after fall detection (Contact time), the predicted collision impulse, necessary sticking friction coefficients at contact points (μ_A and μ_D), and optimized probability of failure. In all cases the computation time is within 150 ms and the optimal contact is made within 400 ms. The next four columns of this table show that besides the three extreme cases (**Vertical b**, **Ground a** and **Ground b**), our optimal controller manages to confidently reduce the probability that a *catastrophic event* would occur.

Fig. 7 illustrates the trace for each experiment. In each example, the robot is stopped using hand contact with the

TABLE I: Model Parameters

	Mass (kg)	Length (m)	Moment of Inertia ($kg \cdot m^2$)
Leg	0.12	0.125	3.1677e-04
Torso	0.53	0.070	8.8277e-04
Arm	0.05	0.100	4.4271e-05

TABLE II: Misc. Coefficients and Heuristic Values

Controller gains		Target values		Failure parameters	
k_{P_α}	250	α_{ref1}	$5\pi/6\,rad$	$Avg(I)$	$0.45\,N\cdot s$
k_{P_β}	250	β_{ref1}	$5\pi/8\,rad$	$Std(I)$	$0.1\,N\cdot s$
k_{D_α}	20	α_{ref2}	$\pi/2\,rad$	$Avg(\mu_A)$	0.6
k_{D_β}	20	β_{ref2}	$3.6\pi/5\,rad$	$Std(\mu_A)$	0.1
K	14	β^*_{min}	$5\pi/6\,rad$	$Avd(\mu_E)$	0.5
		$\dot{\beta}^*_{cmd}$	$10\,rad/s$	$Std(\mu_E)$	0.1

TABLE III: Robot State at the Instant of Falling

Problem	θ (rad)	α (rad)	β (rad)	$\dot{\theta}$ (rad/s)	$\dot{\alpha}$ (rad/s)	$\dot{\beta}$ (rad/s)
Vertical a	1.48	2.61	0.52	-0.62	0.19	0.00
Vertical b	1.43	2.59	0.52	-1.09	-0.14	0.00
Table a	1.39	2.60	0.52	-3.22	-0.18	0.00
Table b	1.66	2.89	0.17	-2.71	0.24	0.66
Ground a	1.41	2.65	0.52	-2.41	0.14	0.00
Ground b	1.69	2.96	0.14	-3.24	-0.81	1.85
Push Up a	1.47	2.67	0.52	-0.54	0.14	0.00
Push Up b	1.49	2.62	0.52	-0.55	-0.19	0.00

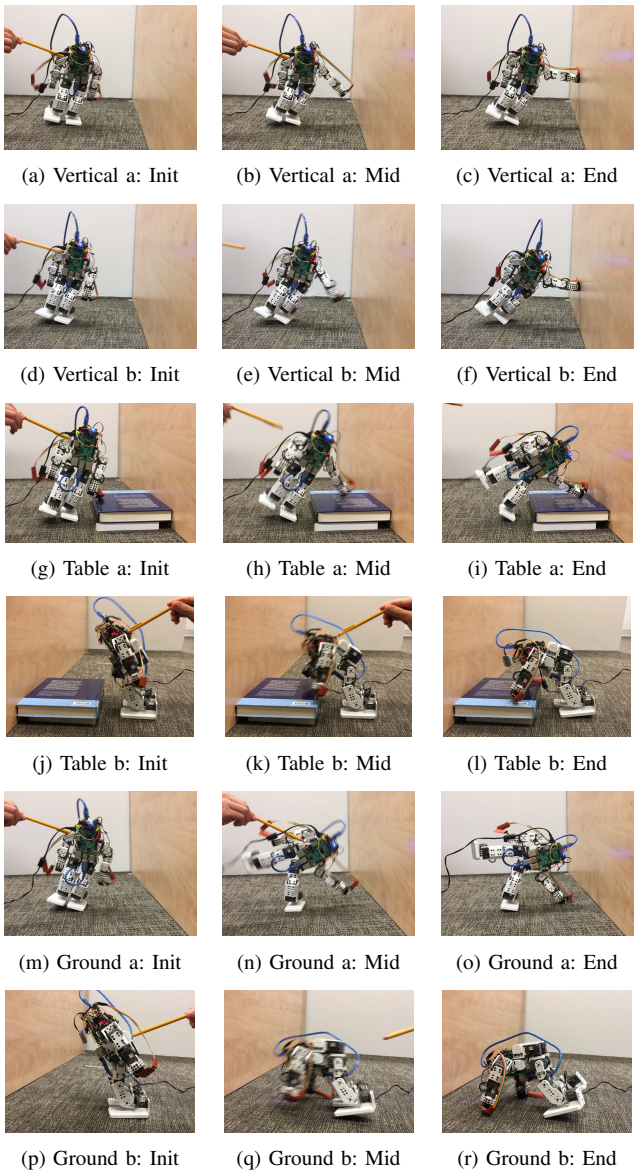
environment obstacles. Comparing **Vertical a** to **Vertical b**, the robot reaches its hand towards the wall to arrest itself from falling and reaching is further for walls at further distance. Comparing **Table a** and **Table b**, our method can exploit the flat table surface to stabilize the robot and extend the falling stabilization strategy to both the sagittal plane and frontal plane instead of the sole 2-D sagittal plane. Comparing **Ground a** and **Ground b**, our method finds the optimal contact with a bent-over posture in which the robot's center of mass remains relatively high above the ground. This result is consistent with prior work on optimal fall mitigation strategy to minimize collision damage [24]. **Push Up a** and **Push Up b** demonstrate that our proposed push-up strategy can recover the robot to a standing posture and enable the robot to continue on its previous task interrupted by the push. Fig. 8 and Fig. 9 demonstrate this recovery process. Fig. 10 shows a representative diagram of the timing of when the fall is detected, when the pre-heuristic trajectory is executed, and when the optimal trajectory is executed to demonstrate the real-time nature of the proposed strategy.

V. CONCLUSION

We presented a system for stabilizing a falling robot using hand contact on environmental features. It is based on a

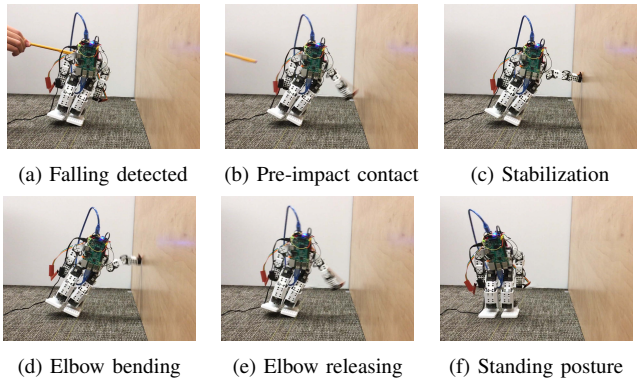
TABLE IV: Model-based optimization results for each experiment. Objective values are shown as a % of the uncontrolled value.

Problem	Comp. time (ms)	Contact time (ms)	Impact $(N\cdot s)$	μ_A	μ_E	Optimized P failure (%)
Vertical a	23	366	0.184	0.503	0.102	17.04
Vertical b	113	358	0.326	0.804	0.429	98.56
Table a	29	386	0.326	0.338	0.304	13.40
Table b	110	391	0.206	0.226	0.110	0.73
Ground a	144	361	0.494	0.351	0.116	67.32
Ground b	54	385	0.527	0.299	0.269	78.10
Push Up a	119	353	0.191	0.548	0.068	30.76
Push Up b	83	368	0.196	0.527	0.083	23.66



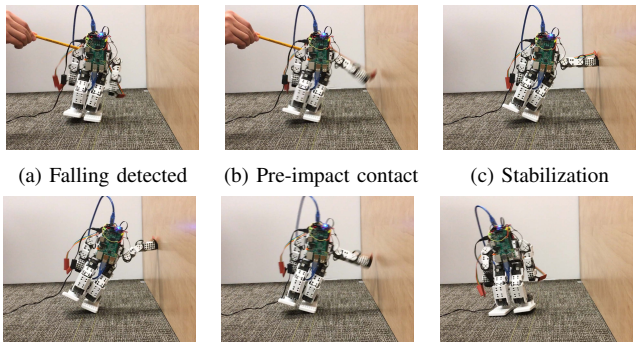
(a) Vertical a: Init (b) Vertical a: Mid (c) Vertical a: End
(d) Vertical b: Init (e) Vertical b: Mid (f) Vertical b: End
(g) Table a: Init (h) Table a: Mid (i) Table a: End
(j) Table b: Init (k) Table b: Mid (l) Table b: End
(m) Ground a: Init (n) Ground a: Mid (o) Ground a: End
(p) Ground b: Init (q) Ground b: Mid (r) Ground b: End

Fig. 7: Experiments of optimal controllers for examples **Vertical 1** to **Ground b**. For each row, the left figure is the robot state at the instant of falling denoted as Init, the middle figure is the transition motion denoted as Mid and the right one is the stabilized motion denoted as End.



(a) Falling detected (b) Pre-impact contact (c) Stabilization
(d) Elbow bending (e) Elbow releasing (f) Standing posture

Fig. 8: Representative traces of **Push Up a** case recovery motion



(a) Falling detected (b) Pre-impact contact (c) Stabilization

(d) Elbow bending (e) Elbow releasing (f) Standing posture

Fig. 9: Representative traces of **Push Up b** case recovery motion

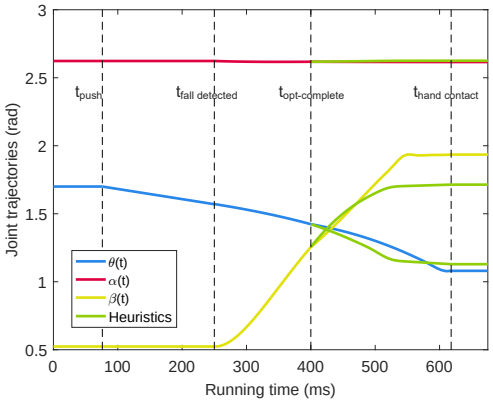


Fig. 10: A representative diagram of the timing using **Push Up b** case. Vertical dashed lines denote the timing of when the robot is pushed t_{push}, when the fall is detected $t_{fall\ detected}$, when the computation of optimal controller is completed $t_{opt\text{-}complete}$ and when the contact is established $t_{hand\ contact}$. Green curves are the heuristic trajectories used in the pre-impact optimization.

three-link dynamic model and optimal control principles to choose a hand contact point. It also overcomes several practical challenges regarding computation and communication delays. Experimental results on the Darwin Mini platform, augmented with additional sensors, show that the proposed falling stabilization system can stabilize the robot with several walls, environmental obstacles, and falling directions. Moreover, a push-recovery strategy can in some cases recover the robot to a standing posture.

There are two directions we hope to explore in the future. First, we have not yet addressed fall detection while forward walking, or the high-level decision about which fall recovery strategy – hand contact, inertia shaping, or protective stepping – should be used upon fall detection. Second, we hope to formulate the push-up recovery method into a more analytical determination whether recovery is feasible. If push-up recovery is infeasible, perhaps the robot could perform a movement that lowers it to the ground in a controlled fashion before recovery from a fallen position.

REFERENCES

[1] J. E. Pratt and G. A. Pratt, "Exploiting Natural Dynamics in the Control of a Planar Bipedal Walking Robot," in *Proceedings of the Thirty-Sixth Annual Allerton Conference on Communication, Control, and Computing*, no. September, 1998.

[2] T. McGeer, "Stability and control of two-dimensional biped walking," in *Technical Report 1.*, Center for Systems Science, Simon Fraser University, Burnaby, B.C., Canada, 1988.

[3] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, "Capture Point: A Step toward Humanoid Push Recovery," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*. IEEE, dec 2006.

[4] S.-J. Yi, B.-T. Zhang, D. Hong, and D. D. Lee, "Online learning of low dimensional strategies for high-level push recovery in bipedal humanoid robots," in *IEEE International Conference on Robotics and Automation*. IEEE, may 2013.

[5] K. Fujiwara, S. Kajita, K. Harada, K. Kaneko, M. Morisawa, F. Kanehiro, S. Nakaoka, and H. Hirukawa, "Towards an Optimal Falling Motion for a Humanoid Robot," in *6th IEEE-RAS International Conference on Humanoid Robots*. IEEE, dec 2006.

[6] S.-k. Yun and A. Goswami, "Tripod fall: Concept and experiments of a novel approach to humanoid robot fall damage reduction," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2014.

[7] S. Wang and K. Hauser, "Real-time stabilization of a falling humanoid robot using hand contact: An optimal control approach," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, Nov 2017.

[8] *Darwin Mini*, ROBOTIS INC Std. [Online]. Available: http://www.robotis.us/robotis-mini-intl/

[9] S. Kajita, T. Yamaura, and A. Kobayashi, "Dynamic walking control of a biped robot along a potential energy conserving orbit," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 4, 1992.

[10] S. Kalyanakrishnan and A. Goswami, "Learning to Predict Humanoid Fall," *International Journal of Humanoid Robotics*, vol. 08, no. 02, jun 2011.

[11] R. Renner and S. Behnke, "Instability detection and fall avoidance for a humanoid using attitude sensors and reflexes," *IEEE International Conference on Intelligent Robots and Systems*, 2006.

[12] J. Karssen and M. Wisse, "Fall detection in walking robots by multi-way principal component analysis," *Robotica, 27 (2009)*, 2008.

[13] K. Ogata, K. Terada, and Y. Kuniyoshi, "Real-time selection and generation of fall damage reduction actions for humanoid robots," in *IEEE-RAS International Conference on Humanoid Robots*. IEEE, dec 2008.

[14] S. Kajita and K. Tani, "Study of dynamic biped locomotion on rugged terrain-derivation and application of the linear inverted pendulum mode," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, 1991.

[15] B. Stephens, "Humanoid push recovery," in *7th IEEE-RAS International Conference on Humanoid Robots*, Nov 2007.

[16] M. Missura and S. Behnke, "Omnidirectional capture steps for bipedal walking," in *13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, oct 2013.

[17] V. Samy, K. Bouyarmane, and A. Kheddar, "Qp-based adaptive-gains compliance control in humanoid falls," in *IEEE International Conference on Robotics and Automation*, May 2017.

[18] E. M. Hoffman, N. Perrin, N. G. Tsagarakis, and D. G. Caldwell, "Upper limb compliant strategy exploiting external physical constraints for humanoid fall avoidance," in *13th IEEE-RAS International Conference on Humanoid Robots*, Oct 2013.

[19] Jiuguang Wang, E. C. Whitman, and M. Stilman, "Whole-body trajectory optimization for humanoid falling," in *American Control Conference (ACC)*. IEEE, jun 2012.

[20] Sehoon Ha and C. K. Liu, "Multiple contact planning for minimizing damage of humanoid falls," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2015.

[21] B. Tam and N. Kottege, "Fall Avoidance and Recovery for Bipedal Robots using Walking Sticks," dec 2016.

[22] V. C. Kumar, S. Ha, and C. K. Liu, "Learning a Unified Control Policy for Safe Falling," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, mar 2017.

[23] Y. Hurmuzlu and D. B. Marghitu, "Rigid Body Collisions of Planar Kinematic Chains With Multiple Contact Points," *The International Journal of Robotics Research*, vol. 13, no. 1, feb 1994.

[24] A. Goswami, S.-k. Yun, U. Nagarajan, S.-H. Lee, K. Yin, and S. Kalyanakrishnan, "Direction-changing fall control of humanoid robots: theory and experiments," *Autonomous Robots*, vol. 36, no. 3, mar 2014.