Using Previous Experience for Humanoid Navigation Planning

Yu-Chi Lin[1] and Dmitry Berenson[1]

***Abstract*— We propose a humanoid robot navigation planning framework that reuses previous experience to decrease planning time. The framework is intended for navigating complex unstructured environments using both palm and foot contacts. In a complex environment, discrete-search-based contact space planners trade-off between high branching factor and action flexibility. Although approaches such as weighted A*, ARA* and ANA* could speed up the search by compromising on optimality, they can be very slow when the heuristic is inaccurate. In the proposed framework, an experience-retrieval module is added in parallel to ANA*. This module collects previously-generated motion plans and clusters them based on contact pose similarity to form a motion plan library. To retrieve an appropriate plan from the library for a given environment, the framework uses a distance between the contact poses in the plan and environment surfaces. Candidate plans are then modified with local trajectory optimization until a plan fitting the query environment is found. Our experiments show that the proposed framework outperforms planning-from-scratch in success rate in unstructured environments by at least 28% and can navigate difficult environments such as rubble and narrow corridors.**

I. Introduction

Humanoid robots designed for disaster response are expected to operate in very complicated and unstructured environments with limited rescue time. Although a robot can be commanded by a human with remote control, with limited communication bandwidth, the robot still needs to plan its motion quickly. When humans navigate in an unstructured environment, such as stepping through rubble, we not only find appropriate footsteps, but also make contact with our hands to achieve more stable movement. Using both palm and foot contacts is also important when the robot is deployed in an environment subject to disturbances, such as aboard a ship. However, existing discrete-search-based contact space planners suffer from the high branching factor induced by using both palms and feet. The approach proposed in this paper adds an experience-retrieval module in parallel to ANA* [1], which reduces the planning time by reusing previous experience.

In the proposed framework, there are two modules running in parallel: the Planning-from-Scratch (PFS) module and the Retrieve and Adapt (RA) module, as shown in Figure 1. Planning in Configuration Space (C-Space) for humanoid locomotion is very computationally expensive due to the high number of degrees of freedom. Therefore, contact space planners [2]–[8] trade the completeness properties of planning in C-space for the computational efficiency of planning in contact space. Our PFS module likewise follows this approach by first planning in contact space without

[1]Yu-Chi Lin and Dmitry Berenson are with the University of Michigan linyuchi@umich.edu, berenson@eecs.umich.edu

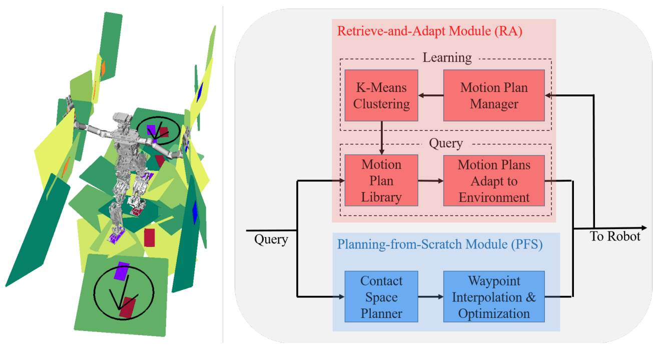

Retrieve-and-Adapt Module (RA)
Learning
K-Means Clustering
Motion Plan Manager
Query
Motion Plan Library
Motion Plans Adapt to Environment
Query
To Robot
Planning-from-Scratch Module (PFS)
Contact Space Planner
Waypoint Interpolation & Optimization

Fig. 1. Left: A humanoid follows a planned sequence of contact poses to navigate in a complex unstructured environment modeled as a set of contact regions. Right: The structure of the proposed framework

considering collision with the environment (except for the end-effector links). The resulting sequence of contact poses is then interpolated, inverse kinematics is computed, and the entire sequence of configurations is optimized locally to avoid obstacles. We call the output of this process a ”motion plan.” A feasibility database is also introduced to speed up the evaluation of state feasibility within the planner.

RA, on the other hand, provides solutions by retrieving motion plans from a library. RA stores and clusters motion plans generated by the framework to form a motion plan library based on the contact poses of each plan. Given a new environment, RA queries the library to find an appropriate plan based on its contact poses and modifies it to fit the environment. Both modules start planning simultaneously and the one that finishes first stops the other one. Finally, the generated motion plan is added to the library if it differs significantly from other plans in the library.

The main contributions of this paper are: (1) A framework for building humanoid robot motion plan libraries based on contact pose sequences; (2) A fast distance function for retrieving feasible plans from the library for a new environment; and (3) A feasibility database that speeds up state feasibility checks during planning.

Our experiments show that the proposed framework achieves a higher success rate in unstructured environments compared to planning-from-scratch. Additionally, the framework is agnostic to the PFS module, so new developments in navigation planning can be integrated easily.

II. Related Work

Reusing pre-computed plans has been studied in computer animation [9] and trajectory optimization [10]. However, using path libraries for high-dimensional humanoid locomotion planning with balance and collision constraints and both hand and foot contact has not yet been explored.

Robot motion libraries have been used to speed up motion planning in C-Space [11]. However, the distance metric of [11] is not adequate in our context because it does not consider contact with the environment, which is key for humanoid robot navigation. Recently [12] improved on [11] by storing the experience in a sparse roadmap spanner. However, humanoid navigation involves multiple contact switches, necessitating that the robot travel through manifolds of differing dimension, which cannot be done with this sparse roadmap spanner. [13] also proposed a motion plan library framework by learning the mapping between environments and plans. However, this approach may overlook plans that come from an environment which is not similar to the query environment, but are nevertheless a good fit.

Humanoid footstep planning has been studied extensively: [2]–[6] are discrete-search-based approaches, which use A*-like algorithms. We also use such an algorithm in the planning-from-scratch module. Here we address both palm and foot contact, which induces a very high branching factor in the search, causing search-based techniques to perform slowly in difficult contact scenarios. There are also optimization approaches, which deform an existing series of steps to follow the constraints [7], [8]. Our path library can provide a good path to initialize these kinds of methods.

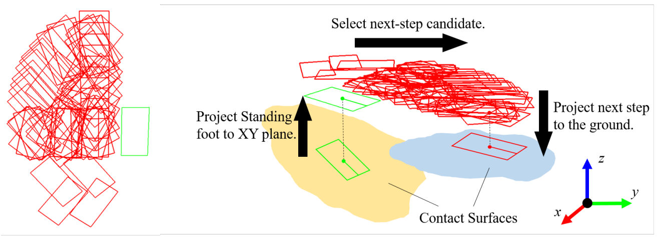
For humanoid navigation in unstructured environments using multiple contacts, [14] used optimization to find contacts in the neighborhood of a "rough" trajectory. However, its planning time is prohibitively long. [15] proposed a humanoid robot planner that used learned motion primitives. Unlike their approach, we focus on the retrieval and adaptation of the full motion plan from the start to the goal. Contact-consistent Elastic Strips (CES) [16] combined discrete-search-based contact space planning with a local trajectory optimizer. They generated an initial trajectory only obeying the reachability constraint, and locally optimize it to be feasible. However, they do not consider the use of previous experience. In our framework, we use CES to adapt motion plans to the query environment.

III. Problem Statement

We address the humanoid navigation planning problem. Given an environment represented as a set of contactable surfaces, we wish to output a feasible trajectory from the start configuration to a goal region, defined as an area the feet must be within. We are interested in using the hands to help balance the robot against potential disturbances. Therefore, a feasible path should have at least three end-effectors in contact at any time, and must obey balance and collision constraints. We assume (as in [17]) that the robot can generate sufficient torque to balance itself. We also assume the friction coefficients are known.

IV. Method Overview

Our framework consists of two modules running in parallel: PFS and RA (see Figure 1). PFS plans a contact sequence from start to goal with Anytime Non-parametric A* [1], and interpolates the contact sequence to form a trajectory which is then optimized with the CES algorithm [16].


Select next-step candidate.
Project Standing foot to XY plane.
Project next step to the ground.
Contact Surfaces
z
y
x
Fig. 2. Left: Step transition model used in PFS module. (57 steps) Right: The projections to get the next step pose.

RA, which is the focus of this paper, consists of two parts: learning (i.e incrementally building the library) and querying the library. In the first part, the plans generated by the motion planner are collected. Note that we do not store the environment corresponding to each motion plan, which saves memory. The motion plans are clustered into small groups based on the contact pose distance between plans using the K-means algorithm. In each cluster, the motion plan with the minimum sum of squared distances to other motion plans in the cluster is selected as the cluster representative.

In the query part, each cluster representative is modeled as a series of contacts, and the query environment is modeled as a set of contact regions. The RA module determines whether plans inside the cluster are worth searching by calculating the contact distance between the cluster representative and the environment, and tests the clusters in order of increasing distance. Finally, the motion plans in the best untested cluster are mapped to the environment and modified with CES[16] one-by-one until a feasible plan is found. We describe PFS and RA in greater detail below.

CES [16] is a local trajectory optimization algorithm which focuses on multi-contact motions based on null-space projection of Jacobian matrices. The primary task in CES is to ensure that the contact poses of consecutive configurations converge to the same pose when in contact. Movements for collision avoidance and maintaining balance are projected into the null-space of this primary task.

V. Planning-from-Scratch Module

The PFS module first plans a sequence of contacts using ANA* and then uses inverse kinematics and CES to construct a feasible trajectory from the contacts. The contact space ANA* planner considers reachability, balance, and self-collision constraints. To verify if the constraints are met for a given set of contact poses, Inverse Kinematics (IK) is required. Computing IK and checking balance at every planner node is computationally expensive. To speed up the process, we introduce a feasibility database to store and reuse the results of previous IK and balance queries.

A. Contact Space Planning

In the PFS module, the contact space planning problem includes both palm and foot contacts, and is formulated as a graph search, which is solved with ANA*. The state of the planner is defined as the set of the contact poses corresponding to each end-effector. The end-effectors should be on one of the contact regions, and free from collision with all the surfaces except for the contact surface. The robot should also be in static balance in every configuration.

An action in the planner is defined as the robot switching one of the end-effectors' contact poses. Given a state, the possible next actions are described as a pre-defined transition model relative to the current end-effector poses. We do not specify the order of end-effector transitions other than requiring that the same end-effector is not used in consecutive actions. To determine the next foot contact, we first project the standing foot contact pose to the XY plane along the global Z axis, use the transition model to find the next step in the XY plane, and then project the pose to the ground to get the next foot contact pose, as shown in Figure 2.

For palm contacts, we first approximate the shoulder position based on the foot poses, and project palm contacts from the approximated shoulder point to the environment to get the possible next palm contact poses.

For each action a, the edge cost Δg is defined as:

$$\Delta g(a) = d_e(a) + w_\theta d_\theta(a) + w_s \tag{1}$$

where d_e is the translation of the moving end-effector, d_θ is the difference in robot orientation (defined as the mean of the two feet's rotation about the Z axis), and w_θ and w_s are the weight for robot orientation difference and the step cost, respectively. Adding step cost helps reduce the number of steps used in the plan.

The heuristic for each state x used in the planner is:

$$h(x) = w_\theta h_\theta(x) + \sum_{i}^{|\mathit{end\text{-}effectors}|} h_{e,i}(x) + w_s \frac{h_{e,i}(x)}{d_{e,i,max}} \tag{2}$$

where h_θ is the difference between the current and goal robot orientations, $h_{e,i}$ is the Euclidean distance between the pose of end-effector i and the goal, and $d_{e,i,max}$ is the maximum possible translation for end-effector i in one action.

To obtain the final robot trajectory, the contact sequence returned by the contact space planner is interpolated with parabolic trajectories for each contact transition. IK is computed for the interpolated poses and the sequence of resulting configurations is then optimized with the CES algorithm to avoid obstacles in the environment.

B. State Feasibility Check

The contact space planner targets planning in unstructured environments. When expanding the planner tree, the discretized foot and palm contacts are projected to environment surfaces to get the contact poses. Therefore, we must decide a state's feasibility online.

Besides obeying joint limit and self-collision constraints, the robot must be able to reach the specified contact poses and maintain balance when moving from the parent state to the current state. A typical approach to address reachability in contact space planning is to derive a contact transition model in which all possible moves are within a conservative bound of reachability. Since we expect the robot to navigate in an unstructured environment, such a boundary is hard to derive without sacrificing significant reachability. To better describe the robot's reachability, we filter out impossible transitions with a loose bound based on the length of the manipulators, and then use Jacobian-based IK to directly check reachability. Joint limit and self-collision constraints are also checked.

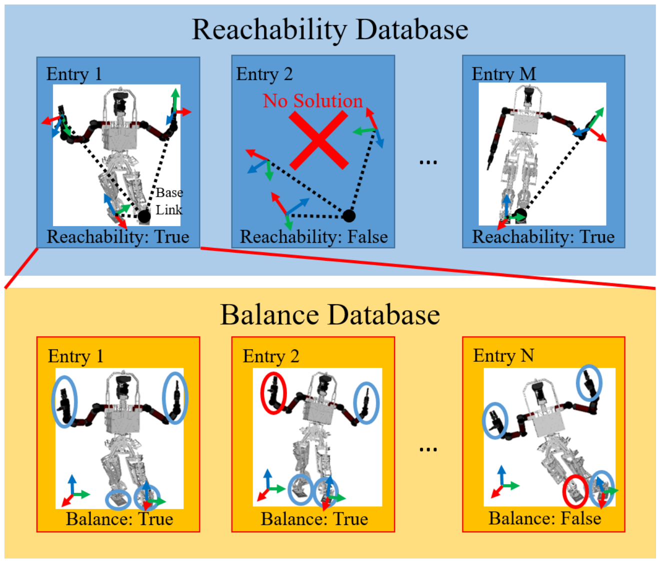


Fig. 3. The state feasibility database. The blue/red mark on the end-effectors represent whether the end-effector is/is not in contact.

Checking balance at a large number of configurations along a contact transition path is computationally expensive. To speed up this process, we approximate this check by only checking the beginning of the path where the foot/palm has just broken contact and the end of the path where the foot/palm is about to make contact. These are generally (though not always) the worst-case configurations for balance. Note that the final plan (after optimization) does check balance at a fine discretization for each step to ensure the plan is feasible. We use the method in [17] to check static balance with non-coplanar contacts.

C. Feasibility Database

Although IK and balance checking determine the feasibility of each state, this approach is slow and performs duplicate queries of (nearly) identical contact poses. To speed up the feasibility check, we record each IK query result to construct a feasibility database and reuse it when a similar set of contact poses is revisited.

The feasibility database is separated into two layers - the reachability database and the balance database, as shown in Figure 3. To encode the key for the reachability database, we first select an end-effector as the base link, and concatenate the relative pose of every other end-effector to the base link. All the dimensions of the keys are densely discretized, and every new entry is mapped to a discrete cell. If a query is mapped to an occupied cell, the feasibility database reuses the result in the cell. As the robot plans in the environment more and more times, it will collect more IK results, reducing the number of IK queries for each plan, and speeding up the feasibility query process.

Under the assumption of sufficient joint torque, static balance only depends on the poses of end-effectors in contact, the friction coefficient, and the position of the Center of Mass (CoM) [17]. If there exists a CoM for which the robot can be in static balance in a given state, this state obeys the balance constraint. Since the reachability database already encodes the relative poses of each end-effector to the base link, the balance database is constructed under each reachable entry of the reachability database by specifying the rotation of the base link in the world frame and which end-effectors are in contact with the environment.

The proposed feasibility database can be considered as a discretization of the contact space. In implementation, the databases are constructed as hash tables. Although the dimension is extremely high, the database is sparse, and built from past experience. Therefore, it can focus on a class of environments over time and need not densely cover the entire space of possible contact poses.

VI. Learning Part of the RA Module

To efficiently retrieve a feasible motion plan in a new environment, the RA module needs to identify promising plans in the library quickly. This is achieved by clustering the motion plans. Motion plans inside each cluster are represented by a cluster representative, so that the RA module can find promising motion plans by checking these cluster representatives instead of the entire library. We denote the motion plan library as L, which can also be represented as K clusters of motion plans: $L = [C_1, C_2, ..., C_K]$.

Each new motion plan P_{new} generated by the proposed framework is first examined by the motion plan manager. If the motion plan's distance to other motion plans in the library is above a user-defined threshold d_{min}, it will be added to the library. The algorithm used in the learning part of the RA module is shown in Algorithm 1.

Algorithm 1: Learning Part of the RA Module

```
$close\_to\_existing\_motion\_plan \leftarrow False$;
for $i$ in $1$ to $K$ do
    $C_i \leftarrow L[i]$;
    for $j$ in $1$ to $|C_i|$ do
        $P_{i,j} \leftarrow C_i[j]$;
        if $d(P_{i,j}, P_{new}) < d_{min}$ then
            $close\_to\_existing\_motion\_plan \leftarrow True$;
        end
    end
end
if not $close\_to\_existing\_motion\_plan$ then
    $L \leftarrow L \cup P_{new}$;
    $K \leftarrow 1$;
    do
        $[C_1, C_2, ..., C_K] \leftarrow$ K-means$(L, K)$;
        $L' \leftarrow [C_1, C_2, ..., C_K]$;
        $[d_{C_1}, d_{C_2}, ..., d_{C_K}] \leftarrow$ Get_In-Cluster_Dist$(L')$;
        $d_C \leftarrow max([d_{C_1}, d_{C_2}, ..., d_{C_K}])$;
        $K \leftarrow K + 1$;
    while $d_C > d_{max}$;
    $L \leftarrow L'$;
end
$return\ L$;
```

A. Motion Plan Feature Extraction

To find promising motion plans, the RA module should measure how close the contacts of the motion plan are to the surfaces in the query environment. The set of contact poses $\mathcal{C}(P)$ is extracted from each motion plan P:

$$\mathcal{C}(P) = \{\langle \mathbf{p}_k, e_k \rangle \,|\mathbf{p_k} = \langle \mathbf{X_k}, \mathbf{Q_k} \rangle \in SE(3); k = 1, 2, ..., N\} \tag{3}$$

where N is the number of contact poses, e_k is an index which indicates the corresponding end-effector, and $\mathbf{X_k}$ and $\mathbf{Q_k}$ are the translation vector and the rotation quaternion, respectively. As suggested in [18], the distance between two contact poses is:

$$\delta(\langle \mathbf{p}_i, e_i \rangle, \langle \mathbf{p_j}, e_j \rangle) = \begin{cases} |\mathbf{X_i} - \mathbf{X_j}| + w_r \cdot (1 - |\mathbf{Q_i} \cdot \mathbf{Q_j}|), w_r > 0 & , e_i = e_j \\ \infty & , e_i \neq e_j \end{cases} \tag{4}$$

To calculate the distance between motion plans, the motion plans need to be aligned. We define the start and goal point of the motion plan as the mean position of the feet in the first and last configurations of the trajectory. We then align the start points of the two plans and subsequently rotate the $\mathcal{C}(P)$ of one plan about the global Z axis to align the start and the goal points of both motion plans on the same line.

The distance between a pair of motion plans P_1 and P_2 is defined as the Hausdorff distance of the between their sets of contact poses:

$$\begin{aligned} d(P_1, P_2) = max\{ & \underset{\langle \mathbf{p}_i, e_i \rangle}{sup} \underset{\langle \mathbf{p_j}, e_j \rangle}{inf} \delta(\langle \mathbf{p}_i, e_i \rangle, \langle \mathbf{p_j}, e_j \rangle), \\ & \underset{\langle \mathbf{p_j}, e_j \rangle}{sup} \underset{\langle \mathbf{p}_i, e_i \rangle}{inf} \delta(\langle \mathbf{p}_i, e_i \rangle, \langle \mathbf{p_j}, e_j \rangle)\} \end{aligned} \tag{5}$$

where $\langle \mathbf{p}_i, e_i \rangle \in \mathcal{C}(P_1)$ and $\langle \mathbf{p}_j, e_j \rangle \in \mathcal{C}(P_2)$. Hausdorff distance allows comparisons between motion plans with different numbers of contacts, automatically separating motion plans with different lengths. The drawback of Hausdorff distance is its sensitivity to outlying data. However, it is not an issue in our context because the contact poses are bounded by the reachability and balance constraints.

B. K-Means Clustering

The motion plans in the library are clustered with the K-means algorithm using the Hausdorff distance described in Eq. 5. The K is determined by running K-means with iteratively increasing K until the maximum distance between every motion plan pair in each cluster is below a user-defined bound d_{max}. Lowering the bound can increase the similarity in each cluster, but it also increases the number of clusters and lengthens the time to evaluate all clusters.

In each cluster the plan with the minimum sum-of-squared distance to every other motion plan is selected as the cluster

Algorithm 2: Query part of the RA Module

```
$L_{sorted} \leftarrow$ Env_Match_and_Sort$(L)$ ;
for $i$ in 1 to $K$ do
    $C_i \leftarrow L_{sorted}[i]$;
    $C_{i,sorted} \leftarrow$ Env_Match_and_Sort$(C_i)$;
    for $j$ in 1 to $|C_{i,sorted}|$ do
        $P_j \leftarrow C_{i,sorted}[j]$;
        $P_{optimized} \leftarrow$ CES$(P_j)$;
        if $P_{optimized}$ is feasible then
            return $P_{optimized}$;
        end
    end
end
return Failure
```

representative. The cluster representative is used to estimate how well the plans in this cluster fit the query environment.

VII. Query Part of the RA Module

In the query part, the input is a combination of a goal and a query environment modeled as a set of contact regions. The objective is to generate a feasible motion plan as soon as possible. The RA module first calculates the distance of each cluster representative to the query environment. The clusters are then sorted by the distance, and searched in that order. The plans in the searched cluster are also sorted in the same manner, and then deformed by the CES algorithm [16] to adapt to the query environment one-by-one until a feasible motion plan is found (see Algorithm 2).

A. Contact Region Extraction

The environment can be viewed as a union of possible contact poses for each end-effector. In this representation, the distance between any end-effector pose to the environment is simply the distance between the end-effector pose and the nearest contact pose in the environment. Therefore, it is important to convert the environment into the contact pose union representation in order to define the distance between a motion plan and an environment.

We adopt the idea in [16] to express the environment as a union of circular contact regions. We sample multiple circle origins with a pre-defined density, and sequentially grow circular regions from samples not covered by other regions. The circular regions aim to cover all possible contact poses of the environment. If the radius of a region is smaller than the density, the process will iteratively increase the sampling density in its neighborhood until reaching a density bound $d_{cr,min}$. We denote the set of contact regions as CR. Since the contact region does not consider rotation about the contact normal, this representation is a conservative estimation of the available contact poses in the environment, as shown in Figure 4, but it can be generated automatically. Note that end-effector size is considered in the generation of contact regions. The start and goal regions are modeled as circles centered at the specified start/goal position and bounded by the nearest obstacles or surface boundaries.

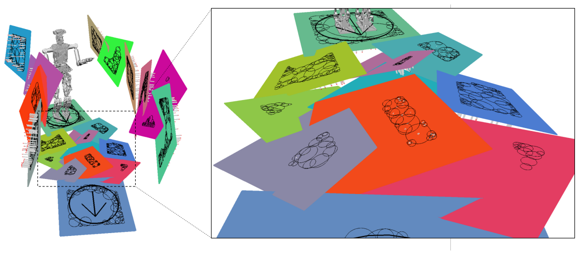


Fig. 4. Contact region sampling

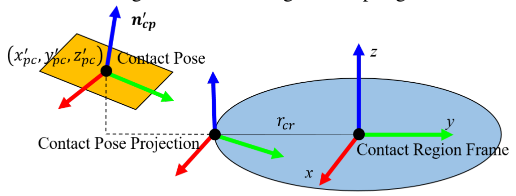


Fig. 5. Contact pose vs. contact region distance.

B. Environment to Motion Plan Cluster Matching

In order to find a promising plan to navigate through the query environment, we define a distance between a motion plan and an environment. Based on the assumption that a motion plan is more likely to be modified to become feasible in the environment if its contact poses are closer to the contact regions in the environment, we match a motion plan to an environment based on the distance between contact poses in the motion plan and the contact regions in the environment. This is done in the *Env_Match_and_Sort* function in Algorithm 2 as follows:

We define a contact region's frame by aligning the Z axis to the contact region normal, and X axis to an arbitrary vector on the surface. The distance between a pose $\mathbf{p}$ and a contact region $cr \in CR$ is defined as:

$$\begin{aligned} &\gamma(\mathbf{p}, cr) = \sqrt{d_{xy}^2 + d_z^2} + w_r d_{ori} \\ &d_{xy} = max\left(0, \left|\left(x'_p, y'_p\right)\right| - r_{cr}\right),\ d_z = \left|z'_p\right| \\ &d_{ori} = 1 - \mathbf{n}'_p \cdot [0,0,1]^T \end{aligned} \tag{6}$$

where $\left(x'_p, y'_p, z'_p\right)$ and $\mathbf{n}'_\mathbf{p}$ are the contact position and normal in the contact region frame, $w_r \in \mathbb{R}^+$ is a weighting factor, and r_{cr} is the radius of the contact region. Furthermore, we can define the projection of the contact pose to the contact region by shifting the contact pose to the closest point inside the contact region, and rotate the pose to align the contact pose normal to the contact region normal, as shown in Figure 5.

Since a motion plan starts and stops inside circular regions, there exist an infinite number of alignments of the plan before deformation of the contact poses. If we treat the whole contact series of a motion plan as a rigid body with 4 degrees of freedom: translation in the X, Y and Z directions, and rotation about the Z axis, expressed as $(x_{rp}, y_{rp}, z_{rp}, \theta_{rp})$, the initialization problem is then to find a transform of the entire plan that minimizes the distance between the plan and the environment. We call this representation of a plan as a rigid body a *rigid plan*. Finding a globally-optimal alignment of the rigid plan is costly so we find a local solution using a Jacobian-based approach. This approach "snaps" the rigid plan to the nearest set of contact surfaces.

Given a query environment, the start region is at (x_s, y_s, z_s) with radius r_s and the goal region is at (x_g, y_g, z_g) with radius r_g. The distance between the start and goal poses is l_{sg}, and the distance between the first and last poses of the rigid plan is l_{rp}. The algorithm initializes the rigid plan pose $\mathbf{T}_{rp} = (x_{0,rp}, y_{0,rp}, z_{0,rp}, \theta_{0,rp})$ as:

$$\begin{aligned} x_{0,rp} &= x_s + (l_{sg} - l_{rp}) \frac{r_s}{r_s + r_g} \frac{|x_g - x_s|}{l_{sg}} \\ y_{0,rp} &= y_s + (l_{sg} - l_{rp}) \frac{r_s}{r_s + r_g} \frac{|y_g - y_s|}{l_{sg}} \\ z_{0,rp} &= (z_s + z_g)/2 \\ \theta_{0,rp} &= atan2(y_g - y_s, x_g - x_s) \end{aligned} \tag{7}$$

This initialization guarantees the rigid plan's first and last poses will be inside the start and goal regions, respectively, if $l_{sg} - r_s - r_g \leq l_{rp} \leq l_{sg} + r_s + r_g$.

After initialization, we iteratively update $\mathbf{T}_{rp}$ to move the rigid plan's $\mathcal{C}(P)$ closer to their nearest contact regions. At each iteration, we find $cr_{min,i}$, the closest contact region to $\langle \mathbf{p}_i, e_i \rangle \in \mathcal{C}(P)$. To ensure that the motion plan connects the start and the goal, the foot poses of the start and the goal configurations are matched to the start and the goal regions, respectively. Jacobian J_i relates $\dot{\mathbf{T}}_{rp}$, the change in the rigid plan pose, to $\dot{\mathbf{p}}_i$, the desired change in the pose of contact $\mathbf{p}_i$. We can then combine the Jacobians for all $\mathbf{p}_i$:

$$\begin{bmatrix} \dot{\mathbf{p}_1} \\ \dot{\mathbf{p}_2} \\ \vdots \\ \dot{\mathbf{p}_N} \end{bmatrix} = \begin{bmatrix} J_1 \\ J_2 \\ \vdots \\ J_N \end{bmatrix} \dot{\mathbf{T}}_{rp} = J \dot{\mathbf{T}}_{rp} \tag{8}$$

We then use the pseudo-inverse J^+ to arrive at a $\dot{\mathbf{T}}_{rp}$ that takes into account the desired motion of all $\mathbf{p}_i$:

$$\dot{\mathbf{T}}_{rp} = J^+ \left[\dot{\mathbf{p}_1}^T, \quad \dot{\mathbf{p}_2}^T, \quad \ldots, \quad \dot{\mathbf{p}_N}^T \right]^T \tag{9}$$

The rigid plan pose will converge to a local minimum $\mathbf{T}'_{rp}$ through iterative application of Eq. 9. The distance between a rigid plan and the query environment is then defined as:

$$\Gamma(\mathcal{C}(P), CR) = \frac{1}{N} \sum_{i=1}^{N} \gamma \left(\mathbf{p}'_i, cr_{min,i} \right) \tag{10}$$

where $\mathbf{p}'_i$ is the ith contact pose in $\mathcal{C}(P)$ transformed by $\mathbf{T}'_{rp}$. The clusters are sorted by this distance, and searched in this order. Motion plans inside the searched cluster are also sorted in the same manner.

C. Local Trajectory Optimization

The motion plan, expressed as a sequence of configurations, is modified and optimized to fit the query environment with CES [16]. Each configuration of the trajectory will move its contact toward the nearest contact region, remain balanced, and avoid obstacles simultaneously. Although each contact pose converges to the nearest contact region according to the contact constraint, the contact pose is also affected by balance constraints and collision avoidance during each iteration. Therefore, contact poses may not end in the initial nearest contact region. In our setup the task priority of CES was (1) obey joint limits; (2) three tasks in parallel: maintain contact, remain in balance, avoid collision; and (3) "internal forces" used to smooth the trajectory, as described in [19].

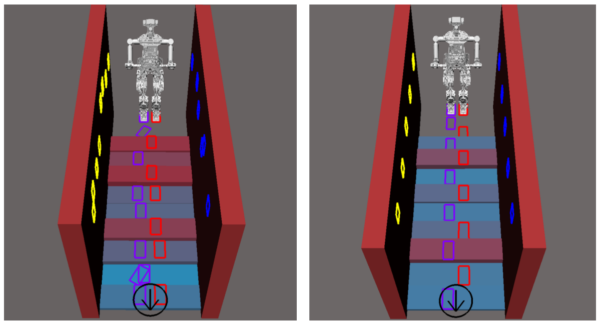


Fig. 6. Examples of staircase test environments

VIII. Experiments

We test on the ESCHER humanoid robot. ESCHER had 33 degrees of freedom (DOF) in our setup: two 7-DOF arms, two 6-DOF legs, one waist joint, and a 6-DOF base transform. The robot is to be in contact with at least 3 of its manipulators at any given time. We implemented our algorithms and test examples in OpenRAVE [20] and also tested in the Gazebo physics simulator[21]. All experiments were run on an Intel Core i7-4790K 4.40 GHz CPU with 16GB RAM. The values of the parameters used in the experiments are the following: $w_\theta = 0.3\text{m/rad}, w_s = 10\text{m}, d_{cr,min} = 0.01\text{m}, w_r = 0.5, d_{min} = 0.1, d_{max} = 0.5$. Time limit for each trial is 5 minutes.

A. Feasibility Database Test

To test the feasibility database, we set up a staircase with wall environment with each stair's height being a random number uniformly distributed in $[0, 0.3]\,\text{m}$, as shown in Figure 6, and plan motions with the PFS module alone. The planner starts with an empty feasibility database, and collects data from the randomly-generated environments in each trial. We define the recall rate of the database as the number of queries answered by the database over the number of total queries. We run 1000 trials, and average results in every 50 trials to get the recall rate and planning time over the number of trials, as shown in Figure 8. The recall rate of the database grows higher as the number of trials increases, and converges to 95% within 500 trials. Furthermore, the planning time drops quickly because fewer direct queries of the IK solver are needed. After 1000 trials, we evaluate the accuracy of the feasibility database by running another 500 trials and comparing its result to direct IK solver queries. The accuracy for the reachability database and the balance database are 99.5% and 99.9%, respectively.

B. Navigating a Narrow Corridor

To show the versatility of the planner, we tested the robot's ability to navigate in a narrow corridor. Using our algorithm with the same parameters as above, the robot is able to turn its body and walk laterally with palms on opposite walls (to balance against disturbances) in a narrow corridor, as

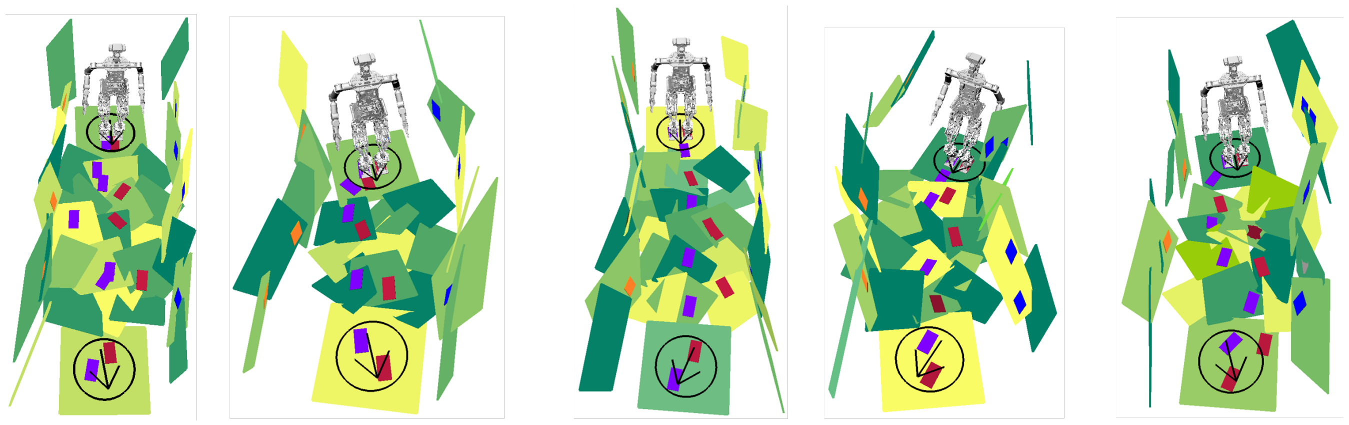


Fig. 7. Examples of plans in rubble-like environments. Planned contacts for left foot (red), purple (right foot), blue (left palm), and orange (right palm).

shown in Figure 9. This plan took 1.55s to compute using the feasibility database.

C. Random Surface Environment Test

In this experiment, we set up complex random surface test environments. We generate the environments with randomly tilted quadrilateral surfaces, as shown in Figure 7. Each environment is between 2m and 4m long. The surfaces may cover or intersect with each other, and leave part of the surface non-contactable, which causes the environment to be very challenging. Since this environment is extremely complex, the recall rate of the reachability and balance databases in these test environments are 10.3% and 30.2%, respectively with 1.5 million entries in the database. This does not provide a significant improvement in planning time, further motivating the need for the RA module.

To evaluate the proposed framework, we generated 100 random surface environments, and record the performance of PFS alone (the baseline) vs. the proposed framework with different sizes of motion plan libraries. If a trial's runtime exceeds 5 minutes, it is counted as a failure. For the PFS module, the failure cases also include optimization failure: the case when the local optimization after contact space planning cannot find a feasible trajectory. For the RA module, the case when no feasible motion plan can be found in the library is counted as a failure. Examples of the test environments and plans generated by our framework are shown in Figure 7.

Figure 11 shows the success rate of the proposed framework with different library sizes. Even with a small library size, the proposed framework significantly improves the success rate. One of the major reason is that CES used in the RA module can shift contact poses in continuous space to arrive at small contact regions. However, PFS may not find feasible next contacts in the transition model, and needs redundant contacts in order to adjust the standing foot to find feasible contacts at the next transition. Those redundant contacts increase the search depth and the planning time.

Furthermore, to navigate in such a complex environment, PFS requires a large transition model that densely discretizes the reachable space of the end-effectors. This entails a higher

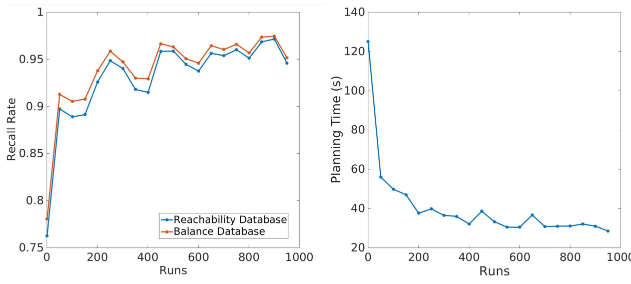


Fig. 8. Left: recall rate, Right: planning time of the feasibility database

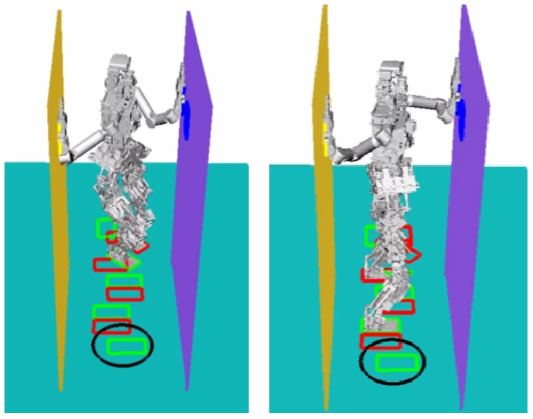


Fig. 9. Snapshots from the plan to navigate through a narrow corridor.

branching factor. When the heuristic is not accurate, this high branching factor slows down the planner.

Figure 11 shows the average planning time of the **successful trials**. Although the RA module takes more time to find a solution with a larger library, the increase in planning time of the successful trials is partly because the RA module with a large library can find solutions in difficult cases which cannot be solved within the time limit using a small library. For a library with 20 entries, RA outperforms the PFS success rate by 10%, and the combined framework outperforms PFS by 28%. For a library with 200 entries, RA outperforms the PFS success rate by 44%, and the combined framework outperforms PFS by 49%.

In Figure 12, we can observe that the number of trials in which the RA module finishes first increases as the size of the motion plan library grows. However, the trend saturates after the size exceeds 100. This is a mixture of two effects:

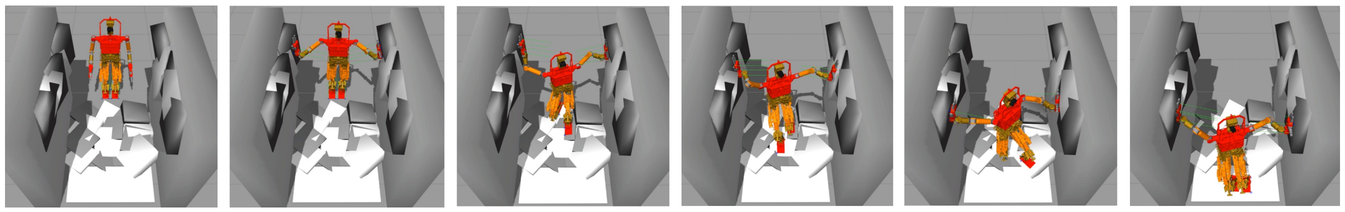
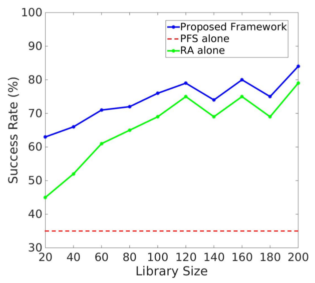


Fig. 10. Gazebo simulation of robot navigating through a rubble-like environment.



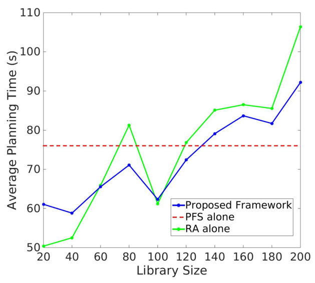


Fig. 11. Left: Success rate and Right: Average planning time of **successful trials** for the PFS module, the RA module and the proposed framework with different sizes of libraries

(1) The RA module can solve more problems with a larger library. (2) The RA module requires more time to find a feasible motion plan in a larger library. This effect can be observed in Figure 12 as the successful and timeout cases both increase with a larger library.

D. Testing in Physics Simulation

To verify the feasibility of trajectories produced by the framework, we executed the plans in a random surface environment in the Gazebo simulator. The robot can walk through the "rubble" while using palm contacts for balance, as shown in Figure 10 and the attached video.

IX. Conclusion

In this paper, we proposed a humanoid navigation planning framework running two modules in parallel: Planning from Scratch (PFS) and Retrieve and Adapt (RA). PFS is a discrete-search-based contact space planner. RA stores and clusters previously generated motion plans based on the Hausdorff distance of the contact poses. When the robot encounters a new environment, the module matches each cluster representative to the environment, and searches motion plan clusters based on the distance between the motion plan cluster representative and the environment. Each plan in the searched cluster is then sorted by distance to the environment and then modified by CES algorithm to fit the environment until a valid plan is found. The results show that the proposed framework outperforms the baseline planning-from-scratch algorithm in success rate in difficult unstructured environments. The design of the RA module also opens the possibility of using parallel computing to further speed up the framework, which we would like to address in future work.

Acknowledgments

This work was supported in part by the Office of Naval Research grant N00014-15-1-2138. Special thanks to TREC Lab, Virginia Tech, for technical support, and colleagues in ARC Lab for feedback and discussion.

	Plan found	Opt. Fail	Time out	Lib. Exhaust
PFS	35	21	44	-
RA(20)	45	-	0	55
RA(40)	52	-	1	47
RA(60)	61	-	2	37
RA(80)	65	-	6	29
RA(100)	69	-	6	25
RA(120)	75	-	4	21
RA(140)	69	-	10	21
RA(160)	75	-	7	18
RA(180)	69	-	11	20
RA(200)	79	-	10	11

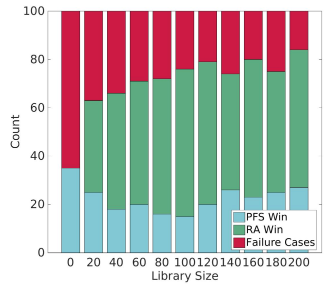


Fig. 12. Left: Results with different library sizes; Right: Number of trials in which PFS or RA finishes first for different library sizes.

References

[1] J. van den Berg, R. Shah, A. Huang, and K. Goldberg, "Anytime nonparametric A*," in *AAAI*, 2011.

[2] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Footstep planning among obstacles for biped robots," in *IROS*, 2001.

[3] J. Chestnutt, J. Kuffner, K. Nishiwaki, and S. Kagami, "Planning biped navigation strategies in complex environments," in *Humanoids*, 2003.

[4] P. Michel, J. Chestnutt, J. Kuffner, and T. Kanade, "Vision-guided humanoid footstep planning for dynamic environments," in *Humanoids*, 2005.

[5] L. Baudouin, N. Perrin, T. Moulard, F. Lamiraux, O. Stasse, and E. Yoshida, "Real-time replanning using 3d environment for humanoid robot," in humanoids," in *Humanoids*, 2011.

[6] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz, "Anytime search-based footstep planning with suboptimality bounds," in *Humanoids*, 2012.

[7] O. Kanoun, E. Yoshida, and J. P. Laumond, "An optimization formulation for footsteps planning," in *Humanoids*, 2009.

[8] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *Humanoids*, 2014.

[9] M. Lau and J. Kuffner, "Precomputed search trees: planning for interactive goal-driven animation," in *SCA*, 2006.

[10] H. Myung, J. Kuffner, and T. Kanade, "Efficient two-phase 3d motion planning for small fixed-wing uavs," in *ICRA*, 2007.

[11] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *ICRA*, 2012.

[12] D. Coleman, I. Sucan, M. Moll, K. Okada, and N. Correll, "Experience-based planning with sparse roadmap spanners," in *ICRA*, 2015.

[13] N. Jetchev and M. Toussaint, "Fast motion planning from experience: trajectory prediction for speeding up movement generation," in *Autonomous Robots*, 2013.

[14] A. Escande, A. Kheddar, S. Miossec, and S. Garsault, "Planning support contact-points for acyclic motions and experiments on hrp-2," in *ISER*, 2008.

[15] K. Hauser, T. Bretl, K. Harada, and J. Latombe, "Using motion primitives in probabilistic sample-based planning for humanoid robots," in *WAFR*, 2006.

[16] S. Chung and O. Khatib, "Contact-consistent elastic strips for multi-contact locomotion planning of humanoid robots," in *ICRA*, 2015.

[17] S. Caron, Q. Pham, and Y. Nakamura, "Leveraging cone double description for multi-contact stability of humanoids with applications to statics and dynamics," in *RSS*, 2015.

[18] J. Kuffner, "Effective sampling and distance metrics for 3d rigid body path planning," in *ICRA*, 2004.

[19] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," in *IJRR*, 2002.

[20] R. Diankov, *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, 2010.

[21] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IROS*, 2004.