Humanoid Navigation in Uneven Terrain using Learned Estimates of Traversability

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

***Abstract*— In this paper we explore discrete search-based contact space planning for humanoids using both palm and foot contact in complex unstructured environments. With a high branching factor and sparse contactable regions, it is challenging for the planner to find a contact sequence in such environments quickly. Therefore, we propose to learn a function which predicts *traversability*—a measure of how quickly the contact space planner can generate contact sequences to traverse a certain region. By including a learned traversability estimate into the heuristic function of the contact space planner, we can bias the planner to search the areas with more contactable regions, and thus find contact sequences more efficiently. In this paper we propose and evaluate two kinds of feature vectors for estimating traversability: Exact Contact Checking (ECC) and Approximate Contact Checking (ACC), which make different trade-offs between speed and accuracy. The experimental results show that the proposed approach using ACC outperforms both ECC and the baseline heuristic for contact space planning; ACC increases the planning success rate by 19% and reduces average planning time by 24% compared to the baseline in difficult environments with uneven terrain.**

I. Introduction

In humanoid robot locomotion planning, the introduction of palm contacts can help the robot locomote more robustly under disturbances as well as allowing navigation through unstructured environments, such as disaster sites. However, adding palm contacts complicates the problem by significantly increasing the branching factor of search-based planners. Furthermore, balance checking with multiple non-coplanar contacts is also expensive to compute. Using discrete search-based planners [13], [19] to compute a sequence of contacts thus becomes inefficient because the high branching factor and expensive state validity check leads to very large computation times. Our approach to overcoming this problem is to create a more informative heuristic for discrete search-based planners that plan in the combined foot and palm contact space.

In unstructured environments with uneven terrain, conventional distance-based heuristics do not capture the environment geometry, and may cause the planner to waste a large amount of time exploring in a cul-de-sac due to a lack of contactable surfaces in a certain region. Therefore, we propose to estimate the *traversability* of a region in the environment as part of the planner's heuristic, as shown in Figure 1. Traversability is a measurement on how easy the contact space planner can generate contact sequences to traverse through a certain region. In order to be effective,

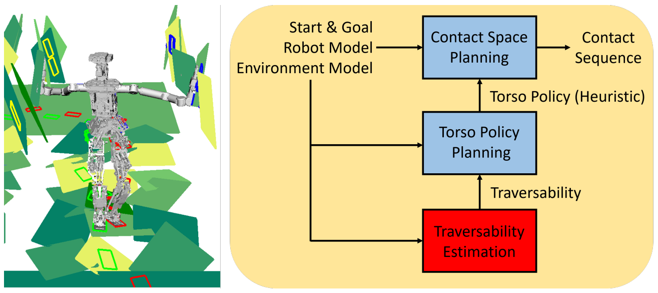

Fig. 1. Left: A humanoid follows a planned sequence of contact poses to navigate in a complex unstructured environment. Right: The structure of the proposed framework. We focus on traversability estimation to guide the robot to avoid difficult regions in the environment.

our estimate of traversability needs to be computed quickly, but this is difficult because computing whether the robot can truly move through a certain region requires expensive geometric tests, collision checks, and balance checks. Instead we propose a learning framework to estimate the traversability of a region in the environment.

To guide contact space planning, we first generate a torso policy to inform the contact space planner's heuristic to avoid obstacles and move toward the goal. The torso policy is computed by treating the robot as a mobile robot traveling on a map represented as a grid, and contains a path from each cell of the grid to the goal. Our key contribution is that we extend the standard guiding policy computation [10] to also consider the traversability of each transition of the torso with an estimate learned directly from the trials generated by the planner, thus taking into account contacts in the environment. For a given torso transition on the grid, we measure the traversability as the number of useful footstep combinations corresponding to the transition.

Computing this value however is computationally expensive, so we use learning to predict traversability. We propose two kinds of features to estimate traversability: Exact Contact Checking (ECC) and Approximate Contact Checking (ACC). For a given torso pose and its surrounding environment, ECC checks collision for each possible foot or hand transition target. ACC, on the other hand, approximates the collision check by using a scoring function to measure the distance of the potential new contact position to the closest obstacle. Since ACC replaces collision check with a simple table lookup, it is much more efficient than ECC. For both ECC and ACC, we evaluate learning regressors with Support Vector Regression (SVR) and Nearest Neighbor (NN).

Our results suggest that using ACC to generate the estimate of traversability as part of the heuristic improves both the success rate and planning time of navigating

[1]Yu-Chi Lin and Dmitry Berenson are with the University of Michigan, Ann Arbor, MI, U.S.A. `linyuchi@umich.edu`, `berenson@eecs.umich.edu`

through difficult environments as compared with the baseline heuristic and with the heuristic using ECC. Furthermore, SVR outperforms NN for both ECC and ACC in terms of planning time and success rate, which shows that SVR helps generalize from the training data.

Preliminary work on this topic was presented as a workshop paper [14] and focused on the implementation of ECC. In this paper we introduce a new feature vector, ACC, which greatly improves the performance of the contact space planner and perform extensive new comparisons.

II. Related Work

Humanoid footstep planning has been studied extensively: [1], [3], [10], [12], [15] are discrete-search-based approaches, which use A*-like algorithms. 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], [11]. Among these works, [3] is the most related to our work. [3] described several ground geometry features and used A* search or best-first search to guide the robot away from obstacles, steep slopes and uneven terrain. However, in their environment, there exists some part of the environment which is flat or piecewise flat, so that the robot can effectively avoid difficult regions. In our work, we seek to identify regions that are easier to traverse even if the region is as geometrically cluttered as other regions.

There is also recent work addressing humanoid navigation in unstructured environments using multiple contacts. [9] used optimization to find contacts in the neighborhood of a “rough” trajectory. However, the planning time was prohibitively long. [5] combined discrete-search-based contact space planning with a local trajectory optimizer to quickly compute a whole-body trajectory using multiple contacts. [18] used reachability volumes as a hint to plan for a robot guiding path first, and then planned for contact placement along this path, which significantly sped up the planning process. We share the idea of using a guiding path to reduce the search space. However, in the test cases of these works, the contactable region is either rich or is identified before planning, so the planner is less likely to be stuck in a cul-de-sac. In this paper, we are developing methods that work with limited information about where contactable regions exist and aim to avoid wasting time searching a region with sparse contactable surfaces.

There has been work proposing traversability estimation algorithms for mobile robots, [6], [16], [17]. These methods learn models to estimate the terrain types based on visual, range or thermal inertia sensor data. The goal is to avoid certain types of terrain which may cause the mobile robot to slip or be stuck. In our work, the traversability does not measure the effect of the texture of the terrains for navigation, instead, it measures the richness of the space for humanoid robot contact placement.

Researchers have investigated predicting traversability for quadruped robots [4], [21]. They computed traversability features such as slope, terrain roughness and step height from visual data. Those features are combined in a weighted-sum cost function, which guides the robot. In our approach, we not only capture features from the environment, but also use simulation to learn a model to predict the actual traversability of the robot in the 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 path from the start configuration to a goal region in the workspace. The path is defined as a series of foot and palm placements. When executing this path, the robot must obey balance and collision constraints at all times. We are interested in using the hands to help balance the robot against potential disturbances and to assist in traversing uneven terrain. Therefore, on uneven terrain, a feasible path should have at least three end-effectors in contact at any time. We assume that the robot can generate sufficient torque to balance itself. We also assume the friction coefficients are given. Our goal is to compute a feasible path for the robot as quickly as possible.

IV. Proposed Approach

To navigate through a complex environment, it is common to plan the contact sequence of the humanoid robot using a discrete search-based approach. In our work we use the ANA* algorithm [19] because of its anytime planning properties and lack of tuning parameters. However, any discrete search-based method that uses heuristics can benefit from the method we propose here. In the proposed planning framework, the contact space planner is guided by a torso policy. The torso policy is generated by treating the robot as a mobile robot and planing a shortest path from each grid cell in a grid over the environment to the goal while avoiding infeasible regions, such as walls or holes. The contact space planner will query the torso policy as part of its heuristic to evaluate which nodes are most promising to expand. Our key contribution is that we compute the torso policy while accounting not only for the length of a transition (as in previous work [10]), but also the traversability of each region in the environment. In the following sections, we first explain the discrete search-based planner used in this paper and then introduce a learning approach to estimate traversability, which is combined with a standard heuristic to speed up contact space planning.

A. Contact Space Planner

We formulate contact space planning as a graph search problem. We define each state of the planner as the set of poses of all end-effectors which are in contact with the environment. The actions connecting these states are either shifting one contact to a new contacting pose or breaking one contact. For each state, the set of actions—called the *transition model*—is predefined. For foot contact transitions,

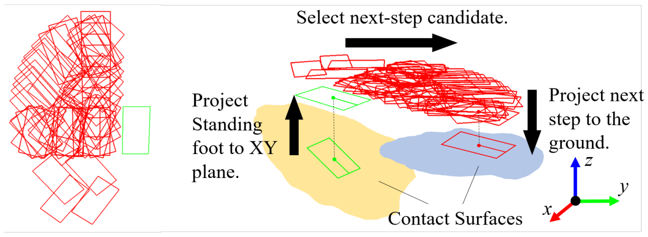
Select next-step candidate.
Project Standing foot to XY plane.
Project next step to the ground.
Contact Surfaces
z
y
x

Fig. 2. Left: Foot contact transition model (57 steps) Right: The projections to get the next step pose.

we follow the projection procedure shown in Figure 2. A dense foot transition model is used because we expect the robot to navigate in a complex environment. We denote the set of all foot contact transitions as $\mathbf{FC}_{\boldsymbol{\Delta}}$.

For palm contacts, we first approximate the torso pose p_t based on the poses of the feet with the following equations:

$$p_t = \begin{bmatrix} \frac{x_{lf}+x_{rf}}{2} & \frac{y_{lf}+y_{rf}}{2} & \frac{z_{lf}+z_{rf}}{2}+z_{\text{offset}} & 0 & 0 & \frac{\theta_{lf}+\theta_{rf}}{2} \end{bmatrix}^T \tag{1}$$

where $[x_{lf}, y_{lf}, z_{lf}]$ and $[x_{rf}, y_{rf}, z_{rf}]$ are the left and right foot positions, respectively, θ_{lf} and θ_{rf} are the rotations of each foot about the z axis, and z_{offset} represents the nominal body height relative to the feet.

Given the approximated torso pose in each state, we can derive the approximated shoulder points of the robot. The potential palm contacts are then computed by ray-casting from the approximated shoulder points, as shown in Figure 3. We denote the set of all palm contact transitions as $\mathbf{PC}_{\boldsymbol{\Delta}}$.

Besides a transition model, search-based planners require a function to estimate the cost of performing each action. The cost of a foot transition action is defined as:

$$\Delta g_f = d_t + w_s \tag{2}$$

where d_t is the distance the approximated torso travels in this action, and w_s is a fixed cost of taking a step. For the palm action, we define the cost as:

$$\Delta g_p = d_p + w_s \tag{3}$$

where d_p is the distance the palm travels in this action. Each state is feasible if there exists a collision-free and statically-balanced inverse kinematics solution based on the specified end-effector poses. We use the method described in [2] as the balance checker which is treated as a constraint in the inverse kinematics solver. To speed up the process, we approximate the balance check for the entire transition by checking two critical configurations: the beginning of the contact transition where the moving end-effector has just broken contact and the end of the contact transition where the moving end-effector is about to make contact.

As formulated above, the search problem has a very high branching factor, 80 to 120 depending on the state, and thus the search would be very time-consuming without biasing the exploration. We thus need to use a heuristic function to allow the planner to explore transitions in a goal-biased way. To compute the heuristic for each state, we first plan with

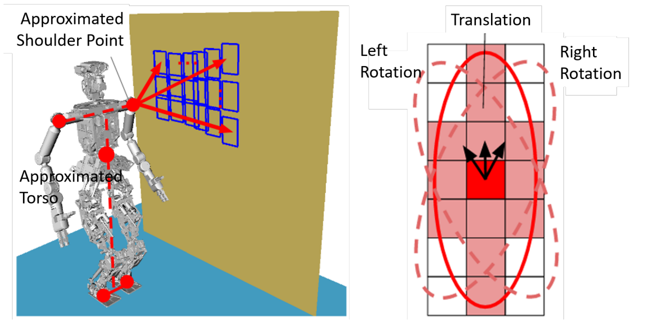

Fig. 3. Left: Palm contact transition model Right: Torso pose transition model in the torso pose grid. Note that we only show the translation for one torso orientation here. To generate translation for other torso orientation, we rotate the ellipse, which represents the moving range of the torso, to align with the torso orientation and count all cells inside the ellipse as possible translations for the torso orientation.

a more coarse robot model which only considers the robot torso. This coarse planning produces a policy which is used as a heuristic for subsequent contact space planning. With this heuristic, the robot can avoid exploring paths that are dead ends, contain holes in the ground, or are blocked by large obstacles, such as pillars and walls.

B. Torso Pose Policy

To compute the robot torso policy, we discretize the robot torso pose in x and y, and the rotation about the z axis, θ, and call the resulting grid the *torso pose grid*. In this paper, we assume that the robot is traveling on a surface, so z is uniquely defined by the x and y coordinates. Thus we do not include translation in z in the grid. The grid cells in which there is no contactable surface or the torso collides with the environment will be marked as invalid by the torso planner. The possible transitions of the robot torso for one step are shown in Figure 3. The ellipse shape captures the fact that the robot can travel farther with a forward or backward step than a lateral step.

Since the robot torso policy is used as the heuristic for the contact space planner, the torso planning starts from the grid cell where the goal is and expands to every other cell in the torso pose grid using Dijkstra's algorithm. This algorithm outputs a policy (i.e. a direction to move for each cell) in the form of a tree. Besides outputting the policy, we also record the cost of each edge in the torso policy Δg_{tp}, and the cost of each cell g_{tp} for use in the contact space planner's heuristic function:

$$\begin{aligned} \Delta g_{tp}\left(p_{t,\text{parent}}, p_t\right) &= l_{p_t}^{p_{t,\text{parent}}} + w_s \\ g_{tp}\left(p_t\right) &= \sum \Delta g_{tp} = l_{goal}^{t}(p_t) + w_s N_s \end{aligned} \tag{4}$$

where $p_{t,\text{parent}}$ is the parent cell of the cell containing p_t in the torso policy tree, l_{goal}^t is the length of the path from the cell containing p_t to the goal cell, and N_s is the number of steps taken along that path.

C. Heuristic for Contact Space Planning

The torso policy above will be queried as part of the heuristic for contact space planning. However, since the torso policy does not consider palm contact, we need to add a component to estimate the cost of palm contact transitions along the path to the goal. We define the palm component of the contact space planner's heuristic as:

$$h_p\left(p_t\right) = l_{lp}(P_{p_t}) + l_{rp}(P_{p_t}) + w_s\left(\frac{l_{lp}(P_{p_t})}{d_{lp,max}} + \frac{l_{rp}(P_{p_t})}{d_{rp,max}}\right) \tag{5}$$

where P_{p_t} is the path from the cell containing p_t to the goal in the torso policy, l_{lp} is the length of the portion of P_{p_t} where it is possible to make left and palm contact with the environment, and likewise l_{rp} for right palm contact. $d_{lp,max}$ and $d_{rp,max}$ are the maximum distances each palm contact can travel in one action.

To evaluate the heuristic for each state in contact space planning we find the grid cell containing p_t, which is estimated from the contact state using Eq. 1. We then combine that cell's cost g_{tp} from the torso policy with the palm component $h_p(p_t)$ to arrive at the heuristic for the contact space planner:

$$h\left(p_t\right) = g_{tp}\left(p_t\right) + h_p\left(p_t\right) \tag{6}$$

This heuristic is similar to the Dijkstra heuristic described in [10] with the modification of adding h_p.

D. Introducing Traversability into the Torso Policy

The heuristic function described above does not capture the difficulty of navigating through uneven terrain. The robot will try to follow the shortest path in the torso pose grid, regardless of the difficulty of finding contact poses for the feet and palms. For example, the ground may be cluttered and may not have enough space for the robot to stand. If such areas are in the neighborhood of the shortest path given by the torso policy above, it forms a cul-de-sac and the planner will spend significant time exploring this area without making progress to the goal.

To better reflect the difficulty of moving through each part of the environment, we add a *traversability estimate* Δg_{tr} to the torso path cost defined in Eq. 4, and we denote this new torso path cost g_{combined}.

$$\begin{aligned}\Delta g_{\text{combined}}\left(p_{t,\text{parent}}, p_t\right) &= l_{p_t}^{p_{t,\text{parent}}} + w_s + \Delta g_{tr}\\ g_{\text{combined}}\left(p_t\right) &= \sum \Delta g_{\text{combined}} = l_{goal}^{t}(p_t) + w_s N_s + g_{tr}\end{aligned} \tag{7}$$

where g_{tr} is the sum of the traversability estimate along the torso path. Torso policies generated with the cost defined in Eq. 7 will avoid areas with low traversability, which will subsequently guide the contact space planner to avoid these areas when possible.

It is important to note that traversability depends on the direction we are moving in, so it must be evaluated for every edge in the graph. To compute the true traversability would require running a contact space planner to move between every pair of adjacent cells, which is clearly too time-consuming. Instead we propose two kinds of features to estimate traversability using machine learning methods:

Algorithm 1: Compute Ground Truth Label in ECC

```
1 Input : $p_t, \phi, \mathbf{FC_{\Delta}}, \mathbf{E}, \lambda, \hat{t}$ ;
2 $\Gamma \leftarrow$ GetFeasibleFootstepCombinations $(p_t, \mathbf{FC_{\Delta}}, \mathbf{E})$;
3 $\Gamma_+ \leftarrow \{\ \}$;
4 for $\gamma$ in $\Gamma$ do
5     if ContactSequenceExists $\left(\gamma, \phi, \mathbf{E}, \lambda, \hat{t}\right)$ then
6         $\Gamma_+ \leftarrow \{\gamma\} \cup \Gamma_+$;
7 return $|\Gamma_+|$;
```

The first is Exact Contact Checking (ECC), which evaluates the traversability by finding exact contact poses projected to the environment, and checking if they are collision-free. The other feature is Approximate Contact Checking (ACC), which approximates collision checking and foot projection with a scoring function. We describe ECC and ACC in the following sections, and compare them in Section V.

E. Learning Traversability with Exact Contact Checking (ECC)

The traversability estimator $\Delta g_{tr} : \{p_t, \phi, \mathbf{E}\} \rightarrow \mathbb{R}_+$ takes as input a torso pose p_t, a direction of motion ϕ, and a local environment model $\mathbf{E}$. The direction of motion ϕ is a discretization of 360 degrees to 12 ranges of direction, so each ϕ covers 30 degrees. Note that the traversability actually depends on not only the direction of motion, but also the distance moved. However, because of the high computation cost of ECC, we only consider the direction of motion ϕ in ECC to reduce the number of queries to the estimator. We will consider distance moved when computing ACC in the next section.

To compute Δg_{tr}, we start by finding a set of feasible footstep combinations Γ at p_t. To compute Γ, we first use the footstep transition model shown in Figure 2 and Eq. 1 to find possible footstep combinations given $p_t = (x_t, y_t, \theta_t)$. For example, if a transition is that the right foot moves to $\left(x_{rf}^{lf}, y_{rf}^{lf}, \theta_{rf}^{lf}\right)$ relative to the left foot, then the left foot and right foot pose can be computed in the world frame as:

$$\begin{aligned}\theta_{lf} = \theta_t - \frac{1}{2}\theta_{rf}^{lf}\ &;\ \theta_{rf} = \theta_t + \frac{1}{2}\theta_{rf}^{lf}\\ \begin{bmatrix} x_{rf} \\ y_{rf} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} \cos\theta_{lf} & -\sin\theta_{lf} \\ \sin\theta_{lf} & \cos\theta_{lf} \end{bmatrix} &\begin{bmatrix} x_{rf}^{lf} \\ y_{rf}^{lf} \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \end{bmatrix}\end{aligned} \tag{8}$$

The calculation is analogous for the left foot moving. These 3D poses of the feet will then be projected to the environment to obtain the full 6D pose. If there exists a valid projection on the environment for both feet, this footstep combination is feasible. The above computation corresponds to GetFeasibleFootstepCombination function in Algorithm 1

Given an environment and a direction of motion, if the contact space planner can generate a contact sequence which applies both palm contacts and moves the robot torso more than a short distance λ starting from a feasible foot combination $\gamma \in \Gamma$ within a planning time limit $\hat{t}$, we call such a γ a *useful* footstep combination. The set of all useful footstep combinations is denoted as $\Gamma_+ \subset \Gamma$, as shown in Lines 3 to 7 in Algorithm 1.

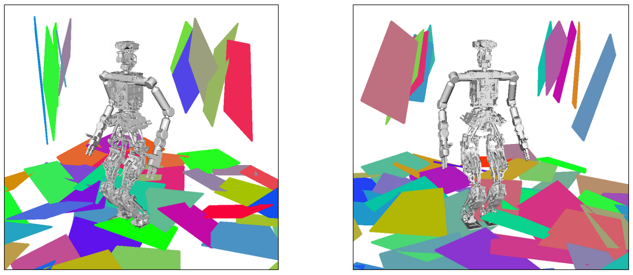
Fig. 4. Two examples of the training environment

The number of useful footstep combinations, denoted $|\Gamma_+|$, can serve as an indicator for how difficult it is for the planner to find a contact sequence moving in a certain direction. Therefore, we would like to learn a regressor which predicts $|\Gamma_+|$ based on features derived from the foot and palm transitions given a torso pose.

The training data consists of features of the local environments in which the robot is expected to operate (described below) with a corresponding $|\Gamma_+|$ produced by evaluating each γ and with the contact-space planner given ϕ. In this paper, we explore the case that the robot navigates in a corridor with tilted surface patches. This is particularly challenging because it is difficult to find collision-free contact and to remain balance on tilted surface patches. To generate training data we randomly generate corridors with different width and orientation relative to the robot as well as different surface dimensions, as shown in Figure 4. In many cases, the ground is too steep for the robot to remain stable with only two foot contacts, so the planner is constrained to keep the palms on the wall in the corridor.

To calculate the feature vector, we perform a transition of a foot contact from each $\gamma \in \Gamma$ for only one step based on the transition model shown in Figure 2. For a given ϕ, we then count the number of $\gamma \in \Gamma$ which have at least one valid footstep transition in the ϕ direction, and denote it as N_{fc}. The calculation of N_{fc} is summarized in Lines 4 to 9 in Algorithm 2. The palm contact direction also affects the planner's ability to move in a certain direction. For example, if there exist only palm contacts in the direction opposite to the direction of motion, the planner may not be able to find a valid contact sequence. Therefore, we add another feature: the number of valid palm contacts with the environment in each quadrant of the torso frame: $N_p[i]$ for the ith quadrant. The process of calculating palm contact features is summarized in Line 10 to 13 in Algorithm 2. Given the training data consisting of the 5D feature vector, $[N_{fc}, N_p[1], N_p[2], N_p[3], N_p[4]]$, and 1D output, $|\Gamma_+|$, we evaluate two learning methods: Support Vector Regression (SVR) with an RBF kernel and Nearest Neighbor. A separate regressor is learned for each ϕ, resulting in 12 regressor functions.

F. Learning Traversability with Approximate Contact Checking (ACC)

Although ECC provides useful information to estimate traversability, it requires many collision checks for each torso pose and direction of motion pair, which is time-consuming

Algorithm 2: Generate the Feature Vector in ECC

```
1  Input : $p_t, \phi, \mathbf{E}, \mathbf{FC_\Delta}, \mathbf{PC_\Delta}$;
2  $\Gamma \leftarrow$ GetFeasibleFootstepCombinations $(p_t, \mathbf{FC_\Delta}, \mathbf{E})$;
3  $N_{fc} \leftarrow 0$;  $\mathbf{N}_p \leftarrow [0,0,0,0]$;
4  for $\gamma$ in $\Gamma$ do
5      for $fc$ in $\mathbf{FC_\Delta}$ do
6          $\gamma_{ex} \leftarrow$ TransitionAndProject $(\gamma, fc)$;
7          if IsCollisionFree $(\gamma_{ex})$ and
             MovesInDirection $(\gamma, \gamma_{ex}, \phi)$ then
8              $N_{fc} \leftarrow N_{fc} + 1$;
9              break;
10 for $pc$ in $\mathbf{PC_\Delta}$ do
11     $p_{palm} \leftarrow$ GetPalmPose $(p_t, pc)$;
12     $i_q \leftarrow$ GetPalmQuadrant $(p_t, p_{palm})$;
13     $\mathbf{N}_p[i_q] \leftarrow \mathbf{N}_p[i_q] + 1$;
14 return $[N_{fc}, \mathbf{N}_p]$;
```

Algorithm 3: Compute Ground Truth Label in ACC

```
1  Input : $p_t, \mathbf{v}, \mathbf{FC_\Delta}, \mathbf{E}, \lambda, \hat{t}$ ;
2  $\Gamma \leftarrow$ GetFeasibleFootstepCombinations $(p_t, \mathbf{FC_\Delta}, \mathbf{E})$;
3  $\Gamma_+ \leftarrow \{\ \}$;
4  for $\gamma$ in $\Gamma$ do
5      if ContactSequenceExists $(\gamma, \mathbf{v}, \mathbf{E}, \lambda, \hat{t})$ then
6          $\Gamma_+ \leftarrow \{\gamma\} \cup \Gamma_+$;
7  return $|\Gamma_+|$;
```

and thus reduces the benefit of learning traversability. Therefore, based on ECC, we propose Approximate Contact Checking (ACC), which approximates the collision checks with a scoring function but is more efficient than ECC.

Since computing ACC is much faster than computing ECC, we can afford more queries to the traversability estimator in reasonable computation time. Therefore, when learning traversability with ACC, we take both direction of motion and distance moved into account. In ACC, we redefine the traversability estimator as: $\Delta g_{tr} : \{p_t, \mathbf{v}, \mathbf{E}\} \rightarrow \mathbb{R}_+$. It takes as input a starting torso pose p_t, and a 2D torso pose translation $\mathbf{v}$ in the XY plane, and a local environment model $\mathbf{E}$. Similar to ϕ used in computing ECC, we have a finite set of $\mathbf{v}$ given a torso pose, as shown in Figure 3. We denote the resulting torso pose of p_t after translation $\mathbf{v}$ as $p_t' = p_t + \begin{bmatrix}\mathbf{v}^T & 0\end{bmatrix}^T$.

To compute Δg_{tr}, we follow Eq. 8 to obtain the set of feasible footstep combinations Γ in p_t. With the new definition of the traversability estimator, we thus redefine the useful footstep combination, Γ_+, used in ACC as: Given an environment, a torso pose translation $\mathbf{v}$ and a starting footstep combination $\gamma \in \Gamma$, if the contact space planner can generate a contact sequence which applies both palm contacts and moves the torso to the cell to which p_t' belongs in the torso pose grid, the set of all such γ is Γ_+. We limit the number of palm contact poses that the planner can explore, denoted $n_{\text{p,lim}}$. Therefore, the planner will return failure only when the search tree is exhausted. The number of useful footstep combinations, $|\Gamma_+|$ serves as an indicator

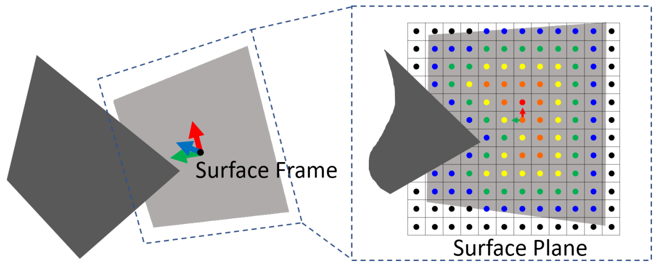

Fig. 5. The grid of contact points on a surface plane. The distance of each contact point to the closest obstacle or surface boundary is marked in color spectrum order. Note that the light gray surface is covered by the dark gray surface, which causes part of its contact points to be infeasible.

for the traversability. The process computing the ground truth is summarized in Algorithm 3. As in ECC, we randomly generate corridor environments shown in Figure 4 to collect training data.

To compute the feature vector with ACC, we first discretize each surface frame into a set of contact points C_i which form a grid. We denote the set of all contact points, which is also the union of all C_is from all surface, as C. For each contact point $c \in C$, we cast a ray from each contact point along the normal of each surface to check if the contact point is collision-free. The distance of each contact point to the closest obstacle, denoted as $\delta(c)$, is approximated as the closest distance to any contact point in collision. Figure 5 shows an example contact point grid of a surface. We can define the following scoring function to represent the clearance of each contact point:

$$S(c) = \begin{cases} 0 & \delta(c) < r_{\text{ins}} \\ \frac{\delta(c) - r_{\text{ins}}}{r_{\text{cir}} - r_{\text{ins}}} & r_{\text{ins}} \leq \delta(c) < r_{\text{cir}} \\ 1 & \delta(c) \geq r_{\text{cir}} \end{cases} \quad (9)$$

r_{ins} and r_{cir} are the radius of the inscribed and circumscribed circle of the contact end-effector shape, respectively. For each contact, if $\delta(c)$ is larger than r_{cir}, there must exist enough free space for any contact pose at the contact point c. However, if $\delta(c)$ is lower than r_{ins}, it is impossible to make contact at c regardless of the orientation of the contact.

S describes how likely a contact pose is feasible given its corresponding contact point c. In other words, each collision check is turned into a table lookup, which speeds up the process. For foot contacts, based on the footstep transition projection shown in Figure 2, we can further project all the contact points on ground surfaces to a 2D grid on XY plane so that S can be queried with only the X and Y coordinate of the foot contact.

For each feasible foot combination $\gamma \in \Gamma$, we expand the contact space planning search tree based on the transition model shown in Figure 2 for only one step. For each expansion, we can define a footstep translation tuple α as $\alpha = \left\{\mathbf{t}^t_{lf}, \mathbf{t}^t_{rf}, \mathbf{t}^t_{ex}\right\}$. $\mathbf{t}^t_{lf}$ and $\mathbf{t}^t_{rf}$ are the 2D translation of the left foot and right foot in the torso frame in the XY plane, and $\mathbf{t}^t_{ex}$ is the 2D translation of the expanded footstep in the torso frame in the XY plane. Following the definition of torso pose p_t in Eq. 1, each α corresponds to a translation $\mathbf{v}$ in the torso grid. Since each position in α is in the XY

Algorithm 4: Generate the Feature Vector in ACC

```
1  Input : $p_t, \mathbf{v}, \mathbf{E}, A(\mathbf{v}), \mathbf{PC}_{\boldsymbol{\Delta}}, \mathrm{C}$;
2  $T_{p_t} \leftarrow \mathrm{GetTransformaionMatrix}(p_t)$;
3  $S_f \leftarrow 0,\ \mathbf{S}_p \leftarrow [0,0,0,0]$;
4  for $\alpha$ in $A(\mathbf{v})$ do
5      $S_\alpha \leftarrow 1$;
6      for t in $\alpha$ do
7          $c_{\text{Nearest}} \leftarrow \mathrm{GetNearestContactPoint}(T_{p_t}\mathbf{t}, \mathrm{C})$;
8          $S_\alpha \leftarrow S_\alpha S(c_{\text{Nearest}})$;
9      $S_f \leftarrow S_f + S_\alpha$;
10 for $pc$ in $\mathbf{PC}_{\boldsymbol{\Delta}}$ do
11     $p_{palm} \leftarrow \mathrm{GetPalmPose}(p_t, pc)$;
12     $c_{\text{Neareset}} \leftarrow \mathrm{GetNearestContactPoint}(p_{palm}, \mathrm{C})$;
13     $i_q \leftarrow \mathrm{GetPalmQuadrant}(p_t, p_{palm})$;
14     $\mathbf{S}_p[i_q] \leftarrow \mathbf{S}_p[i_q] + S(c_{\text{Neareset}})$;
15 return $[S_f, \mathbf{S}_p]$;
```

plane, we can pre-compute all α, and label each α by its corresponding nearest $\mathbf{v}$. We denote the set of α for each $\mathbf{v}$ as $A(\mathbf{v})$.

Lines 4 to 10 in Algorithm 4 describe the process of computing the footstep score S_f. We first iterate through each α labeled as moving in the direction $\mathbf{v}$. For each foot placement in the tuple, we find its nearest contact point, and the corresponding score using Eq. 9. The multiplication shown in Line 9 captures the idea that the feasibility of each footstep transition α requires all three foot contacts to be collision-free. Finally we sum the scores for each α to obtain the footstep score S_f.

For palm contacts, we project palm contacts with a given torso pose p_t as shown in Figure 3. Each projection returns a nearest contact point c on one of the surfaces. Similar to the process in ECC, we divide the palm scores into the four quadrants of the torso frame: $\mathbf{S}_p[i]$ for the ith quadrant. This process corresponds to Lines 10 to 14 in Algorithm 4.

Given the training data consisting of the 5D feature vector, $[S_f, S_p[1], S_p[2], S_p[3], S_p[4]]$, and 1D output, $|\Gamma_+|$, we evaluate two learning methods: Support Vector Regression with an RBF kernel and Nearest Neighbor. A separate regressor is learned for each $\mathbf{v}$, resulting in 30 regressor functions.

G. Heuristic with Traversability

With the $|\Gamma_+|$ predicted by SVR using ECC or ACC, we define the traversability estimate Δg_{tr} as

$$\Delta g_{tr} = w_{tr} e^{-|\Gamma_+|}, w_{tr} \in \mathbb{R}^+. \quad (10)$$

Using this formulation g_{tr} increases when traversability is low, and the contribution of traversability saturates near 0 as $|\Gamma_+|$ increases, which reflects our observation that the planning does not become significantly easier when there is more than a sufficient number of useful footstep combinations. We then use Eq. 7 to compute g_{combined} and our proposed heuristic for contact space planning is:

$$h_{\text{combined}}(p_t) = g_{\text{combined}}(p_t) + h_p(p_t) \quad (11)$$

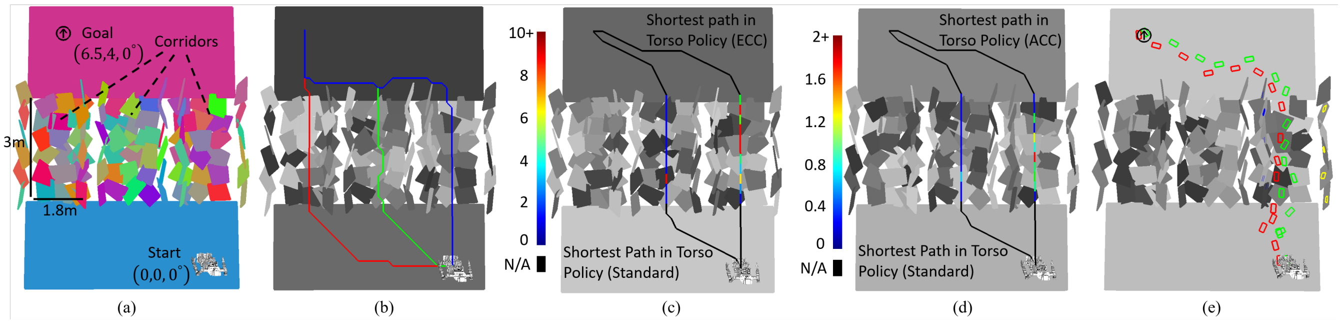
Goal
(6.5,4,0°)
Corridors
3m
1.8m
Start
(0,0,0°)
(a)
(b)
10+
8
6
4
2
0
N/A
Shortest path in Torso Policy (ECC)
Shortest Path in Torso Policy (Standard)
(c)
2+
1.6
1.2
0.8
0.4
0
N/A
Shortest path in Torso Policy (ACC)
Shortest Path in Torso Policy (Standard)
(d)
(e)

Fig. 6. The five figures show the same test environment. From left to right: (a) The testing environment; (b) The paths returned by the homotopic constrained planner; (c) The predicted number of useful footstep combinations for each step along the shortest torso path computed using 1) the standard method and 2) using traversability as predicted by regressors learned with ECC and SVR; (d) Same as (c) but using the regressors learned with ACC and SVR. The value range is smaller than the one in (c) because ACC discretizes the torso transition more finely; (e) The contact sequence returned by the contact space planner with the heuristic with traversability estimated using ACC and SVR.

Note that h_{combined} is not admissible since the traversability cost included in h_{combined} is an estimate from a learned regressor, which may overestimate the true traversability. Furthermore, it is also too expensive to calculate the traversability as part of the cost-to-come to a node in contact space planning, therefore this is not included in a node's cost and the heuristic with traversability will necessarily overestimate the cost. Thus we do not guarantee that our planner finds the optimal path. However, in practice we find that the contact sequences produced are not excessively long and local optimization can be performed as a post-processing step to decrease the length if necessary.

H. Reduced Torso Search Space for Practical Application

It takes 16 ms seconds on average to compute the feature vectors for a torso pose and a direction of motion pair with ECC, and 0.3 ms on average to compute the feature vectors for a torso pose and torso transition pair with ACC. However, since we consider both direction of motion and distance moved in ACC, there are 10 to 15 times more feature vectors calculated with ACC than with ECC. Nevertheless, it is too expensive to compute h_{combined} for every cell using either approach. Therefore, we reduce the computation time by limiting the traversability computation to the proximity of some "promising paths." We first generate a homotopically-diverse set of torso paths using A* with a naive Euclidean distance heuristic using the method described in [20], and we compute the traversability estimate only in the proximity of these guiding paths. For cells not near the guiding paths, we use $\Delta g_{tr} = w_{tr}$.

V. Experiments and Results

We evaluate the performance of using our traversability estimates in planning to navigate through random tilted surface corridors. We compare the actual performance of the planner using h, and using ECC and ACC to compute h_{combined} in test environments similar to those in Figure 6. We implemented our algorithms in OpenRAVE [8]. All experiments were run on an Intel Core i7-4790K 4.40 GHz CPU with 16GB RAM. We used the following parameter values: $\lambda = 0.1m, \hat{t} = 1s, n_{\text{p,lim}} = 6, w_s = 10, w_{tr} = 5$.

To evaluate the performance of the traversability estimate, we set up the test environment to be three parallel random tilted surface corridors, as shown in Figure 6. The planner will find a path through one of these corridors, and use both palm and foot contacts when moving through the corridors. We fix the start and the goal position, and measure the time required for the planner to find a feasible contact sequence from the start to the goal either using h (the baseline) or h_{combined} (the proposed approach) with either ECC or ACC. We set the time limit to be 300 seconds. A trial is counted a success if a plan is returned within the time limit. We run 100 trials in different environment, among which 5 where all planners fail. Since there is no evidence that a feasible path exists in these environments, we do the evaluation on the remaining 95 trials.

Since the model we trained to estimate the traversability is based on the assumption that both palm and foot contacts should be used, we only apply the traversability estimate in the random tilted surfaces corridor part of the environment. In the flat part of the environment, we do not consider the traversability.

In the experiment, the standard planner, which uses the h heuristic, would always take the middle corridor since this path is slightly shorter than the other two. However, using the proposed h_{combined} will also consider the difficulty of going through each corridor, and may take a different path, as shown in Figure 6.

Figure 7 shows the quantitative results of the planner using the two heuristics. We compare the performance of the standard planner and four planners using heuristics with traversability. Among those four planners, two compute the feature vector with ECC, and the other two use ACC. In each pair of planners, one predicts traversability using SVR, and the other finds the nearest neighbor of the feature vector in the training dataset and returns its label (NN). NN serves as a baseline to evaluate the learning performance of SVR.

The planners using heuristic with traversability outperform the standard planner in number of successful trials except for the one using ECC and NN. However, computing the feature vectors with ECC is very time-consuming, which causes it to be slower than the standard planner. The planners with

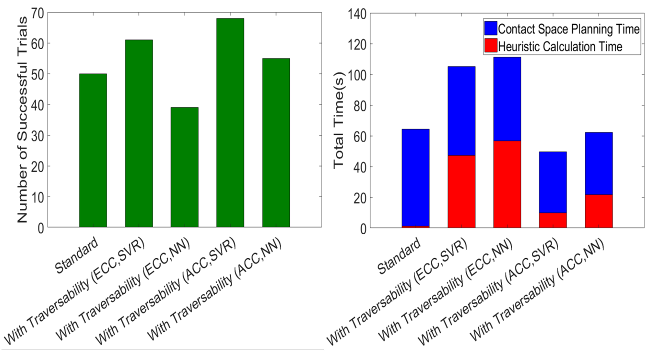

Fig. 7. Left: The number of successful trials using different heuristics Right: The average planning time of successful trials using different heuristics

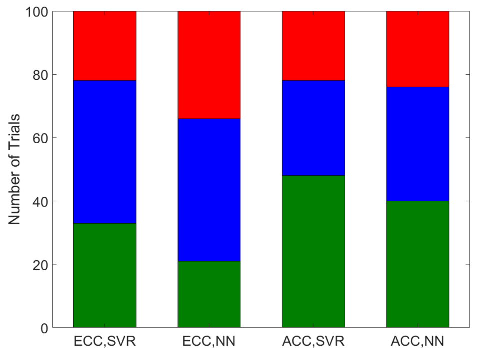

Fig. 8. Number of trials in which the standard planner or each planner using heuristics with traversability return a solution faster. Green bars show the number of trials in which each indicated planner is faster than the standard planner. Blue bars show the number of trials the standard planner is faster. Red bars represent the trials that both planner fails.

traversability-ACC outperforms its counterpart using ECC in terms of both the number of successful trials and planning time. This is due to two reasons: First, ACC is much more efficient than ECC in computing the feature vector. Second, the regressors using ACC capture the torso pose transition more precisely, which leads to a drop in planning time, and thus increases the number of successful trials by reducing the cases over time limit. We can also observe that using NN to estimate traversability results in a much lower number of successful trials than using SVR for both ECC and ACC. This result shows the benefit of using SVR to generalize from the training data.

From Figure 8, we further analyze the efficiency of each planner compared to the standard planner by counting the number of trials each planner returns a contact sequence faster than the standard planner. Although the planner using ECC and SVR has higher number of successful trials than the standard planner, its long heuristic computation time causes it to have fewer winning trials over the standard planner. We can also tell from Figure 8 that the planners using ACC outperform the standard planner in winning trials, and applying SVR further improves the performance.

VI. Conclusion

In this paper, we propose a learning framework and two kinds of features, ECC and ACC, for discrete search-based contact space planning which learns to estimate environment traversability. The results suggest that the proposed ACC heuristic is both more robust and more efficient than conventional distance-based heuristics in dealing with environments with sparse contactable regions. For future work, we would like to expand the application of traversability estimation to allow the planner to apply different strategies in regions with different traversability.

Acknowledgments

This work was supported in part by the Office of Naval Research grant N00014-17-1-2050. Special thanks to colleagues in the ARM Lab for feedback and discussion.

References

[1] 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.

[2] 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.

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

[4] A. Chilian and H. Hirschmuller. Stereo camera based navigation of mobile robots on rough terrain. In *IROS*, 2009.

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

[6] C. Cunningham, W. L. Whittaker, and I. A. Nesnas. Improving slip prediction on mars using thermal inertia measurements. In *RSS*, 2017.

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

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

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

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

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

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

[13] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA* : Anytime A* with provable bounds on sub-optimality. In *NIPS*, 2004.

[14] Y.C. Lin and D. Berenson. Learning traversability using contact space transition models for humanoid navigation in uneven terrain. In *RSS Workshop*, 2017.

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

[16] M. Shneier, T. Chang, T. Hong, W. Shackleford, R.r Bostelman, and J. S. Albus. Learning traversability models for autonomous mobile vehicles. *Autonomous Robots*, 24(1):69–86, Jan 2008.

[17] B. Suger, B. Steder, and W. Burgard. Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3d-lidar data. In *ICRA*, 2015.

[18] S. Tonneau, N. Mansard, C. Park, D. Manocha, F. Multon, and J. Pettre. A reachability-based planner for sequences of acyclic contacts in cluttered environments. In *ISRR*, 2015.

[19] J. van den Berg, R. Shah, A. Huang, and K.Y. Goldberg. Anytime nonparametric A*. In *AAAI*, 2011.

[20] C. Voss, M. Moll, and L. E. Kavraki. A heuristic approach to finding diverse short paths. In *ICRA*, 2015.

[21] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krsi, R. Siegwart, and M. Hutter. Navigation planning for legged robots in challenging terrain. In *IROS*, 2016.