Integrated Affordance Detection and Humanoid Locomotion Planning

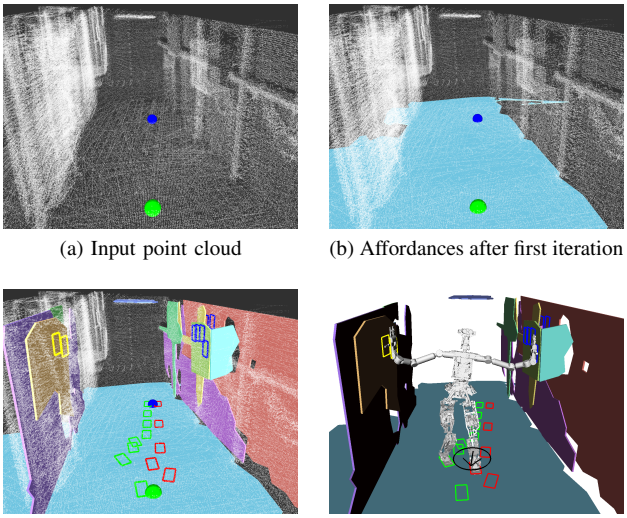
Will Pryor[1], Yu-Chi Lin[2], and Dmitry Berenson[2]

***Abstract*— A humanoid robot navigating in an unstructured environment requires knowledge of the affordances which allow it to make contact with the environment. This knowledge often comes from a perception system, which processes data from 3D sensors such as LIDAR and extracts available areas for the robot to make contact. Because perception systems run independently of the robot's planner, without knowledge of the robot's goal, they must process the entire visible area. In large environments, or those with complex geometry, the perception system may spend significant time processing areas of the environment that the planner will never consider visiting. By integrating the perception process with the planner, we are able to improve the speed with which the robot can compute a motion plan by only processing those areas of the environment which are considered by the planner for navigation. Two experiments with simulated and real-world point cloud data suggest that our framework can produce comparable plans up to seven times faster than the perceive-then-plan approach.**

I. INTRODUCTION

Systems for autonomous robot operation in unstructured environments traditionally consist of three parts: a perception system, which builds a model of the environment from sensor data, a motion planner, which computes a plan for moving through the environment to reach a goal, and a controller, which executes that plan on the robot. Controller efficiency is governed by real-time concerns, but both perception and planning run before robot motion starts. The combined runtime of the perception and planning systems determines how quickly the robot can begin task execution. Improving the speed of perception and planning decreases the time spent before the robot begins acting, which is important in time-sensitive tasks such as disaster response or human-robot interaction.

The relationship between the time costs of the perception and planning systems depends on the chosen environment representation. Those representations vary, but the implementation we consider describes the environment in terms of affordances. The term *afforance* was introduced by J.J. Gibson in the context of cognitive psychology to describe the possible actions suggested by the shape or other visible features of an object [1]. For example, a chair affords "sitting", small objects afford "lifting", and a floor affords "support". Gibson argued that these affordances were directly perceptible by the organism without the need for an intermediate representation, and that this perception was economical, only extracting the necessary information rather than modeling the entire environment. We extend this idea of economical perception by limiting the affordances perceived to only those which may be of use to the planner. This allows the robot to avoid the effort of perceiving many of the affordances in the environment, reducing the time spent performing perception.



(a) Input point cloud (b) Affordances after first iteration

(c) Affordances after planning completes (d) Executing plan

Fig. 1. Integrated Planning and Affordance Detection only identifies the affordances in areas considered by the planner. The start position is shown as a blue sphere and the goal as a green sphere. The computed path, using foot (red and green) and hand (yellow and blue) contacts, is shown along with all the affordances that have been detected. In the background more points are visible which were not processed because affordances there are not necessary to reach the specified goal. This data was collected from a ship environment using a laser scanner mounted on a quadcopter.

In order to do this, the perception system must be aware of which areas of the environment may be of use to the planner. We make use of the fact that common planners, including A* and its variants, build paths iteratively, and each iteration is constrained to use affordances near those used by the previous iteration. This allows us to define a volume of interest for each new node considered by the planner which contains every affordance that can be used at that node. Our key contribution is an integrated planning and perception system which is capable of performing affordance detection in small volumes of the environment without sacrificing accuracy and which integrates with the Anytime Nonparametric A* (ANA*) planner [2] to detect affordances on demand as the planner considers potential actions (see Figure 1). In this paper we limit the types of affordances detected to planar support surfaces, however, the technique can extend to other geometric primitives. Because complex, unstructured

[1]Will Pryor is with Worcester Polytechnic Institute (WPI), Worcester, MA, USA jwpryor@wpi.edu

[2]Yu-Chi Lin and Dmitry Berenson are with the University of Michigan, Ann Arbor, MI, USA linyuchi@umich.edu, berenson@eecs.umich.edu

environments with potential disturbances require both the hands and feet to be used to either traverse the environment or to mitigate disturbances, we wish to detect both hand and foot affordances, i.e. any planar region which will be useful for locomoting to a goal. We also wish to operate in environments where disturbances are expected, such as on board a ship, which necessitates the use of hand contacts to ensure stability.

Given a bounding-box query from the ANA* planner based on the location of the robot at a given node and the proposed transition, our affordance detection method detects every affordance within the bounding box from a point cloud representation of the environment. New affordances within the volume are identified using a combination of Region-Growing Segmentation (RGS) and Random Sample Consensus (RANSAC). New affordances are then expanded to identify points outside the query volume which also belong to the affordance. Once every point within the affordance is identified, the polygon describing the affordance is computed and the affordance is stored for future use. The newly detected affordances are combined with previously detected affordances which are wholly or partially contained within the given volume to form the set of affordances within the volume. The ANA* planner uses this set of detected affordances to determine if the transition it is evaluating is feasible. In this way, the ANA* planner computes a plan as a sequence of palm/foot contacts, where every contact is inside an affordance.

We test our method on two environments using simulated and real-world point cloud data. In these experiments, our method outperforms the standard perceive-then-plan approach by a factor of 1.6x in a simple, low-noise environment and by over 7x in a complex, noisy environment. The experiments also show that the paths produced are of comparable quality to paths produced by a standard perceive-then-plan approach.

II. RELATED WORK

Several works have adapted the theory of affordances from its original psychological context to the field of autonomous robotics. Some, such as [3], [4], [5], investigate learning affordance types from experimentation. Others, such as [6], [7], [8], [9], address the identification of specific instances of affordances within a given environment.

In [6], affordances identified by human operators and refined using model fitting techniques are used to build a task-level plan. In [7], [8], a pipeline is introduced which identifies available affordances of predetermined types in the environment according to rules and uses these affordances to construct a motion plan. In [9], objects within the environment are identified and a classifier is used to assign affordances to these objects. However, to our knowledge, there have been no publications which make use of information from the planner to reduce the number of affordances which must be identified.

With the exception of [6], which relies on human assistance, all methods described above incorporate a segmentation step to separate the objects in the environment before assigning affordances. We consider only affordances associated with planes, and therefore segment the environment into planar surfaces.

This problem of plane detection has been extensively studied outside the context of affordances. Algorithms to do so generally fall in three categories. The first is the Hough transform, used in [10], [11], which defines a parameter space into which the input data is transformed. That parameter space is then clustered to find the representative planes in the environment. The second are RANSAC-related methods, including [12], [13], [14], which repeatedly generate a large number of models from randomly-selected minimal sets of points and select the one which is most successful at explaining the data until no additional models can be found. The last category includes region-growing methods, such as [15], [16], which expand point clusters outwards from seed points according to cluster membership tests, which often include Euclidean distance and difference of point normals. A variation on point-based region growing algorithms are superpixel- or subwindow-based region growing methods, including [17], [18], [19]. In these methods, the points are divided into subwindows, each of which is subjected to a plane fitting. Region growing is performed over the subwindows with successfully fitted planes. In [19], points within subwindows to which planes could not be fit are included by a point-based region growing step which takes place after subwindow-based region growing. We use similar approaches to region growing, however, the key contribution of our approach is to do this within the planner so that only the necessary points are processed.

III. PROBLEM STATEMENT

We consider the problem of locomotion planning for a humanoid robot, which is the problem of finding a sequence of actions which result in the robot occupying some specified goal region while maintaining geometric, static balance, and collision constraints. The affordances required for this task are those which allow the robot to move itself through the environment and to maintain balance. We assume that all static planar surfaces within the environment offer these affordances.

We infer the extents and locations of the static planar surfaces in the environment from data collected from onboard sensors immediately before planning. This enables planning on previously-unmodeled environments, such as those in a disaster site. Such sensors produce data in the form of a point cloud, which is an unordered, finite set of points $P \subset \mathbb{R}^3$ where each point lies on a surface in the environment.

To describe the affordances offered by static surfaces, we approximate the environment with a set of planar surfaces S, where each surface $s \in S$ is defined by a set of points in P which are assumed to be sampled from a single surface in the environment with sensor noise. For a surface s, we wish to define an estimated plane π_s with normal $\hat{\mathbf{n}}_\mathbf{s}$ equal to the outer surface normal. We also define the surface's estimated area as a polygon $a_s = \langle v_s, e_s \rangle$, where v_s is a set of vertices

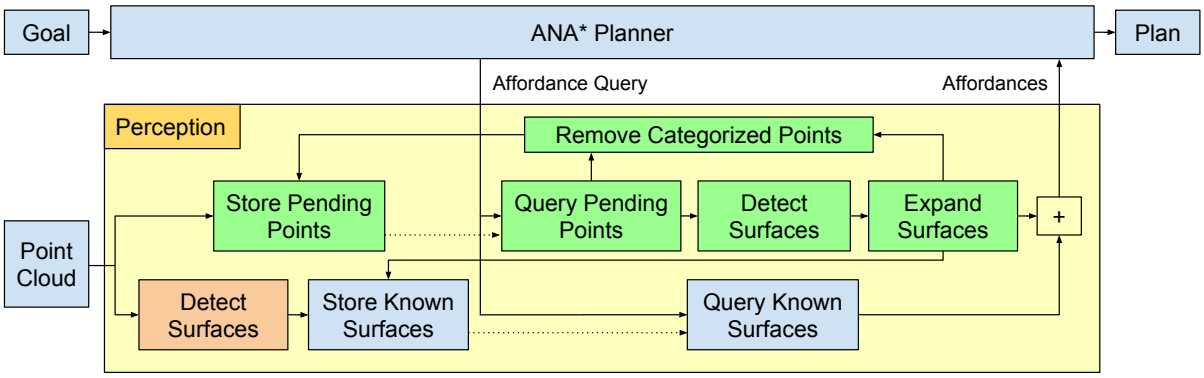


Fig. 2. The incremental affordance detection and the non-integrated approach are shown together. Steps in the proposed method are shown in green, steps in the non-integrated method are shown in orange, and steps common to both are shown in blue. Solid arrows connect consecutive steps while dotted arrows connect corresponding store and query steps. In the non-integrated approach the Detect Surfaces step runs before any affordance queries can be received, while in the proposed method the point cloud is simply stored. When an affordance query is received, the proposed method performs detection using only the points within the query region, then expands the detected affordances. These new surfaces are combined with the result of querying known surfaces and are returned to the planner. In the non-integrated approach, only the Query Known Surfaces step is performed for each affordance query.

lying on the plane π_s and e_s is a set of edges. We also wish to define a triangle mesh m_s, for the purposes of checking robot-environment collisions in the planner.

The action associated with the locomotion and balance affordances is that of making one or more contacts with the environment. Each state in the planner is described as a set of contacts between the environment and an end effector of the robot. We define E as the set of end effectors of the robot. A contact set $C = \{c_{1,\mathbf{p}_1}, c_{2,\mathbf{p}_2}, \ldots\}$ is a set of contacts which should be satisfied simultaneously, where $c_{i,\mathbf{p}_i}$ denotes that the end effector with index i should be placed at some pose $\mathbf{p}_i \in SE(3)$. A contact is valid if the contact area of the end effector contact surface (i.e. palm or sole) lies entirely within the polygon of any $s \in S$. A contact set C is valid if every contact in C is valid and there exists at least one configuration of the robot which achieves every contact of the set while meeting the reachability and balance constraints. A valid path is a sequence of valid contact sets where the first contact set is satisfied by the robot's initial configuration, the last contact set places the robot in the goal region, and every contact set except the first differs from its predecessor by the addition, removal, or change in pose of one contact. Once such a sequence of contact sets is obtained we can use interpolation and Inverse Kinematics (IK) to construct a whole-body trajectory for the humanoid robot. The goal of the method in this paper is to decrease the computation time of the entire perception and planning process needed to generate this trajectory.

IV. INCREMENTAL AFFORDANCE DETECTION

We observe that many of the affordances in a complex environment are not used in the computed plan. We also observe that, for planners which build the plan iteratively, each iteration will only require knowledge of the affordances within a volume bounded by the robot geometry. Using this information we can limit the affordance detection to only those affordances which lie at least partially within the union of those volumes. In large or complex environments, this total volume will often contain only a small fraction of the affordances within the environment, greatly reducing the time required to perform the perception.

Such a system will necessarily require integrating affordance detection with the planner, since the volume to be considered at each iteration depends on the contact set which was determined at a previous iteration. We also observe that the volumes frequently overlap between subsequent transitions. Because of this large degree of overlap, affordance detection must be able to store the results of each detection rather than duplicating work. We refer to an affordance detection system with this ability as *incremental affordance detection.*

Our incremental affordance detection algorithm is described in Algorithm 1 and diagrammed in Figure 2. It receives a request for affordances within a specified volume, which we represent as an oriented bounding box b, and responds with a list of every surface in the environment that is at least partially within b. It maintains a set of pending points $P_{pending}$, which includes all points that have not yet been identified as members of any surface or as outliers to all surfaces, and a set of previously-detected surfaces S_{known}.

A query response begins by querying the set of pending points $P_{pending}$ for all points which could lie on a surface within the query box b. We then run a surface detection step on the resulting points, described in Section IV-A, resulting in a set of new surfaces S_{new}. This is followed by a surface expansion step on the newly detected surfaces, described in Section IV-B, which identifies additional area in each surface outside of b. We refine the estimated plane using the expanded points and compute the representation used by the planner as described in Section IV-C. Simultaneously, the set of previously-detected surfaces S_{known} is queried for surfaces which intersect with the query box $S_{known,b}$. Finally, $S_{new} \cup S_{known,b}$ is returned to the planner. After the data is returned we remove all identified inliers and outliers from $P_{pending}$ and add S_{new} to S_{known}.

In order to ensure correctness, the response must be guaranteed to contain every affordance within the requested

Algorithm 1: Incremental Afforadance Detection

input : b: The query bounding box
$P_{pending}$: Points which have not yet been processed
S_{known}: Surfaces which have been detected
output: S_b: All surfaces within b
$b_+ \leftarrow b$ with each extent increased by w_{min} ;
$P_{b+} \leftarrow P_{pending} \cap b_+$;
$S_{known,b} \leftarrow S_{known} \cap b$;
$S_{new} \leftarrow detectSurfaces(P_{b+})$;
$S_{new} \leftarrow expandSurfaces(S_{new}, P_{pending})$;
for $s \in S_{new}$ **do**
  $\pi_s \leftarrow estimatePlane(s)$;
  $a_s \leftarrow computePolygon(s)$;
  $m_s \leftarrow computeMesh(s)$;
$P \leftarrow \{p \in P \mid p \notin b \wedge \forall s \in S_{new}\, p \notin s\ \}$;
$S_{known} \leftarrow S_{known} \cup S_{new}$;
return $S_{new} \cup S_{known,b}$ **;**

volume. If this guarantee were not met, the planner may incorrectly discard potential contacts. However, we must estimate the surface's plane in the presence of noisy input data, and such estimation is inaccurate in cases where the distribution of points on any axis parallel to the plane is not wider than the distribution of the points perpendicular to the plane. We therefore impose a minimum width w_{min} on all detected surfaces, both to ensure accuracy and to avoid considering surfaces which are too narrow to contain any end effector of the robot. We maintain the guarantee that all affordances within the given area are detected by expanding the query box by w_{min} in all directions. We refer to the resulting padded query box as b_+.

A. Surface Detection

The surface detection algorithm, described in Algorithm 2, detects all surfaces within b_+. The surface normal at each point in $P_{pending}$ within the volume b_+ is computed using Moving Least Squares (MLS) Estimation [20], which computes every point's normal vector by fitting a plane to its neighbors within a specified radius r_{mls}. This normal information is used in Region Growing Segmentation (RGS) [21], which identifies surfaces in the environment by repeatedly growing randomly-seeded regions by recursively adding all neighbors within a distance $d_\parallel$ of existing points whose normal vectors are within a specified angle θ_{rgs} of the seed point's normal. RGS yields a set R_{RGS} of regions, where each region $r \in R_{RGS}$ is a set of points on the same surface or on adjacent, nearly co-planar surfaces.

Each region in R_{RGS} is then subdivided into one or more planar surfaces using Random Sample Consensus (RANSAC) Segmentation [22]. RANSAC fits a plane model to a set of points by repeatedly choosing three points at random to define a plane, finding all points which lie within some distance $d_\perp$ of this plane, and re-computing the plane model using the inliers. This process is repeated many times for each region and the plane with the highest number of inliers is returned. After each application of RANSAC, if more than a specified number of points n_{min} remain, the process is repeated on the remaining points to identify another surface.

Algorithm 2: Surface Detection

input : P_{b+}: Points from which to detect surfaces
output: S_{new}: Set of newly-detected surfaces
$S_{new} \leftarrow \{\}$;
$N \leftarrow MLS(P_{b+})$;
$R_{RGS} \leftarrow RGS(P_{b+}, N)$;
$R_{RANSAC} \leftarrow RANSAC(R_{RGS}, N)$;
$R \leftarrow EuclideanSegmentation(R_{RANSAC})$;
for $r \in R$ **do**
  $\pi = PrincipalComponent(r)$;
  if $width(r, \pi) \geq w_{min}$ **then**
    $S_{new} \leftarrow S_{new} \cup \{r\}$;
return S_{new} **;**

Algorithm 3: Surface Expansion

input : S_{new}: Surfaces to expand
P: Points which have not yet been processed
output: S_{new}: Set of expanded surfaces
for $s \in S_{new}$ **do**
  $P_{plane} \leftarrow \{p \in P \mid -d_\perp \leq \mathrm{SD}(p, s) \leq d_\perp\}$;
  $P_{limit} \leftarrow \{p \in P \mid d_\perp < \mathrm{SD}(p, s) \leq d_\parallel\}$;
  for $p_{inlier} \in s$ **do**
    $p_{limit} \leftarrow$ nearest neighbor to p_{inlier} in P_{limit} ;
    $d \leftarrow \min(d_\parallel, projectedDist(p_{inlier}, p_{limit}))$;
    $s \leftarrow s \cup$ points within d of p_{inlier} in P_{plane} ;
return S_{new} **;**

The plane model of each region r identified by RANSAC is then re-estimated using Principal Component Analysis (PCA), with the last principal component selected as the plane's normal vector. If the minimum width of the plane in any direction is at least w_{min}, the surface defined by r is added to the set of surfaces to return. Because we guarantee that every surface within b is detected (by using the padded query box b_+), and every surface's inliers are identified, we know that every inlier point within b will be found. We therefore know that all other points within b must be outliers, and can be deleted from $P_{pending}$ without affecting any affordance that has not yet been detected.

B. Surface Expansion

Once a contact is placed on a surface by the planner, that surface's plane model cannot be changed without the possibility of invalidating that contact. However, the surfaces identified by Surface Detection only contain inliers within b_+, which may only be a small part of the affordance. Position error due to misalignment of the plane model increases with distance, and errors which may be insignificant across the small distance within b_+ have the potential to lead to large errors if the surface is later found to be much

larger than b_+. Any adjustment to the plane model after its first use by the planner requires either a potentially-costly re-validation for existing contacts or breaking surfaces into multiple parts, artificially introducing seams which the planner must unnecessarily avoid.

To avoid problems caused by returning partial representations of affordances we run an expansion algorithm, described in Algorithm 3, on each surface identified by Surface Detection. This expansion algorithm identifies all points that belong to a partially-detected surface without the need to run the costly Surface Detection step on the entire environment.

Surface Expansion on a surface s behaves similarly to Region Growing Segmentation limited to the plane π_s. However, rather than compute the normal vectors of every point, we use the presence of nearby points above but not within the plane of s to indicate whether a given area belongs to a different surface. We define the signed distance $SD(p, s)$ from any point p to a surface s as the projection of the vector from p to any point on s onto $\hat{\mathbf{n}}_\mathbf{s}$. We consider a point p to be above the surface if the signed distance $SD(p, s)$ is positive, and nearby the surface if the magnitude of $SD(p, s)$ is less than $d_\parallel$ but greater than the inlier distance threshold $d_\perp$. We refer to points which are both above and nearby the surface as *limit points*, denoted P_{limit}.

Every point in P, and therefore every point in P_{limit}, was sampled from either the surface currently being expanded s or another surface in the environment. We assume that $d_\perp$ is sufficiently large that there is a negligible probability of encountering a set of points in P_{limit} which were sampled from s and are dense enough to prevent expansion into their enclosed area. Every point in P_{limit} sampled from a surface other than s indicates the presence of either a surface above s or a surface which intersects π_s. In either case, the area of the surface below the limit points does not afford stability or locomotion, and so points in that area are excluded from the set of inliers.

Surface expansion on a surface s begins by computing P_{plane} as the set of all points in $P_{pending}$ which are within $d_\perp$ of the plane and P_{limit} as the set of all points in $P_{pending}$ between $d_\perp$ and $d_\parallel$ of the plane, i.e. points which indicate the presence of another surface. We then examine the neighborhood of every inlier point $p_{inlier} \in s$. We define the size of the neighborhood d to be the smaller of $d_\parallel$ or the distance between p_{inlier} and its nearest limit point along the plane of the surface, as shown in Figure 3. We then add every point in P_{plane} within distance d of p_{inlier} to the surface s. This process continues recursively until there are no remaining points within the neighborhood of any points in s. Once every point in s has been considered, s approximates the set of every point sampled from a single surface in the environment.

C. Surface Representation

The combination of surface detection and expansion results in surfaces that are completely defined by a plane equation and a set of inlier points. However, the planner requires each surface s be represented as a plane π_s, a polygon a_s, and a mesh m_s. We define a_s as the 2D α-shape of the points in s projected into the plane π_s. The α-shape of a set of points is a generalization of the convex hull which can represent concavities in the set of input points [23]. The parameter α controls the radius of the disc which "carves out" areas of the convex hull, with negative values corresponding to concave shapes. We use an α value of $\alpha = \frac{1}{-d_\parallel/2}$, which causes the α-shape to represent all concavities large enough to inscribe an empty disc of diameter $d_\parallel$.

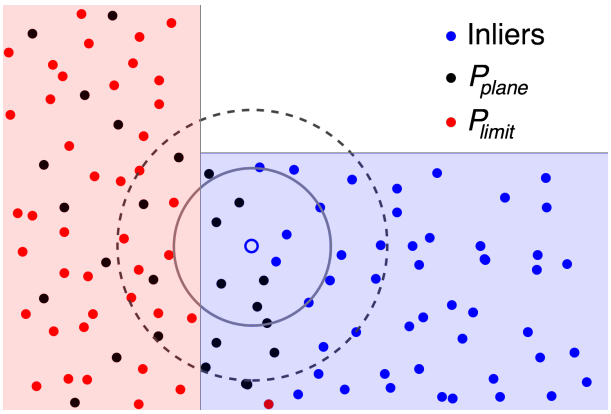


Fig. 3. A top-down view of an environment with a horizontal surface, shown in blue, and a vertical surface, shown in red. Points in blue have been identified as members of the horizontal surface, points in black are in the plane of the horizontal surface, and points in red are limit points, i.e. above the plane of the surface. The outlined point is being considered for expansion. Because of the presence of a nearby limit point, the radius to search is reduced from the dashed circle to the solid circle, preventing expansion from including black points which are not members of the horizontal surface.

According to the definition of the α-shape, every edge in the resulting polygon a_s belongs to a triangle whose circumscribed radius is at most $d_\parallel/2$. Every edge in a_s therefore has length less than $d_\parallel$. However, many real-world surfaces, especially in human-made environments, can accurately be described with a small number of longer edges. Reducing the number of edges in a_s is desirable because it reduces the time required for a point-in-polygon test, which is performed frequently in the planner. We therefore apply the polygon simplification algorithm described in [24]. This algorithm removes every point which can be removed without changing the polygon topology (e.g. introducing intersections) or causing the distance between any original vertex and the simplified boundary to exceed $d_\parallel$.

The planner also performs a collision check on each contact candidate to ensure it does not lie inside a surface. This collision check uses a triangle mesh encompassing the volume created by extruding the area covered by the surface's polygon by some small distance $d_{extrude}$ in the direction opposite the plane normal.

V. PLANNER INTEGRATION

Our planner formulates the contact planning problem as a graph search problem and solves it with ANA* [2]. Each node in the search graph is defined as a contact set $C = \{c_{1,\mathbf{p}_1}, c_{2,\mathbf{p}_2}, \ldots\}$ as described in Section III. Each contact

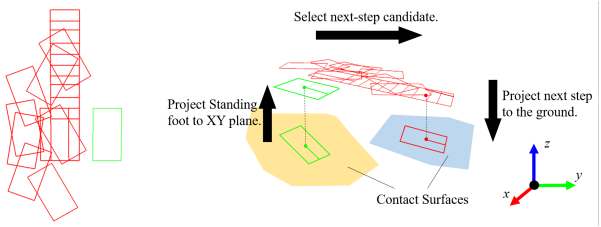


Fig. 4. The foot transition model specifies possible poses for each foot as projections into the xy plane. The actual contact pose is determined by projecting the transition model pose into the appropriate surface plane.

should place the end effector on the polygon of any one surface s_c in the environment, and should not place the end effector into collision with any affordance except s_c. The robot should be able to achieve static balance while satisfying every contact in the set. We require that both feet appear in every contact set and allow the user to specify the minimum size of the contact set. In our experiments we specify a minimum size of three, requiring one hand to be in contact at all nodes.

The actions available to the planner are adding one new contact, removing one contact, and changing the pose of one contact. The planner is constrained to not allow the same end-effector to transition in subsequent actions. Given a contact set, the possible poses of new and changed contacts are described with a pre-defined transition model relative to the current end-effector poses. 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 plane of the corresponding surface to get next foot contact poses, as shown in Figure 4. The corresponding surface for a foot contact is the surface with the highest projected contact pose for which the vertical distance from the projected pose to the standing foot's pose is no more than some distance h_{step} and the end effector lies entirely within the surface's polygon. If no corresponding surface exists for a given contact, the action which created that contact is not added to the ANA* graph.

For hand contacts, we first approximate the shoulder position based on the foot poses, and project hand contacts along predefined rays from the approximated shoulder point to the corresponding surface to get the possible next hand contact poses, as shown in Figure 5. The corresponding surface for a hand contact is the surface with the closest projected contact pose to the approximated shoulder point for which the distance between the projected contact pose and the approximated shoulder point is no more than some distance d_{arm} and the end effector lies entirely within the surface's polygon.

For both of these transition models it is possible to determine an oriented bounding box which contains every possible contact pose, and therefore contains every affordance that will be of use to the planner. In the case of the foot transition model, we align the z axis of the bounding box with the global z axis. The extents of the bounding box in the x and y directions are defined by the 2D bounding box

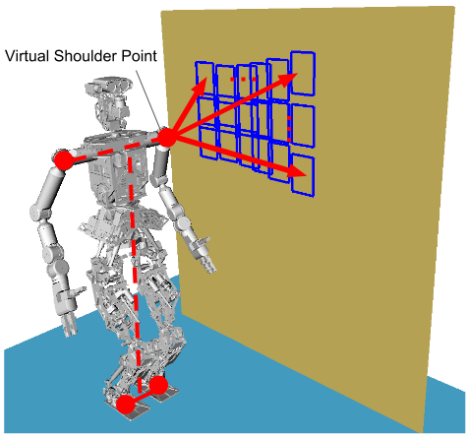


Fig. 5. The hand transition model specifies possible poses for each hand as rays relative to the estimated shoulder position. The actual contact pose is determined by finding the closest intersection with the ray and a surface.

of the step transition model, which is computed in advance. The extent in the z direction is equal to $2h_{step}$, and the box is centered at the standing foot. For the hand transition model, we align the z axis of the bounding box along the vector between the approximated positions of the left and right shoulder. The extent in the z direction is equal to d_{arm}. The extents in the x and y directions are defined by the 2D bounding box of the intersections between the rays of the transition model and a plane perpendicular to the z axis at the distance d_{arm} from the shoulder. The box's center lies on the line between the approximated shoulder positions at a distance of $\frac{1}{2}d_{arm}$ from the shoulder. At each step of the planner, before each end effector's transition model is applied, the planner queries incremental affordance detection to find all surfaces within the specified bounding box.

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

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

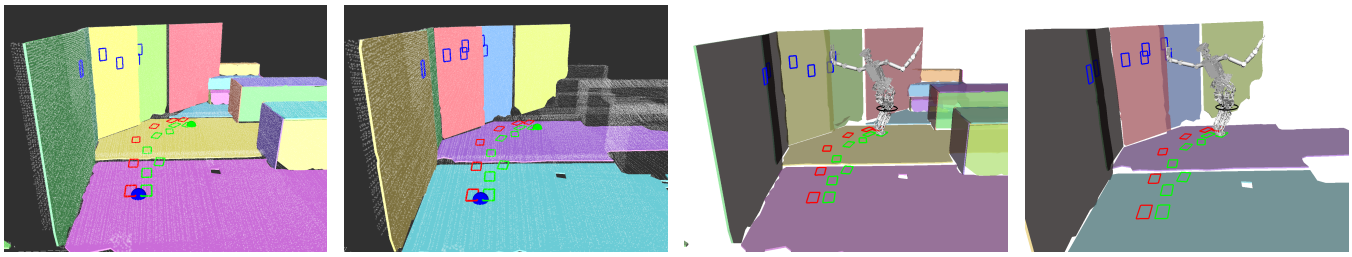
where d_e is the translation of the moving end-effector, d_θ is the difference in robot orientation (defined as the mean of the two feet's rotation about the Z axis), and w_θ and w_s are the weight for robot orientation difference and the step cost, respectively. Adding step cost helps reduce the number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

The heuristic for each state $\mathbf{x}$ used in the planner is:

$$h(\mathbf{x}) = w_\theta h_\theta(\mathbf{x}) + \sum_i^{|E|} \left(h_{e,i}(\mathbf{x}) + w_s \frac{h_{e,i}(\mathbf{x})}{d_{e,i,max}} \right) \tag{2}$$

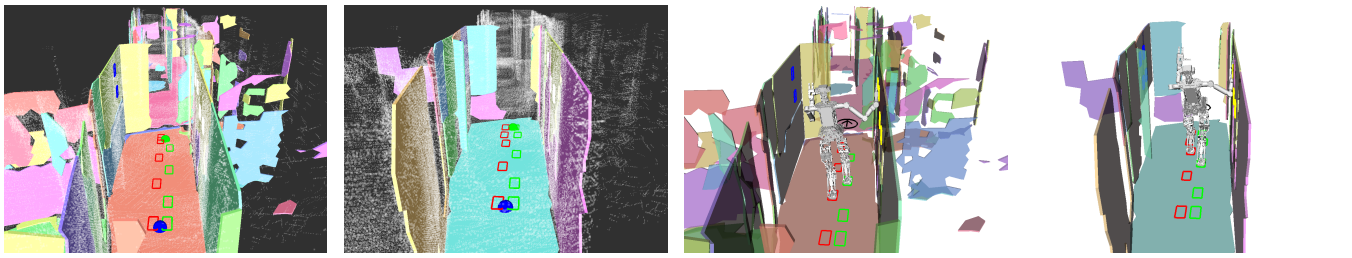
where h_θ is the difference between the current and goal robot orientations, $h_{e,i}$ is the Euclidean distance between the pose of end-effector i and the goal, and $d_{e,i,max}$ is the maximum possible translation for end-effector i in one action.

To get 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 to obtain the robot configurations at each interpolated pose set.



(a) Baseline: Affordances (b) Integrated: Affordances (c) Baseline: Plan (d) Integrated: Plan

Fig. 6. (a-b) Affordances detected by the baseline and integrated framework in the office environment. (c-d) Plans produced by the baseline and integrated planner in this environment.



(a) Baseline: Affordances (b) Integrated: Affordances (c) Baseline: Plan (d) Integrated: Plan

Fig. 7. (a-b) Affordances detected by the baseline and integrated framework in the real-world ship environment. (c-d) Plans produced by the baseline and integrated planner in this environment.

VI. RESULTS

To evaluate the improvement in planning time of our method vs. the standard approach, we implemented a standard perceive-then-plan framework as a baseline by performing one query to the perception system with a bounding box of 10m in every direction and then running ANA* on the result. This baseline uses the same perception algorithms described above so that the comparison is fair. We compare this baseline to our integrated approach in two environments: The "ship" dataset was collected from the internal area of a ship using a laser scanner mounted on a quadcopter and contains 590,000 points. The "office" dataset contains 230,000 points collected from a simulated office environment.

The perception method was implemented in C++ using the normal estimation and segmentation implementations in PCL [25] and the α-shape and polygon simplification algorithm implementations in CGAL [26]. The planner was implemented using the OpenRAVE planning environment [27]. The robot is the ESCHER humanoid robot with six degrees of freedom per leg, seven per arm, and one in the waist. Tests were run on an Intel Core i7-3770K 3.40 GHz CPU with 16GB RAM.

For all experiments we use a minimum plane width of $w_{min} = 0.15$m, a Moving Least Squares radius $r_{mls} = 0.15$m, a Region Growing Segmentation maximum angle of $\theta_{rgs} = 0.1$ radians, a minimum number of points $n_{min} = 50$, and an extrusion distance $d_{extrude} = 0.02$m. For the less noisy office dataset we use an inlier point-to-plane distance threshold $d_{\perp} = 0.02$m and a surface distance threshold $d_{\parallel} = 0.03$. For the noisier ship dataset we use the values $d_{\perp} = 0.04$m and $d_{\parallel} = 0.08$m. In the planner, for all experiments we use a maximum step height $h_{step} = 0.5$m, a maximum arm length $d_{arm} = 0.7$m, a maximum transition distance $d_{e,i,max} = 0.4$m for the feet and 0.6m for the hands, and cost weights $w_{\theta} = 0.01$ and $w_{step} = 10$.

TABLE I
PERFORMANCE OF PROPOSED METHOD

	Method	Total (s)	Perception (s)	Coverage (%)
Ship	Integrated	6.87 (0.116)	5.87 (0.112)	10.0 (0.00)[1]
	Baseline	53.19 (2.69)	39.32 (2.70)	71.7 (0.00)[1]
Office	Integrated	6.84 (3.66)	5.10 (2.46)	33.3 (4.40)
	Baseline	11.13 (1.47)	4.93 (0.096)	78.8 (0.653)

[1]The ship point cloud is a recording of a real measurement and does not change. Because of this, and because all algorithms which use randomness are given a fixed seed, the result of the experiments is deterministic and therefore has zero variance.

Table I shows the average and standard deviation of the combined perception and planning time and of the perception time alone, as well as the percentage of the points in the environment which were identified as part of a surface. Examples of the paths and environment representations produced in these experiments are shown in Figures 6 and 7. In our experiments, the proposed method is able to compute a motion plan from point cloud data more quickly than the baseline. In the more complex ship environment, the speed improvement is more than 7x. In the less complex office environment, the combined perception and planner speed improvement is 1.6x. The reason for the speed improvement is evident in the coverage metric. In both cases, the integrated method identifies many fewer surface inliers despite producing similar-quality paths. In the ship environment the baseline finds 7x more inliers, the same magnitude as the speed increase. In the office environment the baseline finds 2.4x more inliers while

the speed is only improved by 1.6x. The smaller and simpler office environment shows the effect of the overhead of integrated affordance detection, which includes interprocess communication and bounding box membership tests even when there are no new affordances in the query volume. However, the integrated method is still faster overall.

We also record the average and standard deviation of the path length under each method, in number of steps and in Euclidean distance, shown in Table II. We compute the Euclidean distance of a path as the sum of the distances between the mean foot poses of each consecutive pair of contact sets. We can see that the number of steps and Euclidean distance of plans produced by the two methods are not meaningfully different. This suggests that, while the paths produced using the proposed method are not identical to paths produced using the baseline, the path qualities are comparable and the paths are very similar in practice.

TABLE II
PATH LENGTH COMPARISON

	Method	Number of Steps	Euclidean Distance (m)
Ship	Integrated	17.00 (0.000)[1]	2.42 (0.000)[1]
	Baseline	15.00 (0.000)[1]	2.29 (0.000)[1]
Office	Integrated	19.70 (1.487)	2.98 (0.049)
	Baseline	21.00 (2.449)	3.06 (0.059)

[1]See Table I.

VII. CONCLUSION

A humanoid robot navigating in a complex environment must perform some type of perception to detect available affordances with which to make contact. These perception systems may spend significant time processing areas of the environment that the planner will never consider visiting because the perception algorithms have no knowledge of the robot's goal. By implementing a combined perception and planning system which limits perception to only areas considered by the planner, we improved the speed with which the robot can compute a motion plan by only processing those areas of the environment where affordances may be needed by the planner. Two experiments with simulated and real-world point cloud data showed that our framework can produce comparable plans up to seven times faster than the standard perceive-then-plan approach.

ACKNOWLEDGMENTS

This work was supported in part by the Office of Naval Research grant N00014-15-1-2138. Thanks to TREC Lab, Virginia Tech, for support with the ESCHER model, Sebastian Scherer for providing the ship point cloud data, and Calder Phillips-Grafflin for assistance with the paper.

REFERENCES

[1] J. J. Gibson, *The ecological approach to visual perception*. Hoboken, NJ: Taylor & Francis, 2014.

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

[3] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Learning object affordances: From sensory–motor coordination to imitation," *IEEE Transactions on Robotics*, vol. 24, pp. 15–26, Feb. 2008.

[4] E. Ugur, E. Oztop, and E. Sahin, "Goal emulation and planning in perceptual space using learned affordances," *Robotics and Autonomous Systems*, vol. 59, pp. 580–595, July 2011.

[5] E. Ugur, E. Şahin, and E. Oztop, "Self-discovery of motor primitives and learning grasp affordances," in *IROS*, 2012.

[6] M. Fallon, S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. P. D'Arpino, R. Deits, M. DiCicco, D. Fourie, T. Koolen, P. Marion, M. Posa, A. Valenzuela, K.-T. Yu, J. Shah, K. Iagnemma, R. Tedrake, and S. Teller, "An architecture for online affordance-based perception and whole-body planning," *Journal of Field Robotics*, vol. 32, pp. 229–254, Mar. 2015.

[7] P. Kaiser, D. Gonzalez-Aguirre, F. Schültje, J. Borras, N. Vahrenkamp, and T. Asfour, "Extracting whole-body affordances from multimodal exploration," in *Humanoids*, 2014.

[8] P. Kaiser, M. Grotz, E. E. Aksoy, M. Do, N. Vahrenkamp, and T. Asfour, "Validation of whole-body loco-manipulation affordances for pushability and liftability," in *Humanoids*, 2015.

[9] D. I. Kim and G. S. Sukhatme, "Semantic labeling of 3d point clouds with object affordance for robot manipulation," in *ICRA*, 2014.

[10] D. Borrmann, J. Elseberg, K. Lingemann, and A. Nüchter, "The 3D Hough Transform for plane detection in point clouds: A review and a new accumulator design," *3D Research*, vol. 2, pp. 1–13, Nov. 2011.

[11] F. A. Limberger and M. M. Oliveira, "Real-time detection of planar regions in unorganized point clouds," in *ICPR*, 2015.

[12] R. Schnabel, R. Wahl, and R. Klein, "Efficient RANSAC for point-cloud shape detection," *Computer Graphics Forum*, vol. 26, pp. 214–226, June 2007.

[13] F. Donnarumma, V. Lippiello, and M. Saveriano, "Fast incremental clustering and representation of a 3D point cloud sequence with planar regions," in *IROS*, 2012.

[14] F. Mufti, R. Mahony, and J. Heinzmann, "Robust estimation of planar surfaces using spatio-temporal RANSAC for applications in autonomous vehicle navigation," *Robotics and Autonomous Systems*, vol. 60, pp. 16–28, Jan. 2012.

[15] Z. Jin, T. Tillo, and F. Cheng, "Depth-map driven planar surfaces detection," in *VCIP*, 2014.

[16] S. Oesau, F. Lafarge, and P. Alliez, "Planar shape detection and regularization in tandem," *Computer Graphics Forum*, vol. 35, pp. 203–215, Feb. 2016.

[17] K. Matsumoto, F. d. Sorbier, and H. Saito, "Real-time enhancement of RGB-D point clouds using piecewise plane fitting," in *EUVIP*, 2014.

[18] Z. Wang, H. Liu, Y. Qian, and T. Xu, "Real-time plane segmentation and obstacle detection of 3d point clouds for indoor scenes," in *ECCV Workshops and Demonstrations*, 2012.

[19] J. Xiao, J. Zhang, B. Adler, H. Zhang, and J. Zhang, "Three-dimensional point cloud plane segmentation in both structured and unstructured environments," *Robotics and Autonomous Systems*, vol. 61, Dec. 2013.

[20] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Computing and rendering point set surfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, pp. 3–15, Jan. 2003.

[21] T. Rabbani, F. van den Heuvel, and G. Vosselman, "Segmentation of point clouds using smoothness constraint," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 36, pp. 248–253, Jan. 2006.

[22] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, pp. 381–395, June 1981.

[23] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel, "On the shape of a set of points in the plane," *IEEE Transactions on Information Theory*, vol. 29, pp. 551–559, July 1983.

[24] C. Dyken, M. Dæhlen, and T. Sevaldrud, "Simultaneous curve simplification," *Journal of Geographical Systems*, vol. 11, pp. 273–289, Sept. 2009.

[25] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *ICRA*, 2011.

[26] The CGAL Project, *CGAL User and Reference Manual*. CGAL Editorial Board, 4.6.3 ed., 2015.

[27] R. Diankov, *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.