Perception Methods for Continuous Humanoid Locomotion Over Uneven Terrain

by

Pat Marion

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author .....
Department of Electrical Engineering and Computer Science
May 6, 2016

Certified by .....
Russ Tedrake
Professor of Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Theses

Perception Methods for Continuous Humanoid Locomotion Over Uneven Terrain

by

Pat Marion

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Mobile robots can safely operate in environments that pose risks to human health, such as disaster zones and planetary exploration. Robots in these environments may encounter uneven terrain like debris fields, staircases, and rock ledges that are impossible to traverse with wheels or tracks but are surmountable by legged humanoid platforms through careful selection of footholds. The DARPA Robotics Challenge demonstrated that today's field robots are capable of uneven terrain traversal but they moved slowly and only for short durations. The stretches of walking are separated by longer stationary periods consumed by LIDAR data acquisition and human operator decision making. With the goal of improving autonomy, speed, and reliability, this thesis research investigates new algorithms for continuous locomotion over uneven terrain through online terrain perception and continuous footstep re-planning. A new algorithm for planar segmentation of terrain features is presented, along with a novel approach that integrates stereo depth fusion for terrain perception and online footstep re-planning using mixed-integer quadratic optimization. The approach is implemented within a novel software framework called Director, and results are validated on hardware using the Atlas humanoid robot with autonomous laboratory experiments and semi-autonomous field experiments at the DARPA Robotics Challenge Finals.

Thesis Supervisor: Russ Tedrake
Title: Professor of Computer Science

Acknowledgments

I thank my advisor, Russ Tedrake, for his support, guidance, and encouragement of my work, and his leadership of the DARPA Robotics Challenge MIT Team. I have much gratitude for all of the DRC Team MIT members, whose determination and continual hard work throughout the multi-year challenge allowed my work, and the team's work to be demonstrated on grand stage.

I would like to thank the members of the Robot Locomotion Group for many hours of thoughtful discussion and conversations, they are a brilliant group of people and it is a fantastic lab atmosphere to work in. I also thank Maurice Fallon and his team in Edinburgh for their contribution to Director and the OpenHumanoids software stack.

Finally, I would like to thank my friends for their support and many fun times together in Boston, and express my gratitude for my family and wife Bonnie for their everlasting love.

Contents

Chapter 1

Introduction

The DARPA Robotics Challenge (DRC) of 2015 showcased the state of the art of today's field ready humanoid platforms designed for search and rescue operations. The robotics community of researchers and engineers delivered their best designs for general purpose robot systems and operator interfaces to perform general mobile manipulation and locomotion tasks with a human in the loop. Though the Challenge may have been famous for the falls outtakes viral video [1], it produced the best field demonstration of humanoids to date. Indeed, with the aid of a human operations team, the prototype platforms featured many locomotion capabilities that will be required for future bipeds to serve their desired roles in search and rescue, service and aid, and extra planetary exploration.

The robots at the DRC traversed the uneven terrain with slow and deliberately placed footsteps. The robots moved only short distances before pausing to collect stationary LIDAR scans and await human executive decisions on the next course of action. Though several teams had automated planners and terrain perception capabilities, none trusted their systems to run autonomously or continuously when it counted [2].

Leading up to the DRC we were also treated to some short video snippets of an Atlas robot operating in secret at Boston Dynamics' private lab space [3]. In this video, Atlas treks quickly and with great agility over a field of smashed up pieces of cinder block debris and rocks. No laser scanner or camera is equipped, the robot is

sensing the terrain only with its feet. Thus, with this video we have another example of humanoid locomotion over uneven terrain, but it's on the opposite end of a spectrum that spans from slow, deliberately placed footsteps like at the DRC, to fast, reactive walking. The latter is suitable for some terrains, but the terrain at the DRC requires deliberate footstep placement that critically depends on accurate perception. To fully realize the utility of these humanoid robots we need locomotion capabilities from the entire spectrum with better perception to realize increased autonomy and robustness.

1.1 Motivation

A primary motivation for humanoid robotics research is to develop platforms capable of moving through the same environment as a human being – such as squeezing through confined spaces and under overhanging obstacles as well as crossing challenging terrain. While locomotion research spans actuator development, mechanism development, dynamic planning and control, in this work we focus on terrain estimation and footstep planning.

We take as motivation the recent DARPA Robotics Challenge (DRC) [4], where robots in outdoor conditions were required to walk over a course of uneven and discontinuous terrain. For humanoid robots to be useful, this kind of walking task must be automated such that the robot can locomote around or over any obstacles without stopping.

1.2 Related Work

1.2.1 Biped Navigation

Biped Navigation encompasses mapping and localization, collision free path planning, and walking trajectory generation and execution. Several research groups have published results integrating such methods on humanoid robot hardware in order to demonstrate autonomous humanoid locomotion over uneven terrain. Early work

focuses on footstep planning and walking trajectory generation, and more recent methods introduce online terrain perception and map localization.

Kuffner et al. describe an A* search method to find a set of feasible footstep locations to guide a humanoid robot through a flat ground environment with obstacle regions that must be avoided [5]. A discrete set of feasible footstep placements called an action set are sampled from a pre-computed feasible region. The authors describe a heuristic cost function that leads the search to a globally optimal solution for the action set while minimizing the number of footsteps. Variations and extensions of this forward search method are used in most of the remaining works described below. The paper also describes a method for generating statically stable transition trajectories to allow the robot to move between the planned footstep placements.

The next advance from this group is by Chestnutt et al. which extends the footstep planning method to handle uneven terrain and incorporates a method to handle non-statically stable footstep transitions [6]. The cost function encodes many heuristics based on terrain features such as height change and curvature. A simplified version of the algorithm that limits the branching of the search is capable of running online and paper includes a demonstration with an H7 robot using a stereo camera to estimate the terrain, but does not describe in detail the stereo camera and terrain estimation methods. In [7] Chestnutt describes refinements of the method and a demonstration with the Honda ASIMO humanoid. State estimation and localization is performed with an external motion capture system, the focus is online re-planning of footsteps toward a global goal with obstacle avoidance.

Chestnutt's 2009 publication [8] demonstrates biped navigation on the Toyota Partner humanoid using the previously described footstep planning methods, but now onboard sensing and odometry is a featured. Scans from a sweeping planar LIDAR are integrated into a height map- a 2D grid of [x, y, height] cells. In this work, the footstep planner is capable of stepping up onto small stairs, whereas previous work treated everything off the ground plane as an obstacle. To accomplish this, a plane detection algorithm searches the height map for point clusters that belong to a common horizontal surface using a method called two-point random sampling

[9]. Height map cells without a valid plane id are treated as obstacles. Notably, the footstep planner is able to place footsteps that overhang plane boundaries as long as four of the six foot contact sensors overlap cells with a common plane id. The general plane detection algorithm is not fast enough to run online, so a heuristic method is used to limit the sampling locations to prescribed regions where expected footsteps will be placed.

The most recent work from the group by Nishiwaki et al. integrates the previous methods onto the HRP2 humanoid platform with a dynamic walking gait generation and controller capable of stabilizing walking trajectories over terrain estimates with errors of up to a few centimeters [10] [11]. The system does not run a complete SLAM algorithm, instead it allows drift and compensates by refreshing the terrain map. The paper does not claim to be using the plane detection methods from [8], instead it plans footsteps using heuristic cost functions that operate directly on the height map, and the accumulator that builds the height map uses filtering to smooth the heights provided by the raw LIDAR sensor. The onboard computing system is capable of running the perception, planning, and controller online. The footstep planner handles uneven terrain such as stairs and ramps, and the walking controller handles unmodeled terrain features such as walking on gravel. This work is one of the most impressive and complete to date among all the literature investigating continuous biped navigation over uneven terrain. In a recent public demonstration, the SCHAFT company has shown a new commercialized design that walks up and down stairs, and walks on rough dirt terrain, but at the time of this writing no details of the perception processing have been published [12].

The Gepetto Team from the Laboratory for Analysis and Architecture of Systems at the French National Center for Science Research published their integrated biped navigation methods in Stasse et al. [13]. The paper describes their high-level system architecture for integrating vision with a footstep planning method and a dynamic walking pattern generator. The walking pattern generator uses the method proposed by Kajita in [14]. Stasse demonstrates autonomous walking on flat ground with obstacle avoidance using a HRP-2 humanoid platform, and builds on his prior work

in dynamic stepping over an obstacle.

In a series of papers by Perrin et al. the group proposes new motion planning methods for footsteps based on swept volumes of the feet and upper body [15] [16] [17] [18]. The footstep planner uses a discrete set of candidate footstep locations, similar to [5] but integrates it with new methods of collision avoidance and walking pattern generation. The upper legs and upper body parts of the robot are represented with a single bounding box that moves as a swept volume, while the feet and lower legs have a discrete bounding box that can move up, over, and around obstacles. The paper is interesting because it describes their obstacle representation and how it is generated from their environment map. However, in this method the robot is not able to change height, such as the ability to climb stairs, and the swept volume representation is somewhat conservative. The environment is modeled as flat ground with obstacles, and the planner does not support footsteps placed on top of obstacles.

The paper [19] describes how the Gepetto Team's footstep planning strategy is applied within a 3D environment using an occupancy grid. The paper still focuses on flat ground environments with obstacle collision avoidance, however it plans in a full 3D environment rather than projecting the environment to 2.5D like in [10] in order to handle collision detection of the robot's upper body. The paper [20] uses a similar footstep pattern generator, but extends it onto a terrain map model generated from stereo vision. The terrain map is used to inform the controller of foot-terrain contact points. The method was demonstrated in simulation but not applied to their hardware platform.

The Humanoid Robots Lab at Freiburg University has studied biped navigation using the Nao robot as a test platform. They have built a two level environment with a staircase, and have an associated CAD model that matches the environment exactly. Oßwald et al. describe their strategy for autonomous stair climbing in [21]. The work describes the group's Monte Carlo localization methods which leverage the OctoMap software package that was developed by the group [22] [23]. Oßwald provides a summary of plane segmentation algorithms in [9] and describes their approach to stair climbing by localizing against the front edge of individual steps. The robot

alternates between flat ground walking and prescribed step climbing behaviors. The step transition and walking pattern trajectories are designed ahead of time for then known staircase model. In the survey of plane segmentation methods by Oßwald et al., two methods are described and demonstrated with their software. The first method is called scan line grouping which operates on range images [24]. Scan line grouping is a fast image-space algorithm, but it was designed to operate on range images without strong noise, occlusions, or depth outliers. When applied to stereo camera data it can over segment noisy areas, so it is often run with larger distance thresholds that result in it missing smaller regions. The second method (which is preferred by the authors) is an extension of the two point sampling method that was used by Chestnutt in [8]. The two point sampling method is limited to detecting planes oriented along the major coordinate axes of the scanned environment (horizontal ground and stairs, and perpendicular walls), but cannot be applied to fit planes with arbitrary orientation such as ramps. In the evaluation, the plane segmentation is used to localize flat staircase elements which are then fit to their known model, so the orientation limitation is acceptable.

Gutmann et al. have a series of works on biped navigation that use stereo vision to detect edges, planes, and obstacles, which are represented with a 2.5D grid model of the environment [25] [26] [27] [28]. In these works, plane segmentation via the scan line grouping method is used as a preprocessing step to classify terrain types of individual cells of the environment grid model.

Prior work on biped navigation from the Robot Locomotion Group focuses on laser scan matching for localization with a prior map [29], dense terrain reconstruction using stereo cameras and GPU map fusion with truncated signed distance functions whelan2012kintinuous, and footstep planning on convex free space decompositions using mixed-integer quadratic optimization and a heuristic cost function [30]. By integrating the above techniques within an implementation on the Atlas humanoid platform, our work demonstrated autonomous and continuous locomotion over uneven terrain using a simple short horizon re-planning framework [31].

1.2.2 Quadruped Navigation

Although the proposed thesis research is focused on biped applications, there is important related work on quadrupeds and other multi-legged platforms. This work often focuses on rough terrain traversal, since the additional legs provide stability well suited to extreme terrain. Robots for planetary exploration may be required to traverse and climb terrain that requires more contacts points than are used by bipeds (without hands). Bretl describes the free-climbing robot problem and provides hardware validation with the LEMUR robot [32]. The work focuses on the motion planning problem, with particular attention to the frictional contacts required to climb vertical terrain, and proposes a multi-step planning framework to compute footholds and plan posture transitions between them. The work does not consider the terrain perception problem, rather it assumes pre-surveyed terrain contact points are given, nor does it address the state estimation problem to localize the terrain during execution of pre-planned motions.

Hauser et al. study the rough terrain motion planning problem with the six legged robot ATHLETE for extra-terrestrial exploration [33]. The multi-step planning approach is used, selecting footholds first and then using sampling-based techniques to find motion plans to transition between footholds. The work succeeds to find novel motion plans over complex terrain with frictional contact constraints, and is validated in simulation, but it is stated that the planner is too slow for on-the-fly operation, nor is terrain perception considered—the planner operates on synthetically generated height map data.

Krotkov and Simmons describe the software system of the field-tested Ambler robot which carried sensors and computational resources onboard for perception, planning, and control of autonomous navigation over rough terrain [34]. The system integrated a high level task based autonomy system that coordinated the interactions between the perception, planning, and control components. This early work is significant because it highlights the key design considerations of the system integration required for autonomous operation in real world field tests.

Whereas the previously cited works focused on quasi-static stability and foothold transitions, dynamic planning and execution is required to traverse certain terrain features. The Learning Locomotion project motivated new results in dynamic locomotion over rough terrain with the LittleDog quadruped robot. In this project, the terrain model is given so as to remove the perception problem, and external motion tracking cameras are used to remove the global state estimation problem. Byl et al. propose a kinodynamic planning methodology that supports both foothold placements with quasi-static body posture transitions, and dynamic lunges to traverse large terrain features with consideration of ground reaction forces [35]. Shkolnik et al. describe a dynamic motion planning algorithm for bounding trajectories over rough terrain [36]. The authors overcome several challenges of applying RRT motion planners to systems with dynamic constraints by incorporating motion primitives, Reachability Guidance, and task space sampling. The authors provide discussion of the challenges encountered implementing a stabilizing feedback controller to execute pre-planned dynamic trajectories on the LittleDog hardware using offboard computation and motion tracking, and show validation in simulation.

Kolter et al. describe their method for integrated LittleDog terrain perception and footstep planning using a height map representation generated from on-board stereo camera (attached to LitteDog using a mast to raise the camera above the terrain) [37, 38]. Terrain features are extracted from the height map and used to inform a cost map to plan a body path for LittleDog, then a greedy footstep planning algorithm places footsteps to progress along the body path. In this work, holes in the height map due to occlusions are filled using an image based texture synthesis algorithm from the field of computer vision, subject to the constraint that synthesized terrain must not enter line-of-sight free-space. Terrain occlusion filling is an interesting avenue of research, but this work does not attempt to segment terrain, so their evaluation does not assess the benefit of hole filling for segmentation purposes.

1.2.3 Dense Mapping and Fusion

Simultaneous Localization and Mapping (SLAM) is the problem of building a map of an unknown environment while simultaneously localizing the robot within the map [39]. In a 2015 seminar, John Leonard, one of the pioneers of SLAM techniques, presented the following question and answer series to make a point:

> *Q:* What's the most important thing I learned about SLAM *before* 2012?
> *A:* Maintaining *sparsity* in the underlying representation is critical.
>
> *Q:* What's the most important thing I learned about SLAM *since* 2012?
> *A:* Building and maintaining *dense* 3D representations is possible.

Leonard's point about dense representations is in reference to modern dense mapping techniques that consume dense depth inputs (from sensors like the Kinect RGB-D camera) and employ SLAM techniques to fuse the depth data into a dense surface representation of the environment. The Kinect Fusion algorithm introduced techniques that leveraged the high computational capacity of modern GPU hardware to achieve dense mapping in real time, but it was limited to a fixed volume representation [40]. The Kintinuous algorithm introduced a sliding volume map to support environments with unlimited spacial extents [41]. In [31] we have shown that the Kintinuous algorithm performs well with alternate depth sources, in this case a stereo camera rather than RGB-D, and thus is applicable to outdoor environments where Kinect cameras do not perform as well. These fusion algorithms, with hardware acceleration, provide surface estimation of the environment at video rates and are therefore a good source for terrain maps for terrain segmentation and locomotion planning. Terrain maps based on dense camera depth data overcome the limitations of many available LIDAR sensors which are slower to produce point cloud maps at equivalent resolution.

1.2.4 Point Cloud Segmentation

The autonomous biped navigation strategies proposed by the previously cited works each have terrain perception requirements. Terrain perception is a critical component of autonomous and continuous biped navigation for footstep placement and obstacle avoidance. One of the key perception methods used is point cloud segmentation, and specifically plane segmentation of low curvature terrain features. Chestnutt uses plane segmentation on a height map to inform the cost function used by the A* footstep planner [8]. Oßwald uses plane segmentation to localize the robot with respect to a prior staircase model [9]. Gutmann provides an excellent summary of plane segmentation techniques applicable to humanoid navigation in [28], including his improved version of scan line grouping. Gutmann's plane segmentation algorithm runs on raw stereo camera depth images as a preprocessing step to 2.5D grid map building. Segmented planes are filtered so that only horizontal planes are kept, and the grid model supports ground, raised platforms, and obstacles, but not ramps or curved surfaces. Notably, none of the previously cited works have performed footstep planning directly on the plane segmentation results, instead the segmentation results are used for map building and localization, and planning is performed on discretized grid representations of the world.

In the *Leaving Flatland* work by Rusu et al, they describe an integrated system for motion planning in 3D environments that include stairs and ramps [42] [43]. Many of the algorithms and data representations techniques from this work evolved into the Point Cloud Library [44]. In this work, Rusu et al. propose a polygonal representation of the environment and a process to generate the model from a series of raw point clouds. Using an occupancy grid structure, the method uses a local plane fit to each cell and projects inliers to the plane to compute a planar polygonal convex hull. Point clouds are registered to a global grid and local planar polygons are merged based on a polygon-polygon distance function. A geometric heuristic is used to assign a semantic labeling to polygon clusters with categories such as flat ground and stairs. The algorithm relies on several thresholds and parameters. 2D

motion planners are adapted to plan within regions of the polygonal representation, and indoor and outdoor demonstrations are shown with the RHex mobile robot.

Plane segmentation is also used in the preliminary stages of table top object manipulation research. In Rusu et al. 2009 paper [45] the authors describe the table top segmentation algorithm based on RANSAC that has become common place in lab experiments of object manipulation with point clouds from RGB-D sensors. Segmentation methods on point clouds are divided between unorganized point clouds, such as those from laser scans [46] [47] [48], and organized point clouds or range images such as those from RGB-D sensors or stereo cameras. Trevor et al. [49] propose an efficient method for segmentation of organized point clouds, and show how plane segmentation may be used within a general segmentation framework; first clustered planes are removed and remaining non-planar points are segmented using a connected components technique. Strom proposes a combined method wherein laser scan information along with camera images are integrated to perform fast planar segmentation that benefits from both RGB information and 3D features derived from the depth [50]. These segmentation algorithm is applied at room scale to segment planar surfaces and large objects, but not demonstrated at object scale that may be required for manipulation.

Plane segmentation is also useful in surface reconstruction and SLAM techniques. In these methods, surface normal estimation from a point cloud provides important orientation features. Surface normal estimation using least squares regression can vary in quality based on parameters such as local search radius. Sharp corners in the original environment can become rounded in the estimation [44]. Boulch proposes a fast method based on the Randomized Hough Transform that performs surface normal estimation on unorganized point clouds that maintains sharp features [51]. Maintaining sharp features and edges is important for terrain regions like stairs which have limited support area. In [51] the method is tested on point clouds collected at high density with a tripod mounted LIDAR scanner. In our work, we will use RANSAC [52] for surface normal estimation, and apply it to point clouds from fused stereo camera data to extract planar terrain features.

From the fields of computer vision and deep learning, recent results in semantic segmentation have attained state-of-the-art benchmark performance using convolutional neural networks and only RGB camera data [53] [54]. The networks are trained on large, human labeled databases with pixel-level ground truth. The results suggest that convolutional neural networks may be capable of accurate terrain segmentation without using any of the 3D techniques previously cited in this section. It is possible that photo realistic renderings can be used to automatically generate databases with ground truth pixel labels. Perhaps in the future visual simulation will become a new method for training robots to perceive and plan footsteps on uneven terrain.

1.3 A New Approach to Continuous Locomotion

In this thesis research I propose new methods for continuous locomotion over uneven terrain through the combination of online terrain perception and footstep replanning. New challenges arise when the locomotion task is performed continuously and autonomously without a human operator. At the DRC, operators were used to correct errors in terrain perception and to augment footstep planners with expert inputs in the form of navigation goals and footstep placement adjustments. This thesis proposes 3D representations for terrain safe region modeling for footstep planning, and develop novel perception algorithms to generate these representations from 3D inputs. Many prior work examples discussed in the previous sections operate with a 2D height map model, or a flat plane floor model, with obstacles delineated with bounding boxes or marked grid cells. In those models, obstacles are detected and avoided, and possibly stepped over, but not stepped onto. In this work, I propose a 3D representation that supports a broader class of terrain styles and integrate a footstep planner able to step off flat ground and onto terrain that would otherwise be considered an obstacle. This representation is able to encode all features of the environment, such as tables, walls, or doorways, anything that the robot may contact or use for support.

In addition to locomotion, the proposed perception algorithms and environment

representations may have extensions to robotic manipulation. While terrain perception and segmentation techniques find safe regions for the robot to place its feet, similar methods applied at manipulation object scale can find candidate regions for the robot to grasp and place finger contacts. Just like in the locomotion example, better perception will increase autonomy and robustness to allow a robot to continuously estimate and re-plan grasping decisions on newly discovered objects without prior models or restrictive representations. This thesis makes a novel contribution to point cloud segmentation algorithms, and applies the segmentation algorithm to humanoid locomotion over uneven terrain. The contribution is comprised of both the algorithms and the software implementation within a re-usable framework to realize autonomous walking behavior on state-of-the-art humanoid hardware platforms.

Chapter 2

Continuous Humanoid Locomotion Over Uneven Terrain

2.1 Environment and Models

Locomoting robots will encounter a variety of terrain types to traverse. This section describes the environment and model representation that will be targeted by the perception algorithms and walking behaviors presented in this thesis. Motivated by the DARPA Robotics Challenge, we target man-made terrain types that are defined by locally planar regions of varying slopes, such as ramps, stairs, and scattered stepping stones. Figure 2-1 shows the uneven terrain course encountered by humanoid robot competitors at the DARPA Robotics Challenge Finals. The course was composed of concrete block steps with slopes up to 15 degrees and steps as high as 20 centimeters. First we describe the 2D height map representation and its limitations, then describe the 3D representation used in this work.

2.1.1 Height Map

A height map is a 2D grid with uniform spacing. It stores a height value at each grid node. Additional data such surface normals, color, or calibrated camera image coordinates may be stored as well. The height map information is represented in

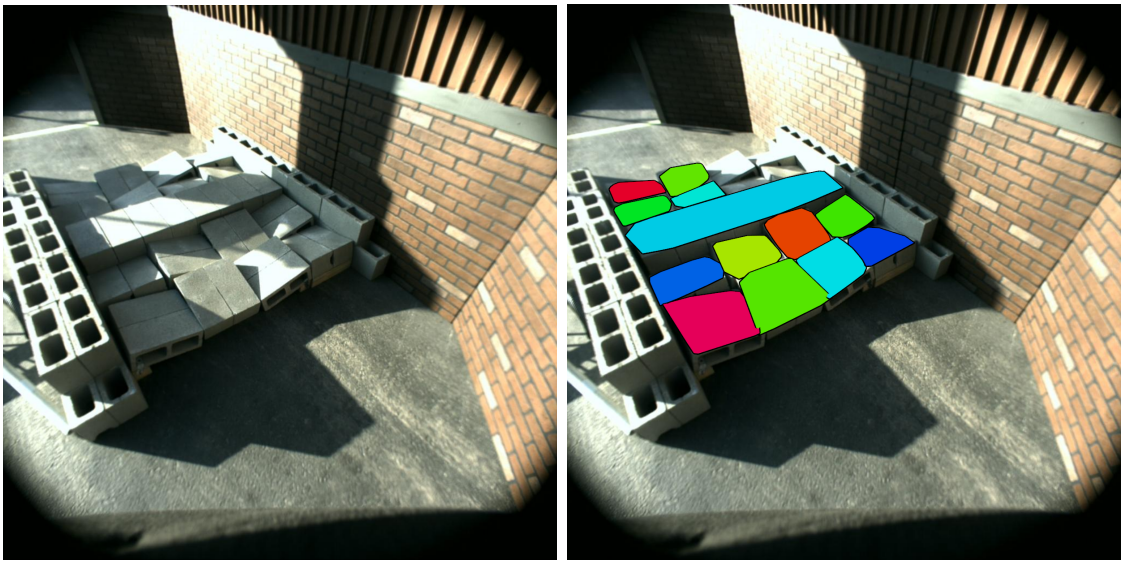


Figure 2-1: Left, the DRC uneven terrain course as viewed from the robot's camera. Right, convex planar regions used for footstep planning are projected into the camera image for visualization.

memory using an image data structure and may be transported using efficient image encoding schemes. Height map data is visualized in 3D as a triangulated mesh, which may be rendered with color camera image textures, or pseudo-colored with height map or surface angle information. The height map normals may be computed using an estimation algorithm, and do not have to agree with the normals defined by the discrete facets of the mesh triangulation.

A height map stores a single height value per node. As a 2D data structure, it is unable to represent multi-height features in the environment such as tables and chairs, vertical walls, toe-kick spaces built into kitchen cabinetry, or overhangs from sloped steps such as found in the DRC terrain and shown in Figure 2-2. When height maps are computed from raw sensor data the height values are filtered using a heuristic algorithm to remove points above a certain height threshold, to remove, for example, the ceiling in an indoor environment. Obstacles such as stairway hand rails can be filtered, or if represented in the height map then down stream consumers should be robust to such extreme outliers in surface representation.

The discretization of height map grids can pose additional problems when the terrain is not axis aligned with the grid axes. As shown in Figure 2-2, the rotated

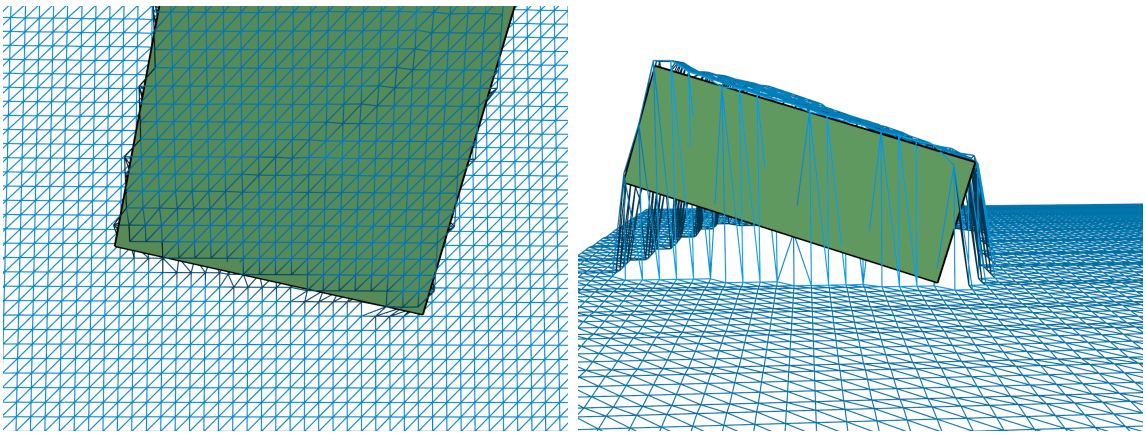


Figure 2-2: Aggregating LIDAR data into a height map can result in loss of 3D information of important terrain features. The green block is ground truth terrain geometry, the blue grid is the height map representation. Left, edges and corners that are not axis-aligned with the height map grid suffer from discretization error. Right, multi-height terrain features such as overhangs cannot be represented using height maps.

terrain block feature produces jagged artifacts along the block edge, making it impossible to represent sharp edge features which are important areas for footstep planners to reason about.

2.1.2 Unorganized Point Cloud

An unorganized point cloud is a list of 3D points defined with respect to some reference frame. In this work, point clouds are always defined with respect to the local, or world coordinate frame, and not relative to the robot or sensor, except when noted. As opposed to an organized point cloud, nearest neighbor information between points is not stored. In practice, algorithms use a spatial search tree to find neighboring points within some radius of a reference point. LIDAR data fused over time from multiple sensor sweeps is often represented using this data structure. Additionally, data from other sources such as depth images or height maps (both are organized sources) can be efficiently converted to unorganized point clouds without loss of spatial data. The reverse is not true, converting an unorganized point cloud to a depth image or height map requires a projection, where unique points in the unorganized cloud may map to the same cell within the image. Thus, unorganized point clouds retain the most

information about terrain features, and are also the most general form of input to a segmentation algorithm, at the cost of leaving out nearest neighbor information, requiring the consuming algorithm to select its own strategy in order to operate on point neighborhoods.

Although unorganized point clouds are a sparse representation of the environment, as opposed to the dense representation implied by the connectivity information of organized point clouds or height map grids, in practice the point clouds produced by LIDAR scanners typically have a high sample density in the regions of interest (subject to the geometry of the scanner sweep), and actually incur a reduction in sample density when projected into height map representations of reasonable grid spacing (0.5 to 1.0 centimeters). Unless otherwise noted, the term point cloud appearing in this document will always refer to unorganized point clouds.

2.1.3 Planar Polygon Regions

We will represent terrain regions using planar polygons embedded in three-space. The polygon is a list of 3D vertices that all lie in the same plane. An alternative representation is an polygon coordinate frame, defined with respect to the local coordinate frame, and a representation of the vertices within the coordinate frame where the 3rd component of each vertex is zero. Figure 2-1 shows an example of the DRC Finals terrain course with planar polygon regions superimposed. These regions are the result of our terrain segmentation algorithm applied to an unorganized point cloud LIDAR scan of the terrain. This representation is able to capture planar terrain features exactly without discretization errors associated with height maps, including sharp edges, non-axis aligned boundaries, vertical walls, and multi-height features such as floor and table, or overlapping ladder steps.

2.1.4 Limitations

Modeling the environment with planar polygons incurs some limitations. In practice, it is a good model for most type of terrain encountered by a robot roaming in man-

made environments. It is not well suited to deformable terrains, such as grass, mud, or snow, or surfaces with high curvature or changing slope, or obstacles such as small diameter pipes (large diameter pipes or low curvature surfaces might be suitably represented with planar approximation, however). The planar representation must select a scale parameter at which the environment should be approximated. For example, a scale could be selected so that a gravel surface is approximated with a single planar region, but a staircase is segmented into multiple planes instead of represented as a single planar ramp. Depending on the scaling parameter, surfaces with high frequency voids such as metal grating may or may not be representable as a safe region for planning supports. The planar polygon representation does not answer the question of convex representation. The proposed segmentation algorithm generates region boundaries that may be non-convex, and we discuss some strategies for convex representation.

2.2 Sensors

The Atlas robot's sensor head, the Carnegie Robotics MultiSense SL (Figure. 2-3), is equipped with a pair of high quality global-shutter cameras with wide FOV lenses, and also includes an embedded Field-Programmable Gate Array (FPGA) that implements the Semi Global Matching stereo disparity estimation algorithm [55]. This hardware allows the sensor to produce rectified RGB-D images on-board at frame rate (15-30 Hz) with low latency (∼90 msec) without impacting the robot's computational load. While device is the highest resolution stereo camera commercially available [56], it produces normals which are unstable from frame to frame — thus requiring fusion to be useful for the purpose of accurate terrain segmentation. Figure 2-5 shows the typical depth quality achieved by the sensor under outdoor lighting conditions with strong sun and shadows.

The MultiSense has a FOV of 90°-by-90° which observes a significant ground footprint in front of the robot while also allowing the background scene to be used for visual odometry. The sensor had also contains a revolving Hokuyo laser rangefinder

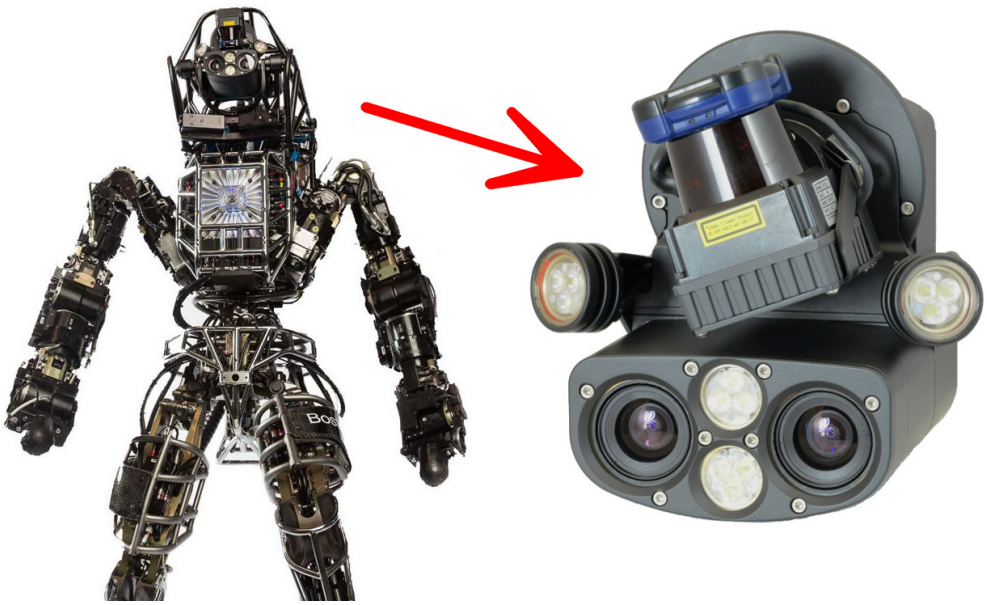


Figure 2-3: The Atlas robot's primary sensing is provided by the Carnegie Robotics MultiSense SL sensor head which is equipped with a stereo camera and a rotating LIDAR scanner. (photo credits: Boston Dynamics and Carnegie Robotics)

which generates range samples at 43,000 points per second. The rangefinder revolves axially around a spindle at a user defined rate, where the speed of revolution affects the sample density within point clouds generated from fused scans. In our work, we rotate the sensor head at 5 rotations per minute, such that a new point cloud is collected every six seconds (each half sweep covers the entire world). Scan lines are accumulated together and fused over time using an interpolated pose estimation of the sensor head. Figure 2-4 shows an example of the point cloud collected during a single sweep of the scanner. Whereas the LIDAR point cloud updates at 0.17 Hz, the stereo point cloud updates at 15 hz, with comparable resolution but greater noise and restricted field of view.

2.2.1 Flying Points Filtering

Scan data from the Hokuyo laser rangefinder of the MultiSense contains outlier points called *flying points*. These are intermediate range values that fall between background and foreground objects as a result of glancing reflections when the laser scans across the boundary of the foreground object. Since flying points occur in areas of dis-

Figure 2-4: Visualization of terrain sensed by the MultiSense LIDAR scanner and camera. Left, the LIDAR point cloud is projected into the 2D camera image to visualize the registration quality between the LIDAR and camera sensors. The LIDAR has been segmented into ground and scene, colored green and magenta. Right, another visualization of the camera-to-LIDAR registration, the LIDAR point cloud is colorized using RGB samples from the camera image.

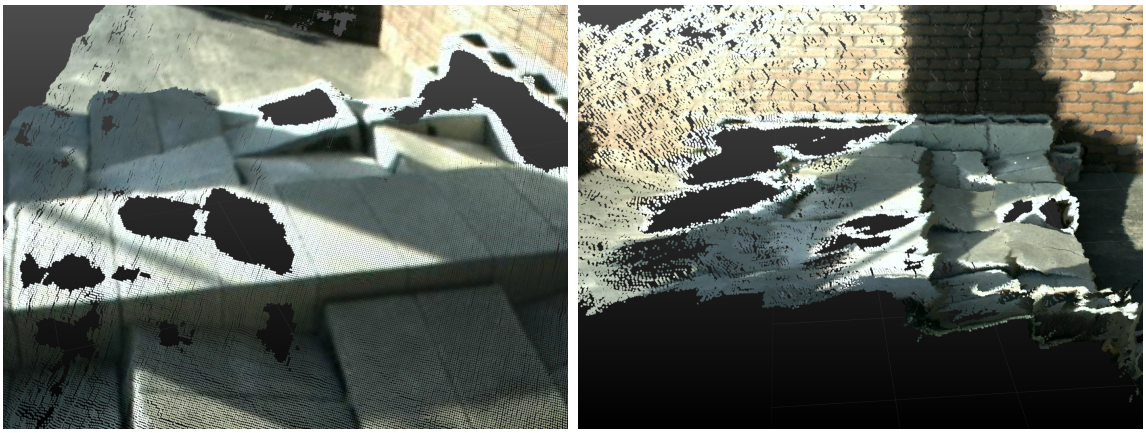


Figure 2-5: Visualization of stereo depth map from the MultiSense. Left, over exposed regions of the camera image result in holes in the disparity image computed on-board the MultiSense (the holes are not the result of occlusions). Right, the stereo disparity image projected into a 3D point cloud and viewed from a direction perpendicular to the sensor view direction. In this view the noise characteristics are apparent—edge boundaries of the terrain features are wavy in appearance due to details of disparity estimation algorithm.

continuous range measurement, a typical method to reject these outliers is to use thresholding on range deltas between neighboring point returns. Equation 2.1 defines a standard filter that appears in prior work [57]. The filter identifies a set of outlier flying points P_f that should be removed from a scan line. A point x_n corresponding to a laser range measurement within a single scan line is included in the set if the distance between x_n and it's neighbors x_{n-1} and x_{n+1} is greater than a threshold value T_f.

$$P_f = \{\ x_n \mid max(\|x_n - x_{n-1}\|, \|x_n - x_{n+1}\|) > T_f\} \tag{2.1}$$

In this equation, the threshold value must be determined empirically. Unfortunately, the appropriate value is not easy to determine as it depends on the distance and the local surface orientation of the object causing the flying points, with respect to the scanner. We propose a rejection method that is instead based on the angle between the scanner view direction and the line segment connecting outlier points with their scan line neighbors. This rejection method is based on the observation that flying points always fall in line with the view direction of the laser ray. This angle based filter is defined by Equation 2.2, where $\angle$ is the function that returns the angle between two vectors.

$$P_f = \{\ x_n \mid \angle(x_n - x_{n-1}, x_n) < A_f \wedge \angle(x_{n+1} - x_n, x_n) < A_f\} \tag{2.2}$$

In this equation, the points are expressed in the scanner coordinate frame, so the position of point x_n is equal to the scanner view direction when the range measurement for x_n is collected. Figures 2-6 and 2-7 show the before and after results of flying points filtering. Two type of outliers in need of filtering are flying points generated from self observations of the robots arms and legs, and flying points spanning boundaries between objects such as uneven terrain blocks. Without outlier rejection, the concrete blocks appear to have ramps connecting the discontinuous surfaces.

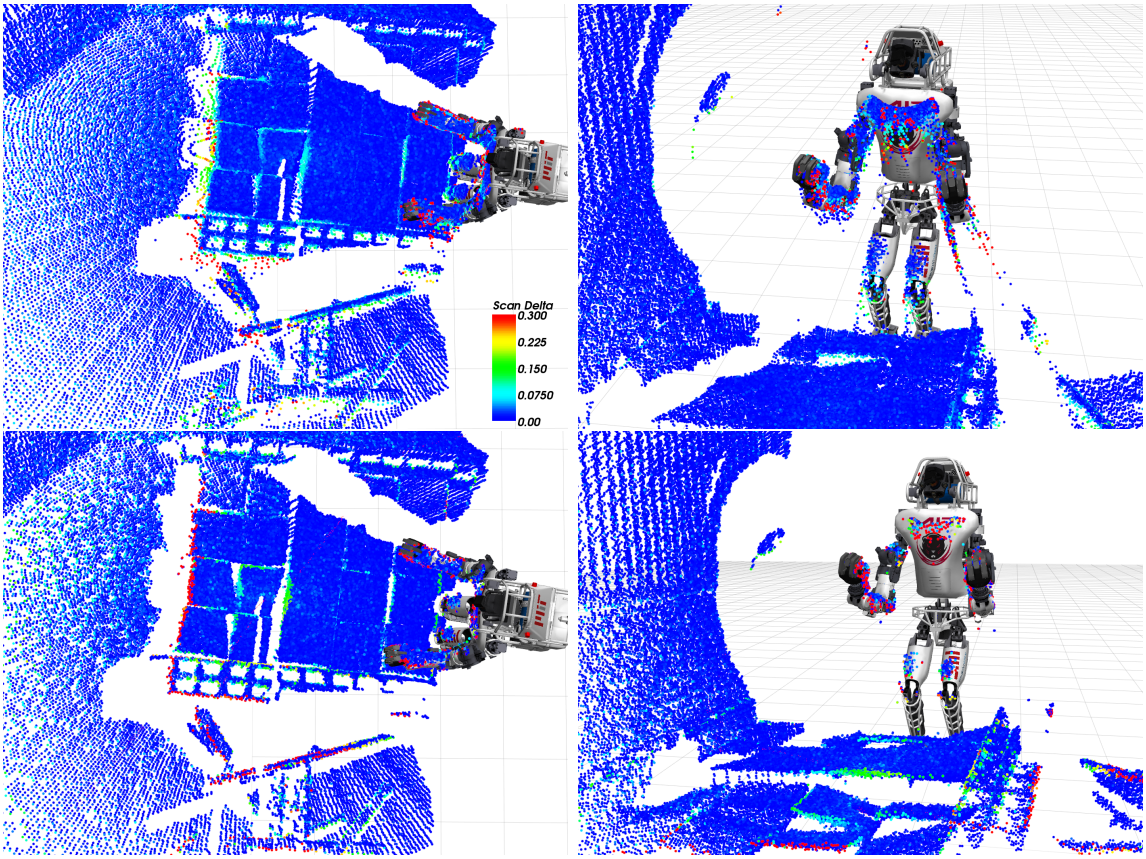


Figure 2-6: LIDAR flying points are outliers. Top left, outliers connect the surfaces of the terrain steps. Top right, outliers cause spurious range measurements surrounding the robot. The bottom images show the result of filtering using Equation 2.2. LIDAR is colored by range delta, the range distance in meters between neighboring returns.

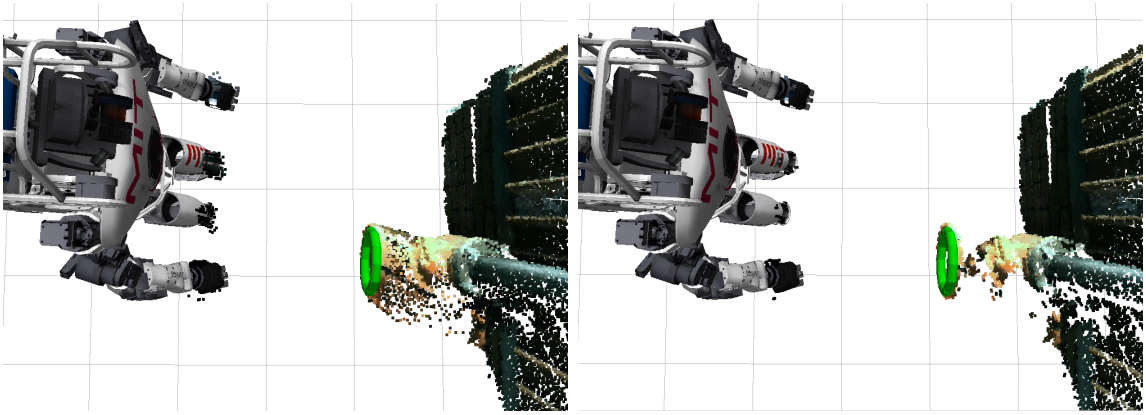


Figure 2-7: A colorized LIDAR point cloud of the valve task at the DRC Finals. Left, the original point cloud before flying points filtering. Right, the result of filtering. The green torus is a valve model fit to the point cloud. The filtered point cloud was input to a perception fitting algorithm to estimate the pose of the valve. Flying points are especially bad at the boundaries of objects with polished surfaces like the valve.

2.2.2 Stereo Depth Filtering

For accurate terrain segmentation on stereo depth maps, it is important that the quality of the maps produced by the vision pipeline be comparable to LIDAR. Visual aliasing, caused by poor image texture and repeated non-distinctive patterns, commonly causes spurious outlier regions within the disparity image and consequently incorrect 3D depth values. An example of the data is shown in Figures 2-5 and 2-8.

As a result it is crucial to correctly filter and fuse the stereo data before attempting segmentation and footstep planning. We first apply a de-noising filter on each frame that (1) labels connected components in the disparity image, where connectedness is determined for adjacent pixels by similarity of depth values; and (2) removes components having a number of pixels below a threshold size, in our case 4,000 pixels. This has the effect of suppressing small isolated disparity regions and pixels that disagree with their neighbors. The result of filtering these outlier regions is shown in Figure 2-8. The algorithm runs quickly enough to keep up with the stereo camera frame rate, and run time performance is listed in Table 2.1. Figure 2-5 shows the noise characteristic of the filtered, but unfused depth map. In the next step we will use fusion to smooth the depth map and reconstruct an estimate of the terrain surface.

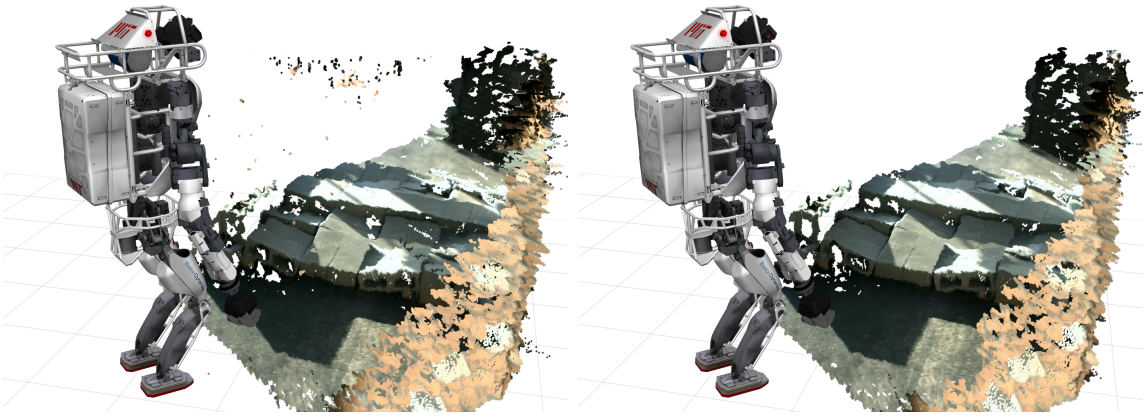


Figure 2-8: Stereo depth data before and after filtering small connected components. The filtered outliers are removed from the void space in front of the robot.

2.3 Fusion and Mapping

Dense stereo correspondence, or matching pixels across calibrated camera pairs in order to infer triangulated distance values, is a well-studied problem in computer vision research; the Middlebury Benchmark System [58], for instance, provides a detailed performance comparison of over 150 algorithms. However, the use of dense stereo in real-time robotics applications — particularly at high frame rates — has been limited. This is mainly due to difficult trade-offs between overcoming inherent algorithmic challenges (properly handling object boundaries and occlusions, disambiguating repeated structures that cause visual aliasing, estimating accurate depth values in regions of low visual texture) and realizing computationally efficient, low-latency implementations suitable for robotic platforms.

2.3.1 Active RGB-D Mapping

Active sensors such as the Microsoft Kinect have spurred new interest in dense visual mapping because they directly address several of these challenges. These sensors provide color (RGB) images registered with dense, centimeter-accurate depth (D) images at video rates using an infrared pattern projector and camera pair. All computation is performed on-board using specialized hardware, and devices are available at commodity prices. As a result, active sensors have quickly become the de-facto standard and have been adopted for a wide range of indoor robotics applications.

Shortly after the release of Microsoft's sensor, the KinectFusion system [59] demonstrated real-time RGB-D data fusion within a volumetric data structure (the Truncated Signed Distance Function, or TSDF) maintained in GPU memory. A two-step process of (1) camera-to-map tracking followed by (2) update of the TSDF via parallel ray casting produces highly accurate dense reconstructions of small 4–6m volumes at centimeter resolution. The Kintinuous algorithm [60] was subsequently developed to accommodate larger-scale exploration for mobile robots. It builds upon KinectFusion to enable mapping of extended environments in real-time, without being limited to a region of fixed volume.

While active RGB-D sensors address many shortcomings of passive stereo, they still have practical limitations. The Kinect has a narrow field of view and short range of 4m, but in particular the on-board cameras have rolling shutters and utilize active illumination. This produces blurred or distorted images while moving and cannot be used outdoors due to interference by sunlight.

2.3.2 3D Fusion

In this work, we adopt the MultiSense as a surrogate for active sensors to generate input data for depth fusion algorithms. Our previous publication [61] describes the details of how this is accomplished.

Each volumetric grid cell within the Kintinuous TSDF contains a signed distance to the nearest surface, and a probability of occupancy which is a measure of confidence. This probability updates as more observations of a surface are made and converges as we become confident of surface's location. Retrieving the full contents of the TSDF volume from the GPU by copying it to main memory of the CPU is too expensive to run at camera framerate, so our approach instead is to generate a depth image from the camera's viewpoint using an efficient GPU ray casting operation to find zero-crossings of the TSDF. The resulting image has the same perspective as the original input stereo data (though it is clipped by the TSDF volume boundary), and contains the improved surface estimation of the Kintinuous fusion algorithm. The depth image is converted to an unorganized point cloud for input to the terrain segmentation algorithm.

The fusion helps to overcome the noise and depth outliers present in the MultiSense disparity images, however it has a tendency to over-smooth sharp features of the terrain. Nevertheless, in this work we show that the fused depth image is of sufficient quality to be used in place of LIDAR scans as input to the terrain segmentation algorithm, producing footstep planning regions to support humanoid locomotion in hardware demonstrations that will be presented in Section 2.7. This exciting result suggests that stereo fusion, leveraging passive and lower cost sensors, is a suitable alternative to the more accurate, lower noise, but slower and expensive LIDAR sen-

sors, and we hope that the faster update rate of the stereo vision approach will enable faster and more dynamic locomotion capabilities in our robots over uneven terrain where accurate footstep placement is a must.

2.4 Terrain Segmentation

The proposed terrain segmentation algorithm applies planar region growing to an unorganized point cloud. It is well suited to model terrain features with sharp edges in the presence of noisy LIDAR data or smoothed fused stereo data. The input to the algorithm is pre-filtered to remove some classes of outliers, as described in Sections 2.2.1 and 2.2.2, but may still contain outliers, so the algorithm uses RANSAC estimation techniques for robustness. The algorithm is fully agnostic as to whether the input data comes from LIDAR or fused stereo 3D reconstruction. Analysis of the performance differences between the two data sources is found in Section 2.7. Algorithm 1 describes the major steps of the process, which will be described in three parts, 1) surface normal estimation and filtering, 2) planar region segmentation, and 3) polygonal shape fitting and sorting. Several of these steps require scale parameters that affect the results. The text provides the parameters that work well in practice, and we note that the parameters are all selected based on a scale that works well for segmenting terrain features suitable for foot placement of a human sized foot.

Algorithm 1 Terrain segmentation algorithm

```
1:  function TerrainSegmentation(p)
2:                                                  ▷ Given point cloud p
3:      scene ← RemoveGroundPoints(p)
4:      scene ← SurfaceNormalEstimation(scene)
5:      planarPoints ← FilterByNormal(scene)
6:      clusters ← PlanarRegionGrowing(planarPoints)
7:      regions ← EmptyList()
8:      for each c in clusters do
9:          regions ← Append(regions, ComputePlanarPolygon(c))
10:     return regions
```

2.4.1 Surface Normal Estimation and Filtering

The first step of the segmentation pipeline is to filter all points that represent the floor (ground plane). This step is not strictly necessary but improves performance for environments that do contain a large planar ground surface.

Voxel Gridding

We apply a voxel grid downsampling filter to enforce an upper bound sample density in the point cloud. This ensures that downstream processing completes with consistent performance and does not depend on the density of the LIDAR point cloud input. The filter outputs an unorganized point cloud where each point is the centroid of the points in the input point cloud that fall within for each cell of the voxel grid. We select a grid size value of 0.5 centimeters which affords enough resolution for accurate boundary segmentation of the terrain features.

Ground Removal

The ground is defined to be the lowest visible surface in the point cloud, which is estimated using the 5th percentile of the height coordinates, z_5 (rather than the minimum height, to exclude outliers), then use RANSAC model fitting [52] of a plane within a search region that is limited to points in the band $z_5 \pm \epsilon$ where ϵ is the half height of the search region. A value of 5 centimeters works well in practice, as it is several times larger than the sensor noise. A larger value must be used for sloped ground, or without concern for performance the ground fitting can be run without any search region.

Robust Surface Normal Estimation

Next, we perform surface normal estimation using a robust method that employs RANSAC plane fitting to recover estimates that retain sharp corner features. Figure 2-9 compares the result of surface normal estimation using a least squares fit versus RANSAC. Surface normal estimation is computed for each point using a local

neighborhood of radius r centered at the point. Neighbor points are located using a spatial search over the input point cloud, implemented with the fast k-d tree of the PCL and FLANN libraries [44, 62]. Using a least squares fit plane to the neighborhood results in reconstructions with rounded corners that do not agree with the real world terrain. The RANSAC algorithm instead chooses a random subset of two neighbors to compute a proposed plane along with the query point, then checks how many points in the neighborhood agree with the proposal within ϵ distance of the plane. This is repeated n times, and the proposal with the greatest neighbor agreement is selected. Finally the surface normal is computed from the least squares fitting plane to the agreeing neighbors. Since point cloud density is controlled, and therefore the number of neighbors has a fixed upper bound, the overhead associated with RANSAC estimation is constant in the worst case. The calculation is not prohibitive and could be efficiently parallelizable, though we leave this opportunity for future work. The run time performance listed in Table 2.1 is for serial computation.

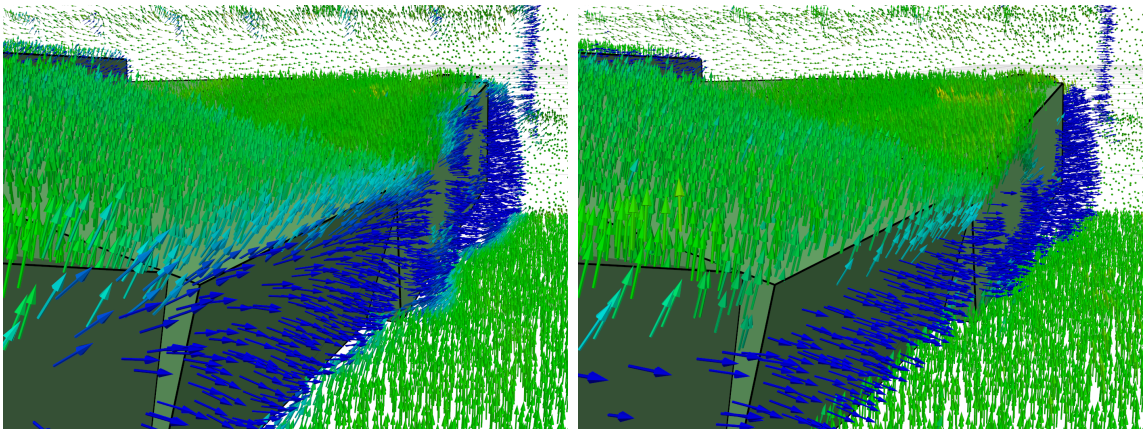


Figure 2-9: Oriented arrow glyphs are used to visualize surface normals computed by two different estimation algorithms. The green blocks represent ground truth terrain geometry. Left, surface normals are estimated using a least-squares plane fit to all points within a select neighborhood, resulting in rounded off corners. Right, RANSAC modeling is used to reject a subset of neighborhood points prior to computing the least-square plane fit, allowing sharp edge reconstruction.

Filtering By Surface Normal

The points are filtered to keep only those within α degrees of horizontal according to surface normals. Steeper regions are deemed infeasible for footsteps and therefore can be filtered. The value of α may depend on the walking controller performance and friction characteristics of the robot foot and support surface. We have selected 25 degrees in practice. This step may be excluded in order to search for all planar regions within the point cloud, to segment vertical walls, for example. Since we did not consider steep support surfaces, it was most efficient to filter the points at this stage before proceeding with planar segmentation.

2.4.2 Planar Region Segmentation

The proposed planar region segmentation algorithm is based on a region growing technique. Planar regions are defined by point cloud neighborhoods where the point positions and surface normal estimates are all within an error distance to the plane, and collectively define the plane. Determining the plane is an iterative process, and the plane estimate is updated as neighboring points are added to the set, as defined by Equation 2.3.

$$R_i = \{\ p_n \mid dist(p_n, P_i) < \epsilon_d \wedge angle(p_n, P_i) < \epsilon_a\} \tag{2.3}$$

R_i is a planar region, defined by the set of point ids that belong to the region for plane estimate P_i. Each point p_n is added to the set if it is within ϵ_d distance perpendicular to the plane estimate P_i and the surface normal at p_n is within ϵ_a degrees of the plane estimate normal.

The region growing method uses a spatial search to locate nearest neighbors, it does not depend on prior computation of mesh connectivity, and is therefore not sensitive to small gaps due to occlusions. Like the surface normal estimation, we use a fixed search radius r, but select a smaller value for region growing than for surface normal estimation. In the case of region growing, the search radius sets the maximum Euclidean distance gap that may be bridged between two otherwise

disconnected regions. This value is determined empirically for the terrain, and we have selected 5 centimeters.

Since the neighboring point candidates are tested for inclusion using their local surface normal estimate, the region growing is robust to outliers and smoothing effects at corners. This allows the regions to expand to the true boundary of sharp terrain features. The pseudo-code algorithm for the region growing search is presented in Algorithm 2.

Algorithm 2 Planar region segmentation algorithm

```
1:  procedure PlanarRegionSegmentation(cloud, r, ε_d, ε_a)
2:
3:      regionMap ← NewRegionMap()
4:
5:      for each p in cloud do
6:
7:          if p in regionMap then
8:              continue
9:
10:         regionId ← MaxId(regionMap) + 1
11:         planeEstimate ← InitPlaneEstimate(p)
12:         processQueue ← NeighborsSortedByDistance(cloud, p, r)
13:
14:         for each p_n in processQueue do
15:             if TestCandidate(p_n, planeEstimate, ε_d, ε_a) then
16:                 planeEstimate ← UpdateEstimate(planeEstimate, p_n)
17:                 regionMap ← AssignRegion(regionMap, p_n, regionId)
18:                 neighbors ← NeighborsSortedByDistance(p_n, cloud, r))
19:                 processQueue ← Append(processQueue, neighbors)
20:
21:     return regionMap
```

The planar region segmentation function returns a label mapping that assigns points in the input point cloud to their respective planar region id. Figure 2-10 visualizes the mapping result of the segmentation algorithm applied to the DRC Finals terrain course. The point cloud is drawn using a random color assignment to highlight unique region ids.

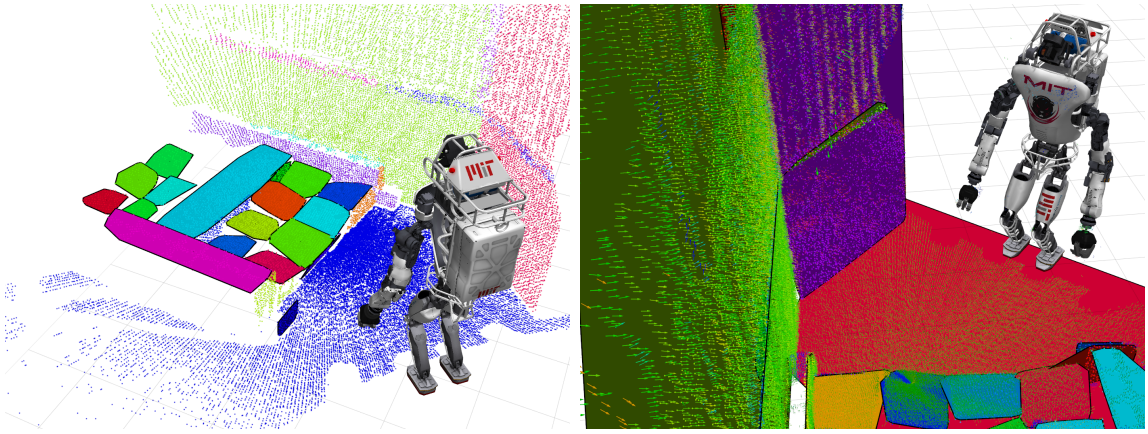


Figure 2-10: Segmentation results of DRC Finals terrain course using LIDAR data. Left, planar polygon convex hulls are drawn for region segmentations computed after ground removal and filtering by normal to keep only only regions with slope less than 25 degrees. Right, region growing has been applied to the entire point cloud without pre-filtering, producing segmentations of vertical walls and other regions that should not be candidates for footstep planning, but could be used for other planning purposes.

2.4.3 Polygon Shape Fitting and Sorting

The region mapping from the prior segmentation step is used to compute planar polygon shapes in three-space. The shapes will be processed into convex regions for input to the footstep planner. An example of the regions represented with convex hulls is shown in Figure 2-10. For each region, the points are written with respect to the coordinate frame of the local plane estimate of the region. These points can now be converted to a set of 2D points by zeroing the third component (a projection to the region plane). Finally, we compute the 2D convex hull of the projected points which is used to fit the minimum-area bounding rectangle using the calipers algorithm. Rather than use the convex hull directly, we used this bounding rectangle to represent the terrain in our experiments and at the DRC Finals, since it filled in portions of the edge areas that were missed during region growing due to imperfect surface normal estimation at edges, despite the vast improvement already gained from robust normal estimation algorithm.

The convex hull and bounding rectangles are converted back to their 3D representation for visualization. We define a standard orientation for the rectangle shapes by

using the robot's stance frame as a reference frame. The stance frame is the average position and orientation of each foot frame, where the X axis is forward and Z is up. The coordinate frame of the rectangle is selected with X axis pointing along the rectangle edge most closely matching the stance frame X axis (by angle projected into the XY plane), and positioned at the center of the rectangle.

In some cases, segmented regions may not be accurately represented by the convex hull of the points, such as an L shaped step. Section 4.1.1 discusses how we can use concave hulls instead to represent the region outline and use convex decomposition to produce several planning regions for the original planar segmentation. In the experimental results presented below we did not attempt to perform convex decomposition, instead limited our environment to terrain built of convex features.

2.5 Footstep Re-Planning

The footstep planning problem typically involves choosing an ordered set of foot positions for the robot, subject to constraints on the relative displacement between those footsteps, in order to bring the robot close to some desired goal pose. Typically, these footstep locations must be chosen in order to avoid some set of obstacles in the environment. Performing a smooth optimization of footstep poses while avoiding obstacles tends to introduce non-convex constraints which can make globally-optimal solutions extremely difficult to find [63].

A common approach among footstep planners is to discretize the set of possible foot transitions, expressed as the relative displacement from one foot pose to the next. From this set of discrete actions, a tree of possible footstep plans can be constructed and searched using existing search algorithms such as A*, D*, and RRT [64, 65, 66, 67, 68]. These techniques, however, are limited by the discretization of the footstep actions, since a small number of actions severely limits the robot's possible footstep plans, while a large number of actions creates an extremely large search space. Choosing an informative and admissible heuristic for footstep planning problems can be very difficult [69].

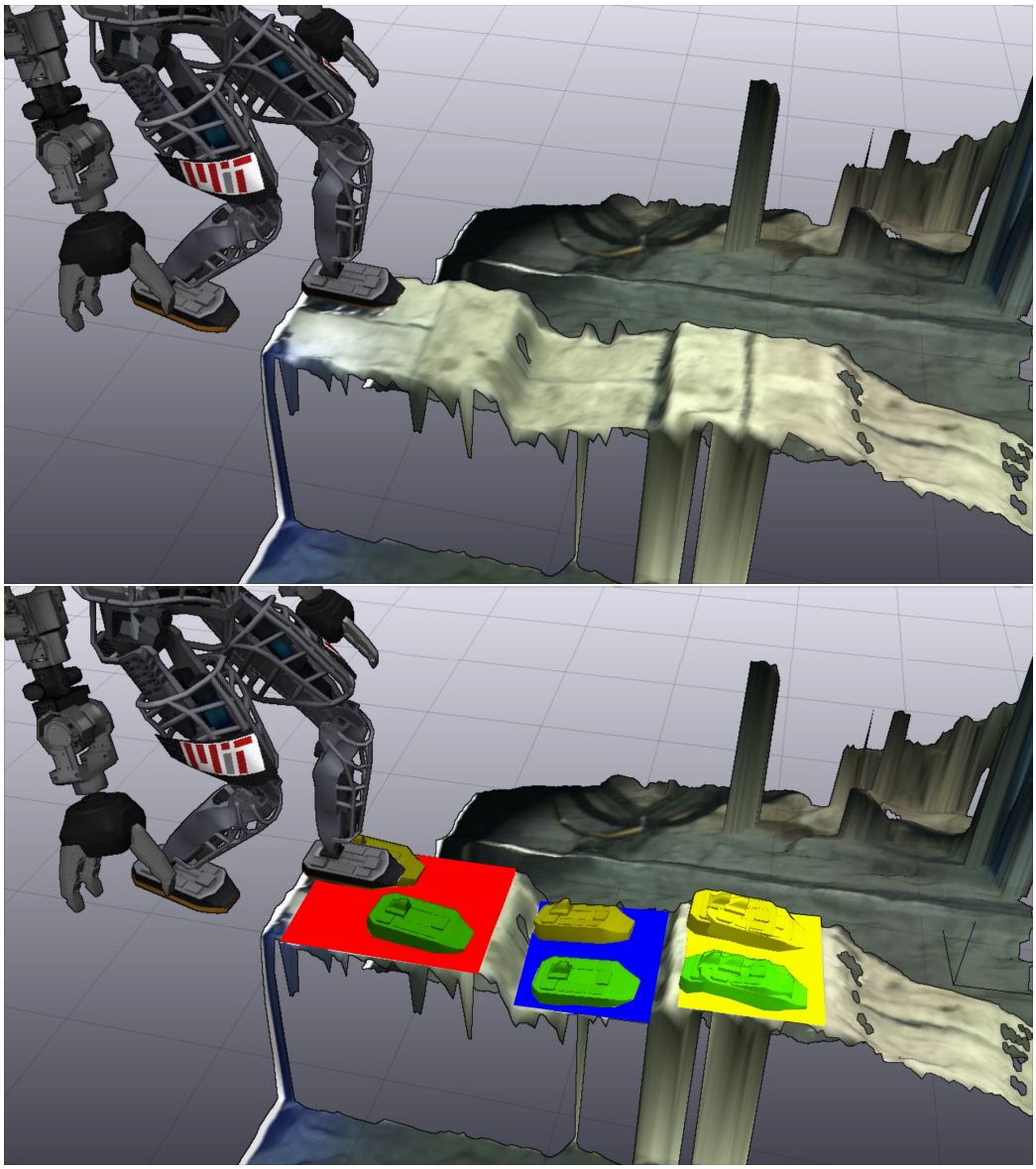


Figure 2-11: As the robot walks over the terrain, its visual mapping system builds a dense reconstruction of the environment ahead. Once per step, the planning system captures the reconstruction, detects convex regions which are large enough to contain a foot and determines suitable footstep placements to be executed by the walking controller. Note that self observations of the robot by itself are filtered and that footsteps are rotated to match the local terrain normal.

Recent work by Deits and Tedrake reversed the problem of obstacle avoidance into one of assigning footsteps to some pre-computed safe regions [70]. If the regions of safe terrain are convex, then the requirement that a footstep remain within some safe region is a convex constraint. A problem consisting of convex constraints (and a convex objective function) can typically be solved to its global optimum extremely efficiently [71]. Combining such a problem with a discrete choice of the assignment of footsteps to safe regions results in a mixed-integer convex program with high worst-case complexity but which can often be solved to global optimality efficiently with modern tools and techniques [72, 73].

We choose to use the mixed-integer convex optimization described by Deits and Tedrake in [70] and available from the Drake toolbox [74], to quickly produce optimal footstep plans for an environment which has been decomposed into convex regions of safe terrain. Rather than operating directly on the point cloud data provided by the LIDAR or stereo system, the footstep planner requires only a description of each safe region as a planar area in 3D. Thus, when planning footsteps, the perception algorithms described in the previous sections can be abstracted away into a tool which produces regions of safe terrain. These regions, along with a desired navigation goal pose, are used as input to the footstep planning optimization, which chooses the number of footsteps to take and the poses of those footsteps. Note that the planner itself decides online how many footsteps to place in each convex region — which varies with the size of the region, the robot configuration, and layout of the upcoming terrain regions.

In prior work, the IRIS segmentation algorithm has been used to compute collision free configuration space regions on discretized height map data [63]. However, IRIS suffered from under segmentation, where some collision free space was not included in the planning regions, and operated on discretized height map data that could not accurately represent edges and corners frequently found in uneven terrain such as stairs and the DRC concrete block terrain course, as discussed in Section 2.1.1. In this work, we have applied a 3D point cloud segmentation algorithm to compute footstep planning regions. Operating directly on the 3D representation overcomes

these limitations and results in high quality regions that more accurately align with the terrain features and include coverage up to the edge and corner boundaries.

2.6 Autonomous Execution

We designed a software interface called Director capable of managing the autonomous execution of the terrain traversal task. Director, described in Chapter 3, provides an asynchronous task execution framework which coordinates behaviors requiring integration between sensor drivers, perception algorithms, and the planning and control subsystems.

The continuous walking behavior that was demonstrated in our experiments described in Section 2.7 is detailed with pseudo-code in Algorithm 3. The behavior applies our point cloud segmentation algorithm to find footstep regions and uses a *carrot-on-a-stick* steering approach to generate navigation goals that lead the robot over the terrain course. Short-horizon footstep plans within the segmented safe regions are computed and sent to the walking controller for execution.

Algorithm 3 Continuous walking algorithm

```
1:  procedure OnFootLiftoff(q, f_step)
2:                                  ▷ Given footstep frame f_step, or nil to bootstrap
3:      if f_step then
4:          q_next ← LandingConfiguration(q, f_step)
5:      else
6:          q_next ← q
7:      p ← PointcloudSnapshot()
8:      r_safe ← TerrainSegmentation(p, q)
9:      if empty(r_safe) then
10:         ContinueQueuedFootstepPlan()
11:     else
12:         f_goal ← NavigationGoal(q, r_safe)
13:         f_plan ← FootstepPlan(q_next, r_safe, f_goal)
14:         if IsValidPlan(f_plan) then
15:             QueueFootstepPlan(f_plan)
16:         else
17:             ContinueQueuedFootstepPlan()
```

The routine *OnFootLiftoff* is called once to begin walking, and again each time

a foot lifts off from the ground during walking. The first stage computes the robot configuration q that will be used as input to the footstep planner. Because the robot is walking continuously and planning on-line, the configuration q used for planning is chosen to be the next double foot support of the robot after the current swing foot has landed, i.e., the configuration the robot is expected to achieve in the very near future upon completion of the current walking step. The configuration q is a list of the robot's joint positions in generalized coordinates plus the 6 degree of freedom position and orientation of the robot's base link in the world coordinate system. The function *LandingConfiguration* uses inverse kinematics to find the next double support configuration using the current stance foot and the next target footstep f_{step}.

Next, the *NavigationGoal* returns a navigation goal 1 meter forward along the steering direction. The segmented regions and navigation goal are passed to the footstep planner to compute a footstep plan to navigate towards the goal. The resulting footstep plan is immediately queued for execution. In typical operation, only the first footstep of a plan is executed because the remainder of the plan is overwritten at the next online re-planning stage.

In this work we focused on navigation over uneven stepped terrains, we do not present results showing exploration over flat terrain with protruding obstacles, or longer distance global path planning. In the case of the former, the approach would be to populate the flat terrain with a spanning set of walkable regions (as discussed in [63]).

2.7 Experiment Results

To demonstrate the described capability we progressively developed the various components of this system with more challenging experiments and more general terrain layouts. Each experiment was carried out in a repeatable manner.

The robot was set up in front of a terrain of uneven concrete blocks and instructed to progress towards a goal, as described in Section 2.6, and repeatedly did so until it reached the end of the course with no flat surfaces and stopped. We did not implement

a general purpose exploration strategy as we focused on the perception and footstep planning problems. As mentioned in Section 2.6, footstep execution was carried out by the manufacturer's stepping controller. The robot autonomously walked over the entire course in 240 seconds for a total of 25 steps and 14 rows of blocks.

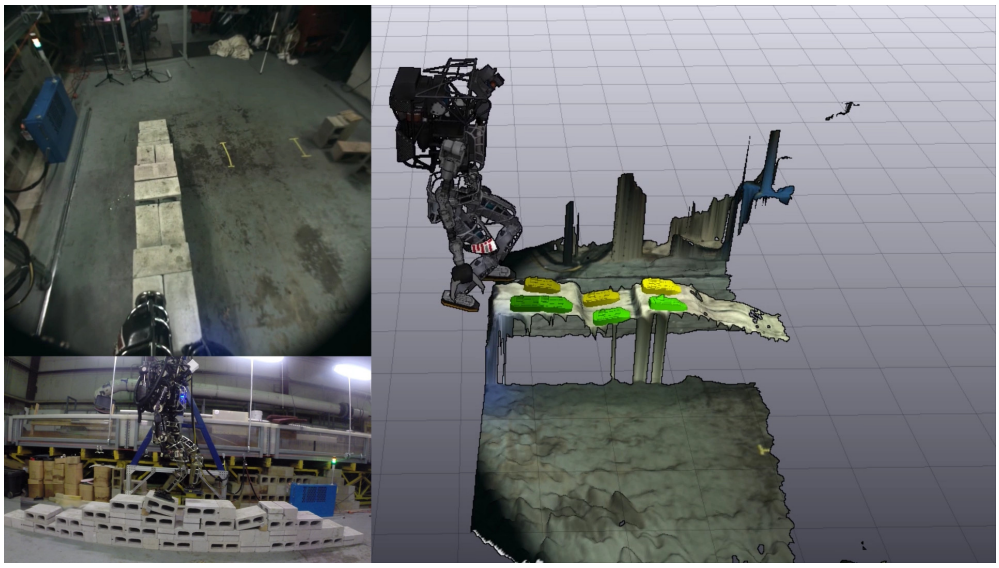


Figure 2-12: Visualization of the robot continuously walking over a complex terrain course. The upper left figure shows the camera view from the robot's sensor head. The lower left figure shows a side view of the experiment. The robot's actuated LIDAR sensor is covered up by a white box to demonstrate that only vision is used here. The main figure shows a rendering of the robot's configuration while mid-step and the placements of the next seven steps on the terrain map.

Our computation was provided by two identical off-board desktops each with a 3.30GHz Intel i7 CPU and an Nvidia GeForce GTX 680 GPU. The computation time of each step of the processing pipeline is shown in Table 2.1

Table 2.1: Algorithm component execution times.

Component	Average Time (msec)
Image acquisition	105
Stereo pre-filtering	40
Kintinuous stereo fusion	110
Planar region segmentation	615
Footstep planning	445

Note that each of the modules operated asynchronously on different threads. In particular, the segmentation and planning chain is time critical as there is a 4 sec period (the foot swing cycle) where a new footstep plan can be accepted by the controller.

In our experiment, we constructed a terrain course containing a climb, uneven and tilted steps, cracks, and gaps which is shown in Figure 2-12. We conducted experiments where only the LIDAR sensor was used as point cloud input, as well as runs where the point clouds were sourced from fused stereo depth maps.

This experiment demonstrated that the segmentation algorithm and footstep planning could support all these terrain complexities and it also dynamically selected the number of steps to place within each region depending on the configuration. In the latter part of the course, when the terrain steps descend, the downward steps only became visible just as the robot approached and was expected to step down onto them. This was a particular challenge to both the stereo fusion algorithm and LIDAR. For the LIDAR, the steps only became unoccluded in the very last sweep before segmentation was necessary, and for the stereo, the fusion algorithm requires several updates to fully resolve the surface reconstruction in the TSDF.

An analysis of the region segmentation point to plane error of the DRC Finals terrain course LIDAR point cloud is shown in Figure 2-13. The histogram of point distance to plane error is an illustration of the range measurement noise of the LIDAR collected under these conditions. The histogram of surface normal estimation error shows that the majority of point normals are within ± 3 degrees of the region plane normal, but there is a long tail of points with normal angle deviations up to the cut-off threshold of the region growing algorithm. Notable in the figure (marked with a red arrow), rendered in the pseudo-color image of surface normal estimation error, the region at the center of the scanner view direction contains a more points with surface normal estimation error greater than 3 degrees. We believe this is because of aliasing effects of overlapping scan lines concentrated in this region at the center of the MultiSense spindle axis. The aliasing of the overlapping scan lines affects the performance of the surface normal estimation, but the effect is not present in the

point distance to plane error plot.

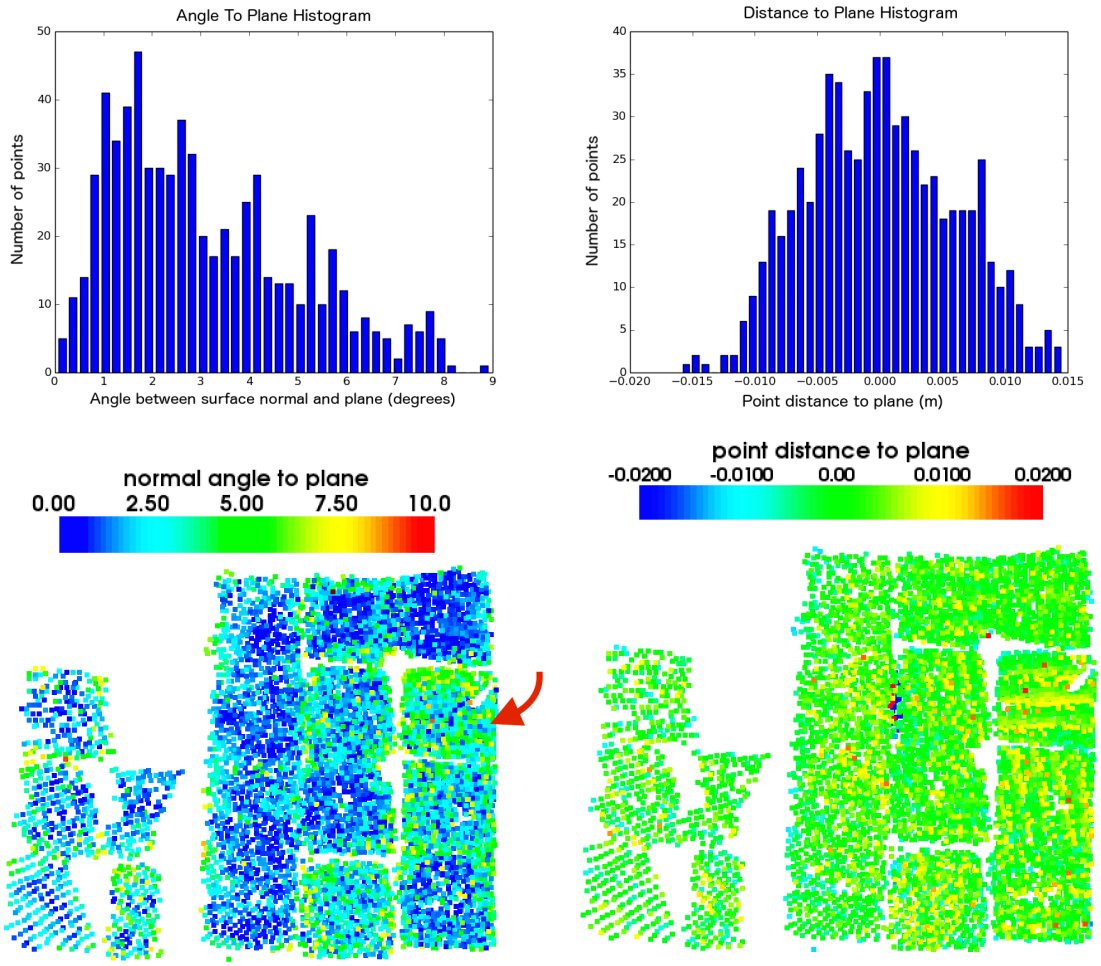


Figure 2-13: Top row, histogram of point cloud attributes within a single segmented region, in this case, the first block of the robot's planned walking path. Top left, the histogram displays the variation in angle between estimated surface normal for each point and the normal of the least-squares fit plane to the points. Top right, the histogram displays the point distance to the least-squares fit plane. Bottom left, pseudo-color visualization of the angle between estimated surface normals and the plane fit normal. Bottom right, pseudo-color visualization of the point distance to plane.

2.8 Conclusion

In this work we have described a terrain segmentation algorithm that operates directly on a 3D representation of the environment, overcoming discretization errors

associated with 2D height maps, and demonstrated the effectiveness of the segmentation approach when combined with on-line footstep planning to enable continuous humanoid locomotion over challenging uneven terrain. We presented results from a hardware experiment where the Atlas robot autonomously traversed a 5.5 meter terrain course requiring up to 30 individual footsteps. The segmentation technique was also deployed by Team MIT at the DARPA Robotics Challenge finals, enabling our Atlas robot to successfully traverse the uneven terrain and stairs course on both days of competition.

With our results, we have also shown how real-time passive stereo fusion can be used as a direct replacement for an actuated LIDAR sensor, and that it produces comparable results in challenging lighting conditions. We anticipate the responsiveness and greater resolution of this type visual reconstruction may be required for humanoids to move at human walking speeds in the future.

It is clear that the optimal footstep placement (i.e. those chosen by a human during human walking) uses subtle information beyond terrain geometry alone — such as hanging footsteps over step edges and reasoning about occlusions when stepping down onto partially observed terrain. In future work we aim to add such features to our system as well as to develop more general navigation strategies.

Chapter 3

Director: Robot Interface Framework

3.1 Introduction

The previous chapter introduced a terrain perception algorithm and demonstrated its useful application in a fully autonomous walking task on the Atlas robot. The autonomous behavior depends on the coordination of a large robotic system architecture that incorporates control, state estimation, perception, planning, sensors and drivers, and network communication. This chapter will present Director, a novel robot interface framework and user interface that we developed to be the high level interface to these components within our robot system architecture. Director provides a programming interface in the C++ and Python languages, a user interface with advanced visualization capabilities, and a task execution runtime. The user interface enables operators to interact with the robot through sliding levels of autonomy, varying from teleoperation modes with no autonomy, to a supervisory role where the operator monitors status and progress of fully autonomous behaviors.

While fully autonomous operation is of critical importance to many robotic applications, there are also applications where it is useful and feasible to incorporate human operators into the robot mission to help ensure robust and safe execution in challenging and unpredictable environments. Operating a high degree of freedom mobile manipulator, such as a humanoid, in a field scenario requires constant situational awareness, capable perception modules, and effective mechanisms for interactive mo-

tion planning and control. A well-designed operator interface presents the operator with enough context to quickly carry out a mission and the flexibility to handle unforeseen operating scenarios robustly. By contrast, an unintuitive user interface can increase the risk of catastrophic operator error by overwhelming the user with unnecessary information. With these principles in mind, we present the philosophy and design decisions behind *Director*—the open-source user interface developed by Team MIT to pilot the Atlas robot in the DARPA Robotics Challenge (DRC). At the heart of Director is an integrated task execution system that specifies sequences of actions needed to achieve a substantive task, such as drilling a wall or climbing a staircase. These task sequences, specified *a priori*, make queries to automated perception and planning algorithms, such as those described in Chapter 2, online with outputs that can be reviewed by the operator and executed by our whole-body controller. Our use of Director at the DRC resulted in efficient high-level task operation while being fully competitive with approaches focusing on teleoperation by highly-trained operators.

The Director interface was designed with the concept of shared autonomy. The design goal was to develop a system capable of completing tasks autonomously, but always be able to fall back to manual operation mode to allow a human to complete part of the task. Additionally, the autonomous behavior should be organized so that it is possible to resume the autonomous mode as soon as possible after a period of manual operation. Entering manual mode should not require the operator to complete the whole task; there should be many options to return control back to the autonomous system. This defined our notion of shared autonomy within the context of the DRC competition: a task execution system that accepts input from both automated perception and planning algorithms as well as human operator inputs. In our shared autonomy design, human inputs range from high-level supervision to low-level teleoperation. The operator can provide input to the perception system by verifying or adjusting the result of a perception algorithm, or providing a seed, such as a search region, to steer the perception algorithm. For task autonomy, the human can provide input through supervision of subtask execution. For example, the operator can approve a motion plan before it is executed by the robot, or pause

automatic execution of a subtask (due to some knowledge about the situation that is not available to the robot) and complete the task manually using teleoperation. At the lower levels of teleoperation, the operator can control Cartesian poses of end-effectors, a wide range of kinematic constraints, or individual joint positions.

Sections 3.2–3.4 describe the interface elements and design decisions that comprise the Director user interface and it's interactive perception algorithms, as well as the programming interface. Sections 3.5 and 3.6 provide analysis of the effectiveness of these methods in the context of the DRC competition tasks. Material from these sections also appears in our publication currently under review [75]. That publication also includes details of the task execution system and shared autonomy concepts within Director. Director interacts with our wider architecture which has been described in a previous publication [76].

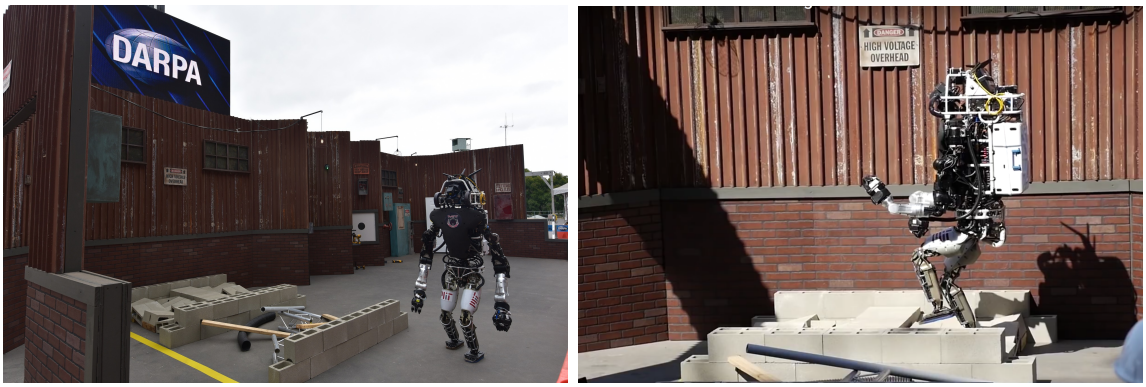


Figure 3-1: MIT Atlas robot at the DRC Finals. Pictured left, this photo was taken on the rehearsal day. It was the first time MIT Atlas walked outdoors without a safety belay and using battery power. Right, MIT Atlas walks the terrain course on the first day of competition. The right arm hangs limp, damaged in a fall during the vehicle egress task.

3.2 User Interface Design

Director is the primary graphical user interface (GUI) that was used to operate the robot in competition, and to test and develop robot capabilities in our laboratory. It is the central location from which the operator initiates all commands to the robot

including startup and calibration procedures, manipulation and walking plan queries, and high-level task operation.

Director was almost entirely developed between the DRC Trials (December, 2013) and the DRC Finals (June, 2015). It replaced the *DRC-trials* user interface described in [76] (Figure 14), whereas the remainder of the architecture described therein was largely retained.

The *DRC-trials* interface was originally developed for MIT's entry in the DARPA Urban Challenge in 2007. Given the nature of that challenge, it was primarily intended as a tool to observe the status of an autonomous robot and to visualize the results of automated perception and planning algorithms. For this reason, it was fundamentally designed as a *viewer* and not as a tool to support on-line interaction. While it was re-engineered to allow a user to operate the Atlas robot for the DRC Trials, it was inefficient and difficult to use for the following reasons:

- Individual modules (such as LIDAR visualization, footstep planning, reach planning etc) were implemented as separate plug-ins within an isolated memory space. This enabled independent development but meant that coordination of the modules wasn't possible, for example, hiding the LIDAR point cloud automatically when examining a prospective motion plan.

- Each module implemented its own interaction elements within a single taskbar. This required the user to expand and then scroll through the taskbar to find the button which requested a particular sensor feed or to change the speed of plan execution. This was both inefficient and unintuitive.

- Rendering was implemented using low level OpenGL commands which did not coordinate across the modules. As mentioned in Section 3.3.3 the combined scene graph approach of the Director gave the operator complete control over the active visual elements.

- The *DRC-trials* interface lacked either task sequencing or a state machine. While the operator could place reaching goals and request a motion plans to

them, these actions were not tailored to the action, for example using specific joint speeds or motion planning constraints for turning a door handle or moving an arm in free space.

- Finally, the *DRC-trials* user interface was implemented in C and C++. Using a low-level language made development slow and prone to runtime crashes. By contrast, several team members could use our higher level Python–based task sequencing to prepare a DRC Finals task (as shown in Figure 3-3) without needing to understand how the Director interface was assembled into an application.

These design limitations directly motivated the development of the Director, which we believe allowed us to more quickly develop semi-autonomous behaviors and was more specifically tailored to the task of operating a humanoid robot — which in turn allowed us to more efficiently execute the DRC tasks.

Director is comprised mainly of two user interface windows: the task panel (Figure 3-2), and the application main window (Figure 3-3). During competition runs, the task panel is the primary interface through which the operator and robot share responsibilities to complete the tasks. The operator supervises task execution through the task panel, but may use the manual interfaces of the main window if the need arises to make corrections, for example, the operator may need to adjust automated perception fitting results or plan and execute through teleoperation.

3.2.1 Director main window

The main application window of the Director is pictured in Figure 3-2. The main window contains a 3D visualization environment to draw the robot's current state, perception sensor data, motion plans, and hardware driver status. Embedded panels are available to interface with hardware drivers for the sensors, grippers, and monitor overall health status of the system state. The interface also provides a teleoperation interface to support manual control by the operator.

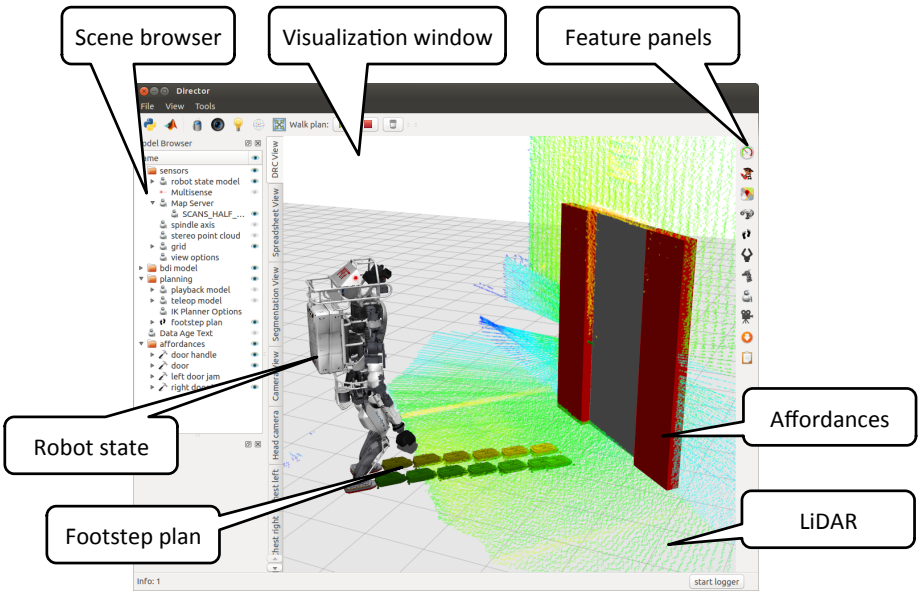


Figure 3-2: The Director main user interface. The main window contains a 3D visualization environment to draw the robot's current state, perception sensor data, motion plans, and hardware driver status. Feature panels are opened from the right side task bar.

Figure 3-3: The task panel interface. In the competition, the task panel was the primary interface used to supervise autonomous behaviors and provide inputs to the guided perception system. As long as there are no failures in execution, the task panel occupied the complete attention of the primary operator.

3.2.2 Visualization

A 3D visualization window is the central widget of the main window, and occupies the majority of the screen real estate. The visualization system is built using the Visualization Toolkit (VTK), an object-oriented scientific visualization library with data filtering and interaction capabilities [77]. Key to the success of our visualization system is its ability to be scripted at a high-level to easily enable users to add new capabilities and prototype algorithms. Algorithm prototyping is supported by a rich visual debugging system capable of drawing primitive shapes, meshes, frames, images, point clouds, text overlays, etc. Through the use of a Python interface to the Point Cloud Library, we are able to prototype and debug new point cloud processing algorithms with seamless integration in the 3D visualization environment [78].

Director also visualizes multiple renderings of the robot model. Separate instances of the model are used to display the state estimate, the interactive teleoperation configuration, and to animate manipulation plans. In Figure 3-2, we see a snapshot of a DRC Finals run: the robot stands with the door in front of the robot, the point cloud is drawn and an affordance model of the door has been fitted. The interface also shows a candidate walking plan returned from the planner at the request of the autonomy system. The operator has the ability to manually adjust footstep plans in the 3D window using interactive widgets. Similar widgets are used to adjust the 3D pose of affordance models and geometric constraints in the teleoperation interface.

3.2.3 Feature panels

A core motivation for the development of our user interface was clarity and minimalism over exposing the operator to needless detail. Where possible, panels are opened only when the user actively needs them and otherwise closed to maximum screen real estate for the visualization window. A vertical toolbar is docked on the right edge of the screen that provides buttons to activate context specific panels. By organizing features in single panels that fit the screen without scroll bars, the user learns to expect interface element positions and may reach them with a single mouse click.

3.2.4 Teleoperation interface

One of the most frequently used feature panels is the *teleop panel*. Together with the visualization window, the teleoperation panel (shown in Figure 3-4) provides the operator with a rich set of controls to design whole-body manipulation poses and trajectories following from a set of geometric constraints. Through this interface, the operator can constrain the position and/or orientation of the robot's hands, feet, and pelvis and the angles of individual joints. Our custom inverse kinematics solver also allowed the operator to express quasi-static stability constraints by requiring that the robot's center of mass remain within the support polygon of one or more of the robot's feet [76]. Together, the kinematic and quasi-static constraints allowed the operator to describe complex whole-body motions with changing contact states through the teleoperation interface.

3.2.5 Task panel

The task panel window is shown in Figure 3-3. The task panel contains a tabbed widget, with each tab holding the task plan for one of the eight tasks in the competition. In the competition, the task panel occupied the full screen of one of the primary operator's monitors and the main Director window occupied a second. As long as there are no failures in execution, the task panel occupied the complete attention of the primary operator. The task panel is a visual representation of the shared autonomy system, it steps through the hierarchy of tasks and asks for inputs from the operator as required to complete the tasks. If something fails, for example, the door fails to unlatch after turning the handle, the task sequence is paused (in some cases, through automatic failure detection, and in other cases by operator intervention) and the operator may switch focus to the main window to manually operate the robot back to a state where the autonomy system is capable of resuming control. The task panel interface was designed so that any individual on our team could be capable of operating the robot to complete tasks. When the system asks for an input from the operator the requested input is clearly stated and designed so that it is not sensitive

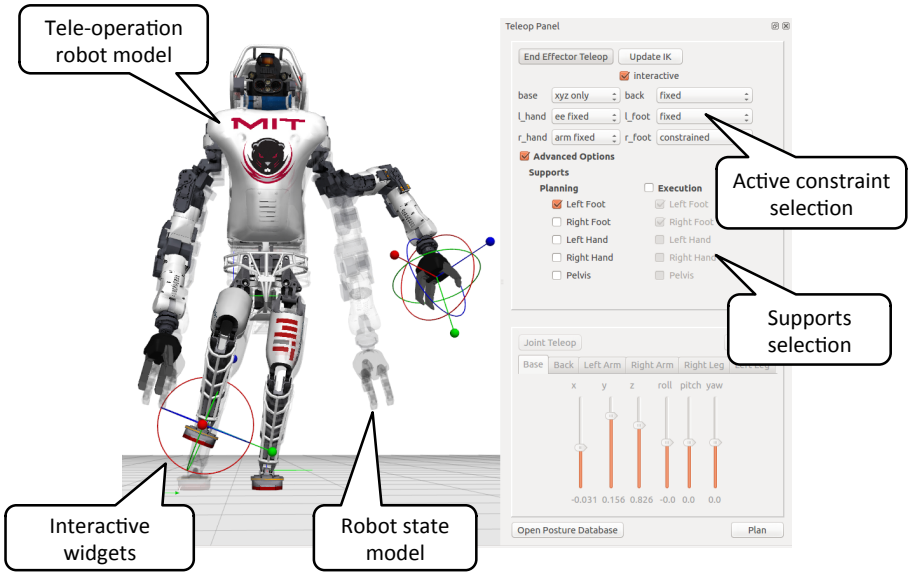


Figure 3-4: The teleoperation interface provides the user with a rich set of preset constraints that are adjusted with widgets in the visualization window. The robot's current configuration is shown as a translucent model, and its desired configuration is shown as the full opaque model. In this example, the operator has indicated constraints on the position and orientation of the robot's left hand and right foot. The operator has also indicated that the robot's final posture should keep its center of mass near the center of the robot's left foot, ensuring that the robot will be able to stably balance on just that foot.

to the variation in possible responses from the operator. For example, a skilled operator is not required to interact with the perception system. An operator need only click on a camera image anywhere on the door in order to convey the required information to a door fitting algorithm. However, if a task fails and manual intervention is required, we found that this operational mode required user training and knowledge of the specific task flows in order to manually operate the robot around the failure and to continue autonomous operation.

The task panel interface has tabs across the top of the panel as pictured in Figure 3-3 to switch between DRC competition tasks. The text area at the bottom of the window displays simple status messages when each task starts and completes. If a task fails for any reason, the failure is displayed in red text so that the operator may quickly locate a description of the detected failure. Above the text area, the task panel is divided into three columns. The central column is the main focus: it contains a tree list of the task hierarchy. In the figure, a user prompt task is active, so it displays the message *Please approve the footstep plan* and displays accept and reject buttons. The interface includes controls to pause, continue, or step through the task sequence. The user can either step through the task sequence conservatively or allow continuous task execution. Task parameters (in the left column) were used to alter task behaviors, for example, to switch hands used in manipulation, or to select a grasping strategy. The right column contains a list of button macros that can be used to execute some steps of the task manually. In normal operation the task parameters and button macros are not needed, but may be used by the operator during edge cases when the planned task sequence is not compatible with the new unforeseen situation. Section 3.3.5 describes the programming model underlying the task panel user interface.

3.3 Programming Interface Design

3.3.1 Software stack

The Director framework is a combination of C++ and Python code. Python, being a high-level scripting language, is the top layer in the software stack where most high level functionality is implemented. C++ is an important language for interfacing to robotics libraries, implementing fast algorithms that loop over large data, performing high rate communication, or run multi-threaded coded.

The software stack is diagramed in Figure 3-5. The C++ layer of Director ties together several advanced scientific computing and robotics libraries, such as Drake, PCL, and VTK, to design components with simplified interfaces for scripting by Python [74, 44, 77]. A key design of the Director software stack is the interface between C++ and Python components. We have selected two core C++ library dependencies, Qt and VTK to build the foundation of our framework. Qt is a cross-platform widget library for building user interfaces, and VTK is a scientific computing library for data processing, I/O, and rendering. Both libraries come with full Python bindings, meaning that their complete C++ API is available to the Python programming language, and objects or data in memory can be accessed from C++ and Python, and passed between layers without overhead. Additionally, C++ classes in the Director source code that inherit from VTK or Qt base classes automatically get Python bindings. This allows developers to work seamlessly between languages without writing specialized interfaces. Recently, SWIG has been used to wrap portions of the Drake library for Python, allowing the Director Python layer to script Drake objects directly without using helper interfaces in the Director C++ code.

3.3.2 Scripting

The user interface embeds a Python programming environment with an interactive console interface pictured in Figure 3-6. Every action that can be performed in the user interface can also be commanded programmatically from the Python console

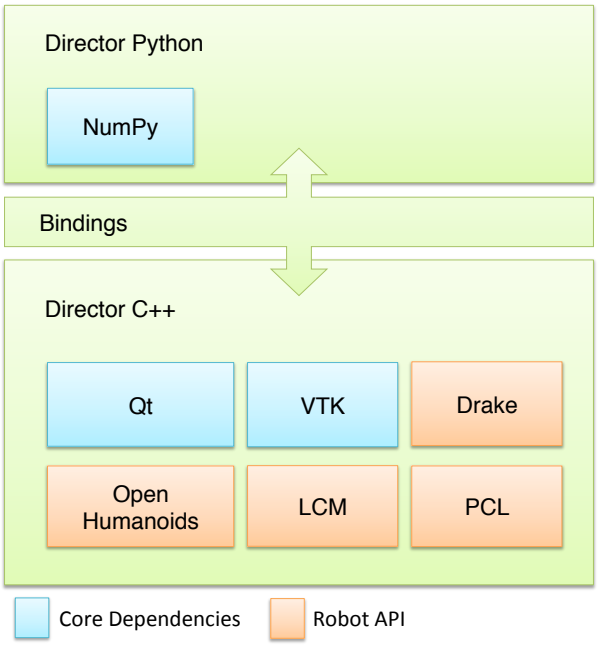


Figure 3-5: The Director software stack includes layers in Python and C++. The core libraries build the application framework, and the robot API pulls in functionality from libraries like Drake, OpenHumanoids, LCM, and PCL. This is a simplified diagram, the actually software has many other optional dependencies, for example, OpenCV for computer vision, Bullet for collision detection, or OctoMap for point cloud occupancy mapping.

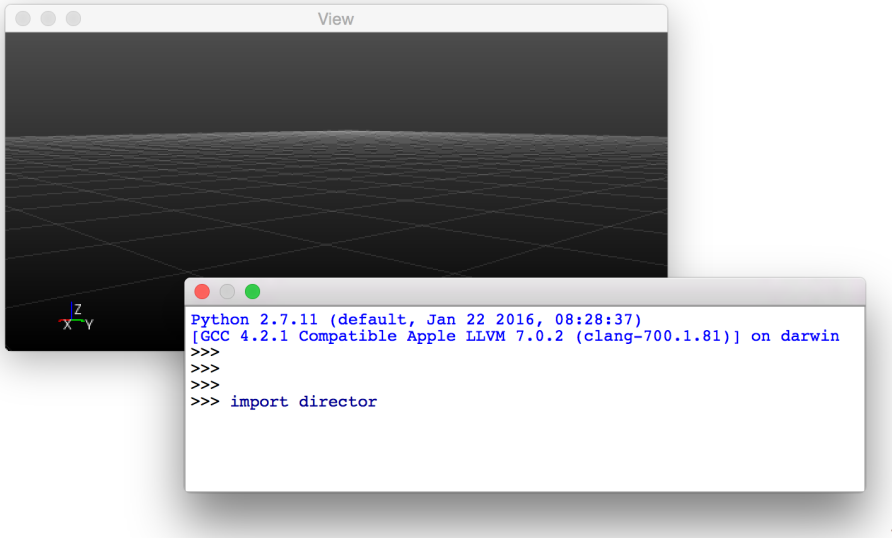


Figure 3-6: The Python console provides an interactive programming environment embedded within the Director user interface.

or from a Python script. For example, the user interface panel available to the operator for requesting walking plans is implemented with a programmatic interface so that Python calls can make the same high-level interface to query the walking planner and execute the resulting plan, or collect point cloud data and invoke an affordance fitting algorithm. In addition to accessing the user interface elements from the Python environment, the user and programmer also has access to data objects stored in the Director scene graph: sensor data such as images and point clouds, robot poses and trajectories, affordance models, and frames and transforms, may be read and manipulated from the Python interface. This was a key design that allowed us to design our shared autonomy tasks with high-level specifications, and allowed members of the team without comprehensive expertise in the lower level APIs to write task routines that interfaced with the full robot system capabilities.

3.3.3 Affordance model

When a physical object was to be interacted with, it was denoted to be an *affordance*, which combined a 3D model of the object with metadata describing the modes of interaction with it. One such set of metadata is a set of named reference frames relative to the body frame of the object. The combination of geometry and annotations of the form of reference frames are used as input to build constraints for motion planning queries called from the task execution system. This affordance model proved a natural way to structure our system's interaction with objects in the world, although our library of affordances is limited to objects that are relevant specifically to the DRC competition (door handle, valve and drill for example), and a set of basic geometric shapes (box, cylinder, prism, and sphere for instance). Our approach to the generalization of affordances was to implement a segmentation routine that would cluster objects in the point cloud and return the convex hull for each one of them. Each convex hull is an instance of the affordance model and therefore treated as such for planning purposes.

3.3.4 Object model

The left dock of the main UI window contains the *scene browser* panel and *properties panel*. These are visual representations of the underlying object model within the application. The object model groups items in named collections using a tree hierarchy which takes inspiration from the concept of a *scene graph* which is common in 3D visual rendering applications. The object model stores a variety of object types, including robot models, affordances, coordinate frames, point clouds, terrain maps, motion plans, and footstep plans. Each object's interface is presented in the form of modifiable properties. Properties can be edited by the user, or automatically by a task execution. Crucially, the object model was used as a data store for tasks to pass data through the execution pipeline, and to present data to the user for approval or adjustments.

3.3.5 Tasks

Subtasks required to execute a complete DRC task were implemented with callable Python objects and functions. Many subtasks were parameterized and reusable, while others were customized for purposes targeted toward specific tasks. As an example, the following is a typical sequence of task executions and the subtasks used: *fit drill, approve drill, plan reach, approve manipulation plan, execute plan, wait for execution, close gripper.* The name *plan reach* in this example refers to an affordance specific planning function—a function that plans a reaching motion to bring the end-effector to a grasping location around the drill affordance. A typical planning task was parametrized to include the reaching side (left, right), and the name of the target affordance i.e. the drill. The task calls subroutines that construct the required constraints based on the coordinate frames of the drill affordance, and then query the manipulation planner. The task waits for a response from the manipulation planner or information about a failure (for example, if the required drill affordance cannot be found or if the planner failed to find a feasible solution).

The *approve drill* and *approve manipulation plan* tasks are examples of user

prompt tasks. The user prompt task presents a message to the user along with options to accept or reject. The task waits for a decision from the user and either completes successfully or raises an exception to pause execution of the task queue. User prompts give the user the opportunity to adjust an input to the system without having to actively intervene to pause execution. During the *approve drill* user prompt, the user can adjust the drill affordance pose if required. The adjusted pose will then be used in the subsequent planning task. The *execute plan* task publishes the manipulation plan on the committed plan channel which will be transmitted to the robot. This task completes immediately. The next task, *wait for execution* monitors the execution of the task by the controller until execution is complete. The last task in this example, *close gripper*, sends a small message that is received by the gripper driver. This task is also parametrized by side (left, right) and the type of grip required.

3.4 Perception for search and rescue robots

We adopted a guided perception approach to model fitting from point clouds and images. Fitting algorithms estimate the 3D pose of objects of interest in the environment which are represented using affordance models described in Section 3.3.3. Operators can provide guidance to the perception system by annotating search regions for the point cloud fitting algorithms. The operator can define annotations by clicking on displayed camera images, or by clicking on 3D positions in a point cloud. For the competition we preferred annotations on 2D camera images because it required less precision than point cloud annotations. For instance, during the valve task, we were required to fit a valve affordance to the valve in the point cloud. As shown in Figure 3-7, the operator reduces the search space by indicating the region where the valve is located by annotating two points that surround the valve using mouse clicks on the camera image.

Using only this two-click input over the 2D image, the algorithm proceeds to fit the valve in the 3D point cloud and creates the affordance as shown in the 2D view

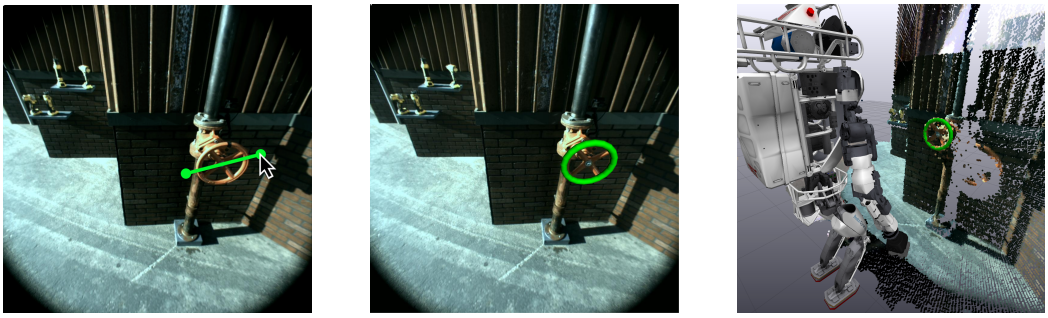


Figure 3-7: Fitting the valve affordance. User provides simple annotation in the 2D camera image (left), the fitting algorithm fits a valve affordance, results shown in 2D (middle) and in the 3D view (right).

in Figure 3-7 (middle) and the 3D view in Figure 3-7 (right).

We developed both automatic and operator guided object fitting algorithms for the competition. While we had some success with fully automated fitting, we focused more on guided fitting due to the competition rules. In the DRC Trials and Finals, unlike the VRC, operator input (operator-to-robot bandwidth usage) was not penalized, and the extra time required to provide operator input to a fitting algorithm was small compared with the time lost due to planning robot motion with a poorly fit affordance model. Thus, guided perception algorithms were used in most manipulation tasks at the DRC.

An operator guided perception algorithm was developed for the debris task to enable rapid fitting of lumber of arbitrary dimensions laying in unstructured configurations with overlap and occlusions. The algorithm was designed to fit rectangular prisms, and could be applied to boards, cinder blocks, bricks, and boxes. The operator began by selecting the expected cross-sectional dimensions of the affordance; for instance, lumber cross sections are typically 2×4 in, 4×4 in, or 2×6 in.

The operator then defined the search region by drawing a line annotation on the display using mouse clicks to define the endpoints. This annotation could be drawn either on a camera image or a 3D point cloud rendering; in practice we always annotated using the 3D rendering because the camera's limited field of view did not always include the region of interest for this task. The line end points were projected as rays in 3D originating from the camera center to define a bounded plane, and

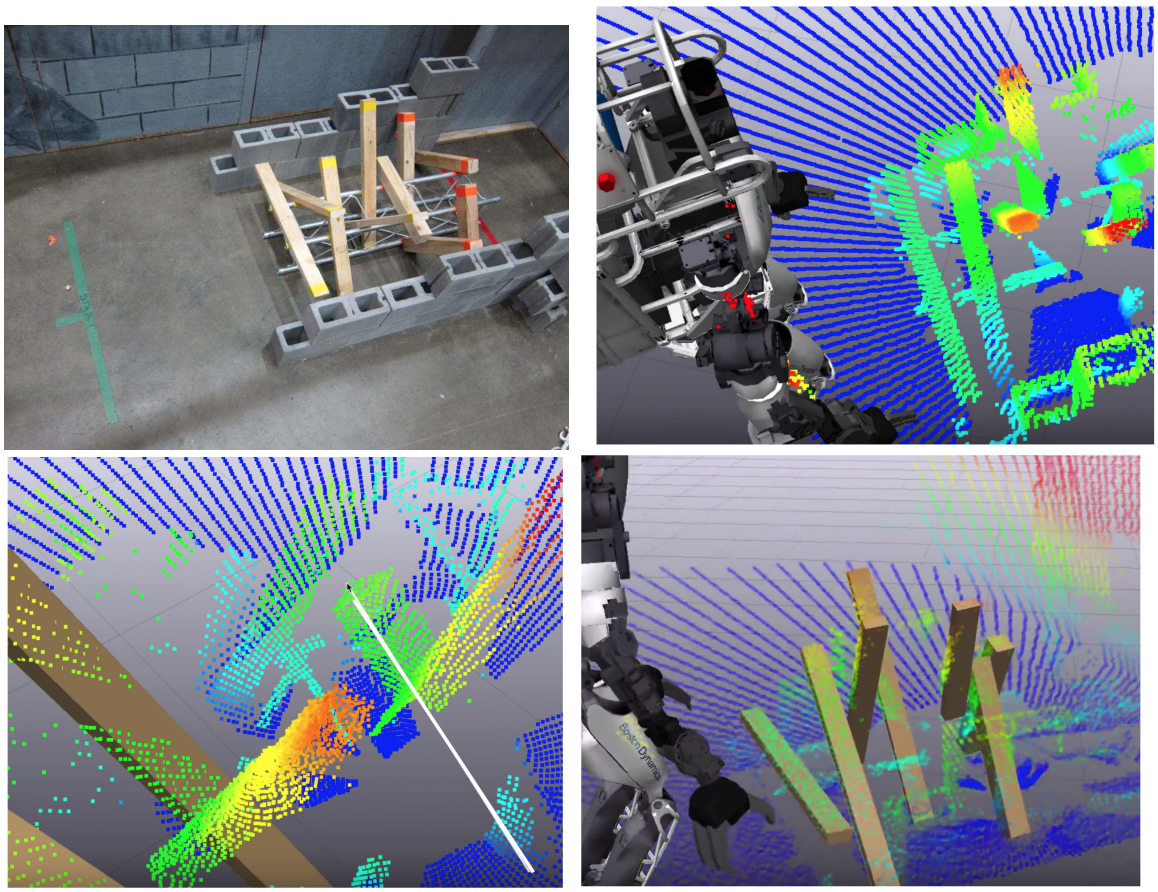


Figure 3-8: A fitting algorithm produces an affordance that can be used to plan a reach motion. A line annotation defines a search region that is used by a segmentation algorithm to fit a 2"x4" board affordance to the 3D point cloud data.

points in the LIDAR point cloud within a threshold distance of this plane were kept as the initial search region.

Given the initial search region, the board-fitting algorithm used RANSAC with a plane model to select inlier points that were then labeled as board face points. These were binned along the vector parallel to the edge connecting the search region ray end points, which was roughly parallel to the board's length axis. The algorithm identified one candidate edge point in each bin by selecting the maximum distance among that bin's points along the vector perpendicular to the RANSAC plane normal and length axis. Due to occlusions and adjacent surfaces, not all candidate edge points were true board edge points, so a RANSAC line fit was applied to the candidate edge points to label inliers presumed to be the board edge. The board affordance model was snapped to the fitted face plane and edge line, and extended along the edge line to encompass the minimum and maximum face point projections onto that line.

The resulting accurately-fit object position, orientation, and scale parameters were finally transmitted to an *affordance manager* which maintained the states of all identified affordances in the vicinity of the robot and made them available to both the user interface for visualizer, and to the planners for manipulation planning. Combined with the current robot state, this affordance information provided inputs necessary for the online planning routines of our shared autonomy task system.

3.5 Performance Evaluation at the DRC Finals

This section describes the performance of the Team MIT Atlas robot in the DRC Finals with qualitative analysis of the usage pattern of the shared autonomy system on the user interface. Our publication [75] presents a summary of each individual field task at the DRC Finals with an emphasis on the outcomes of using the workflow implemented in Director. We highlight the use of automation and teleoperation as it was required in the field, and of particular interest are the events that produced failures and therefore required manual control by the operator.

Figure 3-9 presents a selection of task execution times collected during the two

competition runs at the DRC Finals. On tasks where the shared autonomy system proceeded without interruption (operator intervention), execution times are consistent for each run. Tasks that required manual operator intervention were however slower: for example the door task on Day 1, and the terrain task on Day 2. Table 3.1 summarizes the failures for each task, and counts the number of plans computed automatically by the autonomy system versus the number of plans generated by the teleoperation interface during manual control.

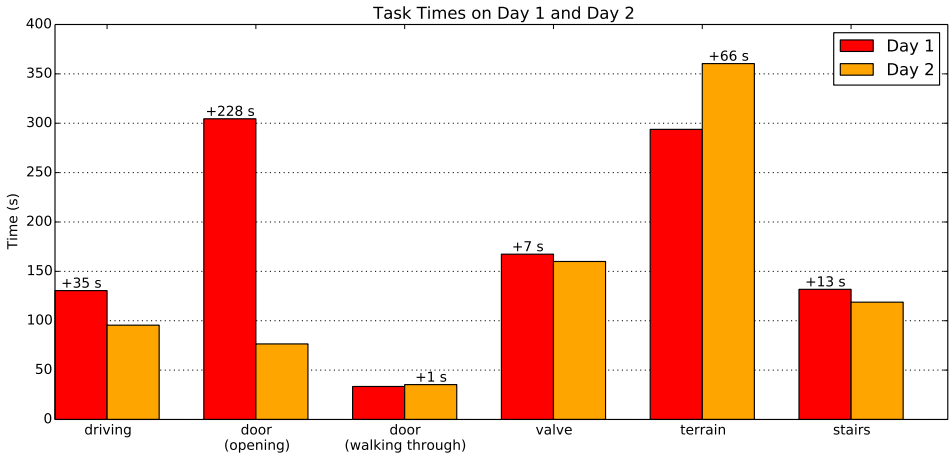


Figure 3-9: Timing comparison between tasks executed on competition Day 1 and Day 2. Task completion success varied over the two days so we cannot directly compare all eight tasks; egress and drill are not shown due to lack of completion on at least one day. The door task required manual intervention of Day 1, and terrain required manual intervention on Day 2, and took longer as a result.

The nominal workflow for task execution is based on the task sequencing that the operator interacts with in the task panel, i.e. the outer loop shown in Figure 3-3. This task sequencing, shown in the task queue in the center panel of the Figure, includes steps of three main types: (1) requires operator's input to an autonomous component, such as the valve fitting process explained before; (2) are fully autonomous, such as reaching to the valve; (3) are only manual by request of the automatic task sequencing, such as manually adjusting the fit of the valve when the task sequencing requires the operator to confirm or adjust the current fitting. A break from the nominal sequence is expected to happen only because of a system failure or unforeseen events that must alter the task sequencing. These events require more manual intervention from

the operator through teleoperation. The teleoperation process is still assisted by the automatic motion planner, once the operator has manually indicated a goal pose and a set of constraints.

Task failures can be detected automatically or detected by an operator. Although there is great benefit to detecting failures automatically, the detection creates cognitive burden for the operator because the operator has to read and understand the failure. For this reason, we preferred to anticipate most of the events that would potentially require the operator's intervention, and include them in the task sequencing as a request to the operator, as opposed to leaving them only to reactive intervention. In the months leading up to the competition we refined our task sequences to replace tasks that had a low success rate with tasks that incorporate shared autonomy to increase success rate. The typical case is a task that uses a perception algorithm to localize an affordance. The task will prompt the user to verify the affordance pose before proceeding with autonomous manipulation.

For this reason, in the competition the only task failures that led to pausing the task sequence were operator detected failures of unforeseen events. There were relatively few of these events, and they are listed in Table 3.1 as interrupts, the details of which are found in [75].

Table 3.1: A summary of the number of task failures during the competition that led to interrupting the automatic task execution sequence, and the types of plans executed for each task. During the driving task joint commands were streamed directly instead of using motion plans. N/a indicates that we did not attempt this task.

	Day 1			Day 2		
Task	Interrupts	Auto Plans	Teleop Plans	Interrupts	Auto Plans	Teleop Plans
Driving	0	0	*	0	0	*
Egress	1	6	0	0	9	0
Door	1	4	6	0	9	0
Valve	0	9	0	0	9	0
Drill	n/a	n/a	n/a	1	19	11
Surprise	0	4	12	n/a	n/a	n/a
Terrain	0	2	0	1	2	0
Stairs	0	3	0	0	3	0

3.6 Contribution of Shared Autonomy

We illustrate the case for increased levels of autonomy by exploring the performance in the valve task during laboratory experiments when using different sets of features available to the operator in the user interface.

Director is the central hub of interaction with many components of a larger and complex system that includes perception, planning, control and communications. While it is possible to evaluate the performance of these components in isolation, the resulting performance of such a complex system is a combination of success of individual components together with the results of the interactions between them. Director interacts directly with each component and uses this information to create an unified representation of the robot's world and actions that enables the operator to remotely control the robot to execute a variety of locomotion and manipulation tasks using a shared autonomy framework. A fundamental contribution of the interface is to properly integrate the interaction with the overall system into a single usable interface that exploits the advantages of the other components while keeping a coherent high-level view of the system, and we use our field performance during the DRC Finals as a proxy to evaluate its efficacy.

Ideally, we would like to quantify the contribution of the shared autonomy interface to our overall performance at the DRC competition relative to all the other components of our robot system. It is difficult to make direct comparisons between Director and our previous user interface used at the DRC Trials in December 2013 because many components of our robot system have changed, and the competition tasks and rules changed. Nonetheless, we performed an evaluation of the results for completion time of a task while enabling different features of our system. As there are limited trials on the field during competition, we present an analysis of the valve task performed with and without certain features of our shared autonomy and task automation performed in our laboratory. We tested the following four feature sets, which are summarized in Table 3.2.

Table 3.2: Explanation of feature sets used in valve task trials.

	Feature Set 1	Feature Set 2	Feature Set 3	Feature Set 4
Affordance Placement	none	manual	auto	auto
Navigation Goal Placement	manual	auto	auto	auto
Manipulation Planning	teleop	auto	auto	auto
Task Sequencing	no	no	no	yes

1. Teleoperation: the operator manually placed a navigation goal near the valve to create the walking plan. To turn the valve, the operator used our teleoperation interface to raise the arm, grasp, and turn the valve.

2. Affordance-based planning: the operator manually aligned a valve affordance, then invoked task specific planners to generate a navigation goal relative to the valve. After walking to the valve, the operator manually re-adjusted the valve affordance model to match the LIDAR point cloud. The operator used task specific planners to perform the manipulation task.

3. Affordance-based planning and automatic fitting: The operator used a perception fitting algorithm to automatically align the valve affordance. Everything is the same as feature set 2, but automatic fitting is used.

4. Task sequencing: the task panel is used to automatically sequence the task. Automatic fitting and affordance-based planning are also used. In this mode, the operator does not have to click in the interface to invoke the task specific planners for navigation and manipulation. The task queue automatically steps through the task while the operator supervises the execution. This feature set was used at the DRC Finals.

Figure 3-10 shows the timing results of the task repeated for 10 trials with each feature set. The trials were performed by the same operator over three different sessions. Even though this is a limited analysis, it demonstrates the case for increased levels of autonomy by measuring the performance benefit of adding assisted perception and planning. Note that the task time variance for teleoperation is higher due to the variability in the manner that the operator carries out manual control, whereas task time is more predictable when the task panel is used to sequence the task.

3.7 Discussion

The shared autonomy framework that Director implements is based on a handful of high-level principles such that the use of affordances as the center of the actions and

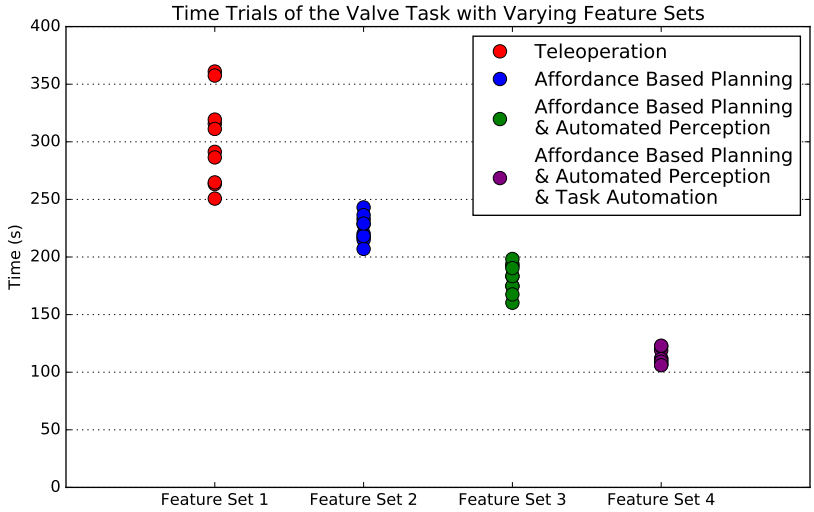


Figure 3-10: Valve task trial data collected from laboratory experiments illustrates the performance benefit (measured in task completion time) of varying feature sets within the Director user interface.

task planning through the generation of a sequence of motion plans that accomplish the task. While the Challenge accelerated progress on these fronts (enabling the execution of field tasks that were never demonstrated or were significantly slower at the start of the program), we have identified key limitation in these components that demand the attention of the research community. Expanding the potential of high-level planning will further unlock the capabilities developed during the DRC. In particular, from the development of the Director, we found challenges on the use of affordances when you encounter new objects and limitations on the assisted task planning when faced to a new task. We also note the need of improvements on collision avoidance planning, object fitting and tracking.

A key contribution we have presented in this chapter is a task execution user interface which assists the operator in commanding and supervising the execution of a queue of low-level atomic actions. While this workflow improved performance speed, the underlying approach is still limited to pre-specified tasks which required a robotics programmer, knowledgeable in the limitations of the robot as well as our algorithms, to explicitly encode task sequences. These steps took significant time to implement and to iterate upon for each new task sequence.

As mentioned in Section 3.4, we defined important objects in the environment as affordances and central to our semi-autonomy were the motion planning and object fitting functions we attached to their software representations. Our original approach to this, discussed in [76], was entirely abstract — focusing on parameterizing object degrees of freedom and defining interaction with the object considering the robot's limitations. For Director, we have extended the representation to encode properties relevant to both object and the robot, such as where to stand and the pre-grasp configuration in SO(3) relative to the position of the object, as well as custom motion planning constraints, in addition to the previously used object geometric parameters. We have found it challenging to transfer this information from one affordance to another without explicit modifications by our UI designer, and, while it is possible to use existing affordances in our library to fit previously unseen affordances as a solution for the perception problem, the planning problem is still difficult and the operator often uses teleoperation to compensate for actions that haven't been encoded.

We have described the graphical user interface and shared autonomy system used by Team MIT to pilot an Atlas robot in the DRC Finals. Our contribution focused on the development and field testing of this interface which supported the execution of complex manipulation and locomotion tasks. The approach alternated between autonomous behaviors represented in a task hierarchy supervised by an operator and teleoperation of the robot as required by task complexity. These methods empowered the operator to perform complex whole-body behaviors with a high-DoF robot, whereas previous technology deployed in field missions has been largely based on joint-level teleoperation of low-DoF robots.

A comparison of task execution in the field was limited to the two competition runs but it showed a consistent indication that using a larger portion of autonomy allowed for task completion in less time compared to manual intervention. To further explore the benefits of increased levels of autonomy, we performed the valve task in repeated timed trials with various feature sets for planning and perception. This laboratory testing allowed us to asses the contribution of individual features of the user interface, which we could not do using field data from the competition.

On both days of competition, the robot successfully completed the course in under 60 minutes, though not all tasks were completed for reasons discussed previously. Manual intervention by an operator was required occasionally, but overall we felt we achieved our goal of fielding a robot system that was largely based in autonomy. We have described how our shared autonomy designs performed under competition pressures, and highlighted what worked well, what failed, and the limitations that remain.

We have released Director[1] under an open-source license, including all of the software described in this article, and the larger codebase developed for our DRC entry[2], in which Director is the user interface submodule. The software has been generalized to support a variety of robots with different kinematics, locomotion methods, end effectors, sensors and firmware, as illustrated in Figure 3-11, and continues to be used by a growing community of users since the DRC project.

[1] http://github.com/RobotLocomotion/director
[2] http://github.com/OpenHumanoids/oh-distro

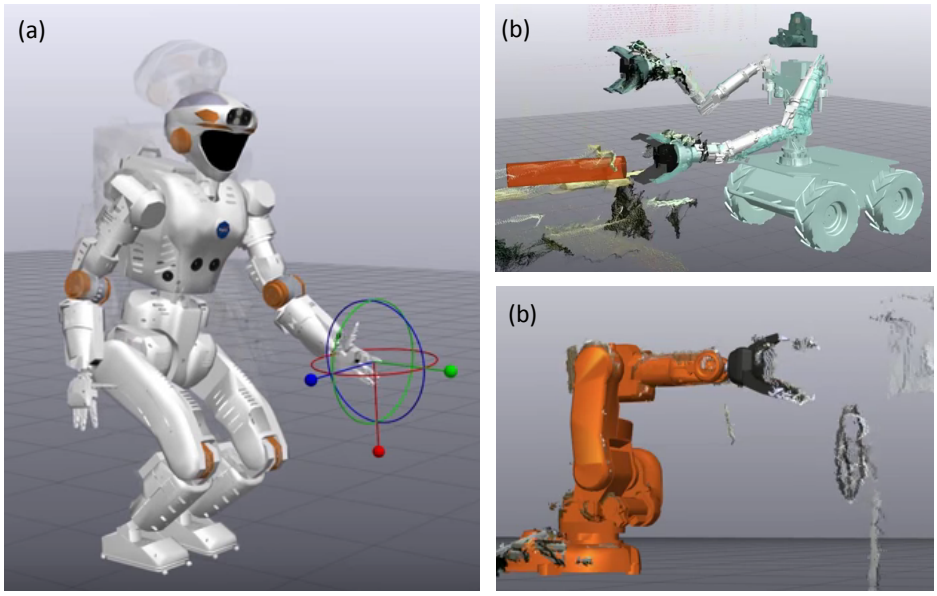


Figure 3-11: Director being used with different classes of robots: (a) NASA's *Valkyrie*, a humanoid robot, shown using the constrained end-effector teleoperation feature; (b) MIT's *Optimus*, a dual arm mobile manipulator for EOD, shown with an affordance fitted to a stereo point cloud [79]; (c) an ABB fixed-base industrial manipulator and Kinect RGB-D data.

Chapter 4

Conclusions and Future Work

4.1 Future Work

This thesis has presented algorithms and hardware demonstrations for continuous humanoid locomotion over uneven terrain, using online perception and footstep planning, but many opportunities for improvements still remain. This chapter will describe some of the most relevant pieces of future work in this problem domain, including region segmentation post-processing using convex decomposition, speed ups and refinements to support faster, more dynamic walking, and algorithmic extensions for grasping and manipulation domains.

4.1.1 Convex Region Decomposition

In this work we have represented planar region segmentations of unorganized point clouds with polygons in 3D. The segmentation is a mapping of points in the point cloud to distinct planar regions, but converting those point sets to polygonal representations presents further opportunities for investigation. For our autonomous walking experiments described in 2.7, we computed the minimum area bounding rectangles of the points for each region. The man-made terrain features traversed by the robot in our experiments and at the DRC Finals course, such as stairs and concrete blocks, had natural convex outlines, and were therefore best represented with convex hulls and

rectangles. However, as shown in Figure 4-1, some planar regions in the environment are not well represented with convex hulls.

In future work, we could explore convex decomposition strategies to break the regions into several pieces that individually may be accurately represented with convex hulls. In IRIS, the segmentation algorithm finds convex regions in configuration space, but it requires multiple searches with seed points derived from user input or random sampling in order to cover concave regions of the environment with overlapping convex regions. The point cloud segmentation algorithm presented in this thesis is able to fully segment concave planar regions in task space, but requires post-processing to decompose segmentations into convex pieces.

Figure 4-1 shows preliminary work in post-processing point could segmentations into convex pieces, applied to a LIDAR scan from the DRC Finals door task. The segmentation algorithm has successfully separated the recessed door from the wall, but representing the wall segmentation with a convex hull produces a polygon that is not adequate because it includes large areas that do not correspond to wall points. Instead, we compute the concave hull of the point set using the Delaunay triangulation based alpha shapes algorithm [80]. Whereas a point set has a unique minimal convex hull, the concave hull boundary depends on the selection of the α parameter used to prune edges of the Delaunay triangulation. Next, the concave hull is broken into convex pieces using approximate convex decomposition [81]. The algorithm splits concave shapes based on a measure of concavity τ, which is the distance from the vertex of a concave notch to the convex hull. The algorithm runs recursively to split the concave hull until all pieces have concavity measure less than a user selected τ value.

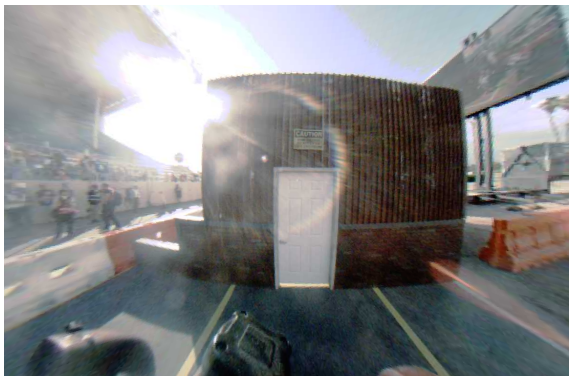
This decomposition strategy works well when applied to the results of our segmentation, but still requires user selected parameters. Examples of applying these algorithms to the LIDAR scan while varying α and τ parameters are shown in Figure 4-2. It is difficult to set these parameters to adequately process a whole environment which contains features of varying scales and sensor data of varying sample density. Although it is possible to find values that are sufficient for the terrain features

evaluated in our experiments so far, in order to plan contact between the robot and the environment using planar approximations of non terrain regions, such as walls, it will be important to study improved methods of convex region decomposition.

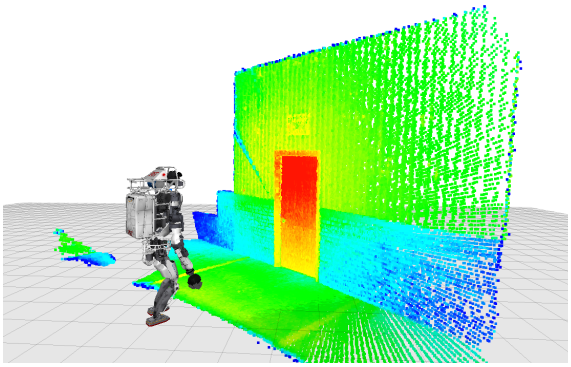
4.1.2 Dynamic Walking

As discussed by Deits [82], extending planning capabilities to support dynamic walking planning is important to enable faster motions and guarantee feasibility of footstep plans. To support these capabilities, in addition to planning extensions, the perception pipeline must be accelerated. Our presented pipeline was tested with slow walking rates on the order of several seconds per footstep. We can make our perception pipeline operate more quickly by tuning parameters. For example, increasing the downsampling parameter results in fewer points to process, so point cloud processing algorithms for surface normal estimation and region growing run more quickly, but at the cost of less accurate representations of terrain features. It is also possible to decrease the search radius of surface normal estimation, so as to decrease the run time of that step of the pipeline. We may be able to achieve a good speed up through parallel implementations of normal estimation and region growing without sacrificing resolution and accuracy.

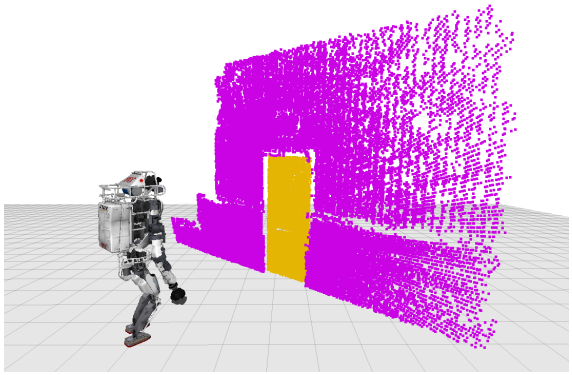
Finally, even with quicker point cloud processing, we still have the problem of data collection rates. Currently, our segmentation pipeline runs more quickly than our LIDAR point cloud update rate, but slower than our stereo update rate. As the robot moves more quickly, the LIDAR update rate will be a bottle neck. The stereo camera updates as sufficient speed, but when the robot moves more quickly the Kintinuous fusion algorithm will have less time to average observations of terrain before the reconstruction must be snapshotted for input to region segmentation. More work has to be done to evaluate the speed of walking at which our perception pipeline fails to provide adequate regions for footstep planning.



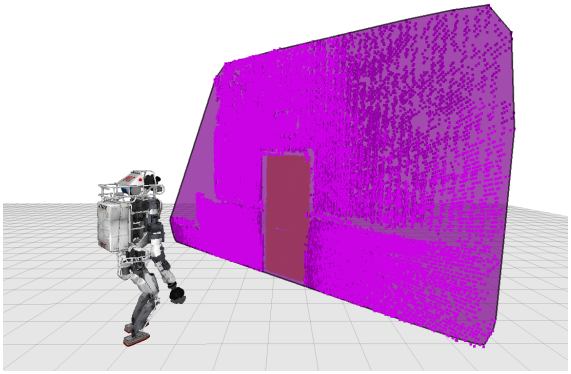
(a) Robot's camera view of the wall and door.



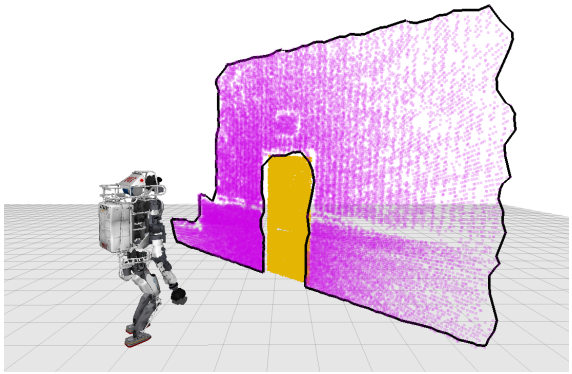
(b) LIDAR scan of the wall and door, colored by intensity of return.



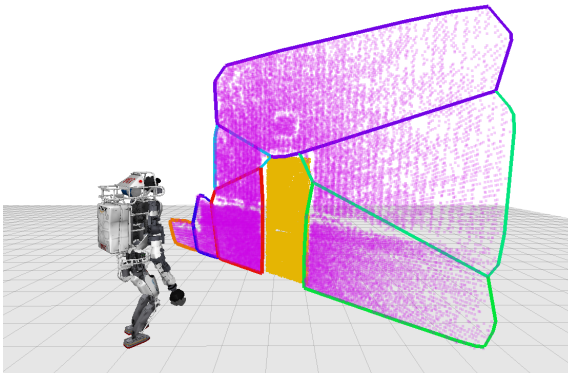
(c) Planar region segmentation result, the recessed door is separated from the wall.



(d) The convex hull of the wall points is not an accurate representation.

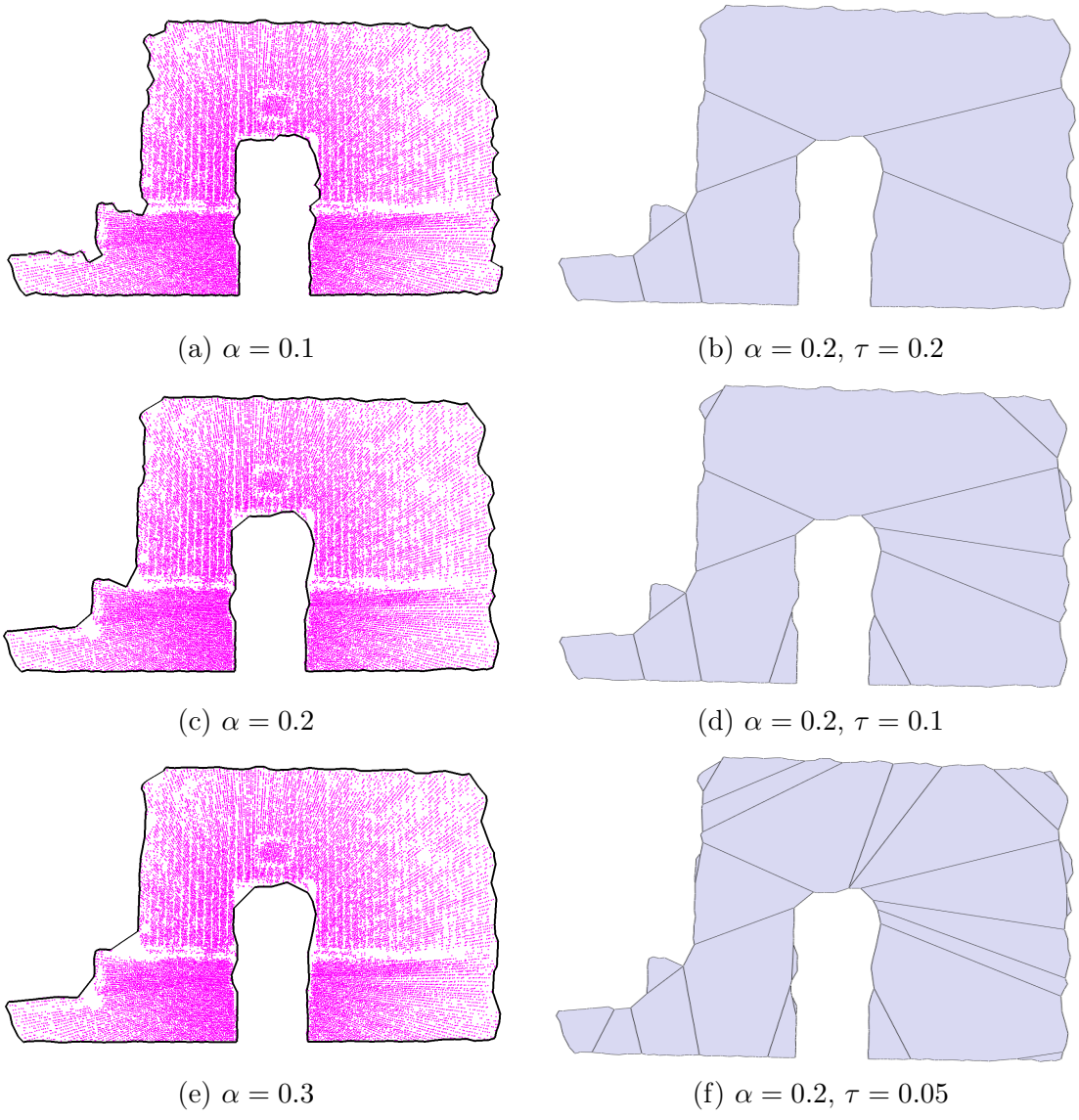


(e) The concave hull of the wall points computed with $\alpha = 0.2$.



(f) An approximate convex decomposition of the concave hull with $\tau = 0.2$ produces 7 new regions.

Figure 4-1: Convex decomposition of planar region segmentation applied to a LIDAR scan. Note that in the planar segmentation, some points on the right side of the wall have been filtered and lost because the sample density was too low for surface normal estimation with our selected search radius.



(a) $\alpha = 0.1$

(b) $\alpha = 0.2$, $\tau = 0.2$

(c) $\alpha = 0.2$

(d) $\alpha = 0.2$, $\tau = 0.1$

(e) $\alpha = 0.3$

(f) $\alpha = 0.2$, $\tau = 0.05$

Figure 4-2: Result of approximate convex decomposition applied to the wall LIDAR scan. Left column, varying the α parameter of the concave hull computation. Right column, with concave hull $\alpha = 0.2$, varying the τ parameter of the approximate convex decomposition algorithm.

4.1.3 Segmentation for Grasp Planning

Although this work has applied planar region segmentation to terrain, the segmentation algorithm is also able to compute planar approximations of graspable objects for robotic manipulation. Figures 4-3 and 4-4 show results of the proposed segmentation pipeline where input parameters have been selected to find planar regions that can be used for contact planning for force closure grasps. This procedure may be useful for computing grasps on objects without prior models.

In Figure 4-3, the segmentation is applied to objects from the Berkeley Instance Recognition Dataset [83]. The objects in the database are provided as meshed surfaces reconstructed from depth images at multiple viewing angles. Only the vertices of the mesh are used and are treated as an unorganized point cloud. However, in manipulation scenarios without prior models, it is unlikely that the robot's depth sensor will be able to observe the object from all angles. Figure 4-4 shows some preliminary results of the segmentation applied to single view RGB-D data of table top objects viewed from a distance of around 1.5 meters.

Planar approximations of small objects presents new challenges that we did not encounter when applying the algorithms for walking. Graspable objects often have rounded features, concave notches, and salient features of varying scales, which are not easily captured with a single tuning of parameters of the segmentation algorithm. Additionally, objects to be manipulated often appear together close proximity, and planar segmentation may group them together. For example, multiple books on a shelf will be considered to be a single planar region. Depth processing alone cannot resolve these challenging situations—the segmentation must be informed by camera RGB data, as proposed by Strom in [50], although that method was not applied to small scale objects. It is not clear that planar approximations are the best representation for small scale objects, but some of the methods that we have applied for terrain estimation such as RANSAC model fitting may be good strategies to apply here, such as the primitive shape fitting proposed by Schnabel et al. in [84], a method designed for high resolution scans of detailed parts.

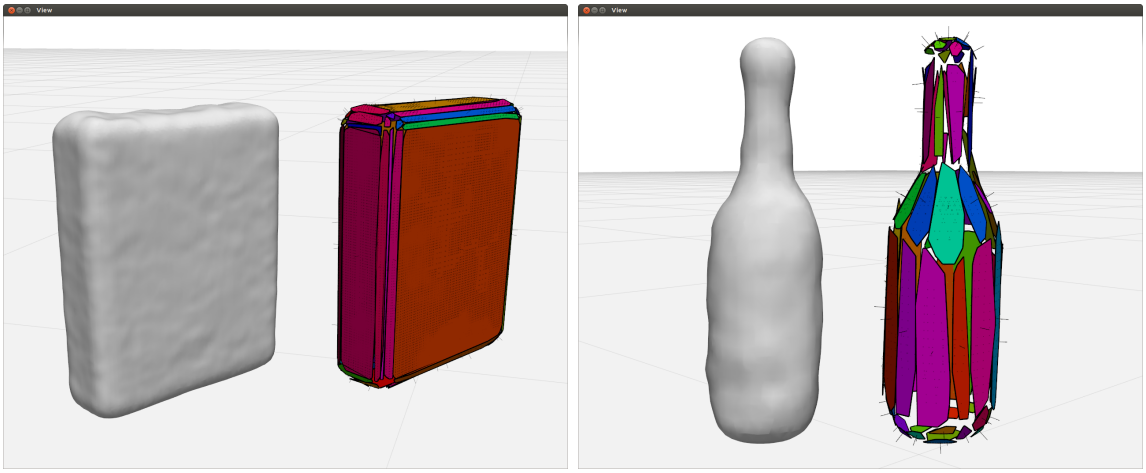


Figure 4-3: Planar segmentation of example objects from the Berkeley Instance Recognition Dataset.

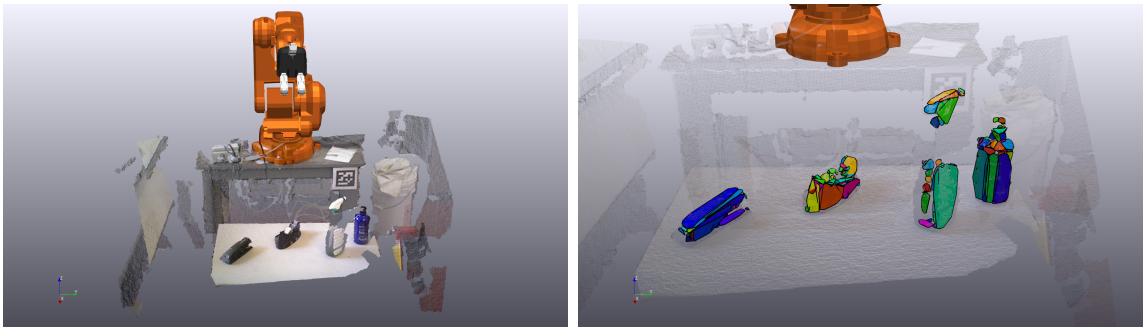


Figure 4-4: Planar segmentation of graspable objects sitting on a table collected with a single depth image of a RGB-D sensor.

4.2 Conclusion

We have presented a terrain perception algorithm and method to combine it with a footstep planner and autonomous execution framework to accomplish continuous humanoid locomotion over uneven terrain. The technique hinges on planar region segmentation using robust surface normal estimation, applied directly to a 3D map, resulting in a 3D polygon representation of regions for footstep planning. We have shown that the algorithm's robust design is well suited to process fused stereo depth maps from passive imaging sensors, as compared to slower but more accurate LIDAR scans, opening the door to faster and more dynamic locomotion that will require quicker updates from the terrain perception pipeline.

The terrain perception and continuous walking task was enabled by Director, our robot user interface and operation software. We described how the design of Director supports shared autonomy through integration of a perception framework that supports rapid prototyping, and integrates human input with the perception and motion planning subsystems. The benefits of the design were tested and validated through a series of human-in-the-loop experiments that demonstrate the increased speed performance when perception and task sequencing are incorporated into a user interface. In the future, we look forward seeing additional robots and task behaviors added to Director to continue supporting humanoid research and operation.

Bibliography

[1] IEEE Spectrum. A compilation of robots falling down at the darpa robotics challenge. https://www.youtube.com/watch?v=g0TaYhjpOfo, 2015.

[2] DRC-Teams. What happened at the DARPA Robotics Challenge? http://www.cs.cmu.edu/~cga/drc/events, 2015.

[3] Boston Dynamics. Atlas update. https://www.youtube.com/watch?v=SD6Okylclb8, 2013.

[4] J. of field robotics: Special issue on the darpa robotics challenge (drc), 2015.

[5] James J Kuffner, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Footstep planning among obstacles for biped robots. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 500–505. IEEE, 2001.

[6] Joel Chestnutt, James Kuffner, Koichi Nishiwaki, and Satoshi Kagami. Planning biped navigation strategies in complex environments. *IEEE Int. Conf. Hum. Rob., Munich, Germany*, 2003.

[7] Joel Chestnutt, Manfred Lau, German Cheung, James Kuffner, Jessica Hodgins, and Takeo Kanade. Footstep planning for the honda asimo humanoid. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 629–634. IEEE, 2005.

[8] Joel Chestnutt, Yutaka Takaoka, Keisuke Suga, Koichi Nishiwaki, James Kuffner, and Satoshi Kagami. Biped navigation in rough environments using on-board sensing. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3543–3548. IEEE, 2009.

[9] Stefan Oßwald, Jens-Steffen Gutmann, Armin Hornung, and Maren Bennewitz. From 3d point clouds to climbing stairs: A comparison of plane segmentation approaches for humanoids. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 93–98. IEEE, 2011.

[10] Koichi Nishiwaki, Joel Chestnutt, and Satoshi Kagami. Autonomous navigation of a humanoid robot over unknown rough terrain using a laser range sensor. *The International Journal of Robotics Research*, 31(11):1251–1262, 2012.

[11] Koichi Nishiwaki, Joel Chestnutt, and Satoshi Kagami. Planning and control of a humanoid robot for navigation on uneven multi-scale terrain. In *Experimental Robotics*, pages 401–415. Springer Berlin Heidelberg, 2014.

[12] Evan Ackerman. SCHAFT unveils awesome new bipedal robot at japan conference. http://spectrum.ieee.org/automaton/robotics/humanoids/shaft-demos-new-bipedal-robot-in-japan, 2016.

[13] Olivier Stasse, François Saïdi, Kazuhito Yokoi, Björn Verrelst, Bram Vanderborght, Andrew Davison, Nicolas Mansard, and Claudia Esteves. Integrating walking and vision to increase humanoid autonomy. *International Journal of Humanoid Robotics*, 5(02):287–310, 2008.

[14] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1620–1626. IEEE, 2003.

[15] Nicolas Perrin, Olivier Stasse, Florent Lamiraux, and Eiichi Yoshida. A biped walking pattern generator based on "half-steps" for dimensionality reduction. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1270–1275. IEEE, 2011.

[16] Nicolas Perrin, Olivier Stasse, Florent Lamiraux, and Eiichi Yoshida. Weakly collision-free paths for continuous humanoid footstep planning. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4408–4413. IEEE, 2011.

[17] Nicolas Perrin, Olivier Stasse, Léo Baudouin, Florent Lamiraux, and Eiichi Yoshida. Fast humanoid robot collision-free footstep planning using swept volume approximations. *Robotics, IEEE Transactions on*, 28(2):427–439, 2012.

[18] Nicolas Perrin, Olivier Stasse, Florent Lamiraux, Young J Kim, and Dinesh Manocha. Real-time footstep planning for humanoid robots among 3d obstacles using a hybrid bounding box. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 977–982. IEEE, 2012.

[19] Léo Baudouin, Nicolas Perrin, Thomas Moulard, Florent Lamiraux, Olivier Stasse, and Eiichi Yoshida. Real-time replanning using 3d environment for humanoid robot. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 584–589. IEEE, 2011.

[20] Oscar E Ramos, Mauricio Garcia, Nicolas Mansard, Olivier Stasse, Jean-Bernard Hayet, and Philippe Soueres. Toward reactive vision-guided walking on rough terrain: An inverse-dynamics based approach. *International Journal of Humanoid Robotics*, 11(02):1441004, 2014.

[21] Stefan Oßwald, Attila Görög, Armin Hornung, and Maren Bennewitz. Autonomous climbing of spiral staircases with humanoids. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4844–4849. IEEE, 2011.

[22] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[23] David Maier, Anja Hornung, and Maren Bennewitz. Real-time navigation in 3d environments based on depth camera data. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 692–697. IEEE, 2012.

[24] Xiaoyi Jiang and Horst Bunke. Fast segmentation of range images into planar regions by scan line grouping. *Machine vision and applications*, 7(2):115–122, 1994.

[25] JS Gutmann, K Kawamoto, K Sabe, and M Fukuchi. Multiple plane segmentation with the application of stair climbing for humanoid robots. In *Symposium on Intelligent Autonomous Vehicles (IAV)*, 2004.

[26] Jens-Steffen Gutmann, Masaki Fukuchi, and Masahiro Fujita. A floor and obstacle height map for 3d navigation of a humanoid robot. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1066–1071. IEEE, 2005.

[27] Jens-Steffen Gutmann, Masaki Fukuchi, and Masahiro Fujita. A modular architecture for humanoid robot navigation. In *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*, pages 26–31. IEEE, 2005.

[28] Jens-Steffen Gutmann, Masaki Fukuchi, and Masahiro Fujita. 3d perception and environment map generation for humanoid robot navigation. *The International Journal of Robotics Research*, 27(10):1117–1134, 2008.

[29] Maurice F Fallon, Matthew Antone, Nicholas Roy, and Seth Teller. Drift-free humanoid state estimation fusing kinematic, inertial and lidar sensing. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 112–119. IEEE, 2014.

[30] Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 279–286. IEEE, 2014.

[31] Maurice F. Fallon, Pat Marion, Robin Deits, Thomas Whelan, Matthew Antone, John McDonald, and Russ Tedrake. Continuous humanoid locomotion over uneven terrain using stereo fusion. In *Humanoid Robots (Humanoids), 2015 15th IEEE-RAS International Conference on*. IEEE, 2015.

[32] Timothy Bretl. Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem. *The International Journal of Robotics Research*, 25(4):317–342, 2006.

[33] Kris Hauser, Timothy Bretl, Jean-Claude Latombe, and Brian Wilcox. Motion planning for a six-legged lunar robot. In *Algorithmic Foundation of Robotics VII*, pages 301–316. Springer Berlin Heidelberg, 2008.

[34] Eric Krotkov and Reid Simmons. Perception, planning, and control for autonomous walking with the ambler planetary rover. *The International journal of robotics research*, 15(2):155–180, 1996.

[35] Katie Byl, Alec Shkolnik, Sam Prentice, Nick Roy, and Russ Tedrake. Reliable dynamic motions for a stiff quadruped. In *Experimental robotics*, pages 319–328. Springer Berlin Heidelberg, 2009.

[36] Alexander Shkolnik, Michael Levashov, Ian R Manchester, and Russ Tedrake. Bounding on rough terrain with the littledog robot. *The International Journal of Robotics Research*, page 0278364910388315, 2010.

[37] J Zico Kolter, Youngjun Kim, and Andrew Y Ng. Stereo vision and terrain modeling for quadruped robots. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 1557–1564. IEEE, 2009.

[38] J Zico Kolter, Mike P Rodgers, and Andrew Y Ng. A control architecture for quadruped locomotion over rough terrain. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 811–818. IEEE, 2008.

[39] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 1442–1447. Ieee, 1991.

[40] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[41] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, and John McDonald. Kintinuous: Spatially extended kinectfusion. 2012.

[42] Radu Rusu, Aravind Sundaresan, Benoit Morisset, Kris Hauser, Motilal Agrawal, Jean-Claude Latombe, and Michael Beetz. Leaving flatland: Efficient real-time three-dimensional perception and motion planning. *Journal of Field Robotics*, 26(10):841–862, 2009.

[43] Benoit Morisset, Radu Bogdan Rusu, Aravind Sundaresan, Kris Hauser, Motilal Agrawal, Jean-Claude Latombe, and Michael Beetz. Leaving flatland: toward real-time 3d navigation. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3786–3793. IEEE, 2009.

[44] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[45] Radu Bogdan Rusu, Ioan Alexandru Şucan, Brian Gerkey, Sachin Chitta, Michael Beetz, and Lydia E Kavraki. Real-time perception-guided motion planning for a personal robot. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4245–4252. IEEE, 2009.

[46] Bertrand Douillard, James Underwood, Noah Kuntz, Vsevolod Vlaskine, Alastair Quadros, Peter Morton, and Alon Frenkel. On the segmentation of 3d lidar point clouds. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2798–2805. IEEE, 2011.

[47] Frank Moosmann, Oliver Pink, and Christoph Stiller. Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion. In *Intelligent Vehicles Symposium, 2009 IEEE*, pages 215–220. IEEE, 2009.

[48] Frederick Pauling, Mike Bosse, and Robert Zlot. Automatic segmentation of 3d laser point clouds by ellipsoidal region growing. In *Australasian Conference on Robotics and Automation (ACRA)*, 2009.

[49] Alexander JB Trevor, Suat Gedikli, Radu B Rusu, and Henrik I Christensen. Efficient organized point cloud segmentation with connected components. *Semantic Perception Mapping and Exploration (SPME)*, 2013.

[50] Johannes Strom, Andrew Richardson, and Edwin Olson. Graph-based segmentation for colored 3d laser point clouds. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2131–2136. IEEE, 2010.

[51] Alexandre Boulch and Renaud Marlet. Fast and robust normal estimation for point clouds with sharp features. In *Computer graphics forum*, volume 31, pages 1765–1774. Wiley Online Library, 2012.

[52] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[53] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[54] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.

[55] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Trans. Pattern Anal. Machine Intell.*, 30(2):328–341, Feb 2008.

[56] Carnegie Robotics. Multisense sl product page. http://carnegierobotics.com/multisense-sl, 2016.

[57] Inwook Shim, Seunghak Shin, Yunsu Bok, Kyungdon Joo, Dong-Geol Choi, Joon-Young Lee, Jaesik Park, Jun-Ho Oh, and In So Kweon. Vision system and depth processing for drc-hubo+. *arXiv preprint arXiv:1509.06114*, 2015.

[58] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Intl. j. of Computer Vision*, pages 7–42, May 2002.

[59] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE/ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*, October 2011.

[60] T. Whelan, M. Kaess, H. Johannsson, M.F. Fallon, J.J. Leonard, and J.B. McDonald. Real-time large scale dense RGB-D SLAM with volumetric fusion. *Intl. J. of Robotics Research*, 2015. To appear.

[61] Maurice F. Fallon, Pat Marion, Robin Deits, Thomas Whelan, Matthew Antone, John McDonald, and Russ Tedrake. Continuous humanoid locomotion over uneven terrain using stereo fusion. In *IEEE/RSJ Int. Conf. on Humanoid Robots*, Seoul, Korea, November 2015.

[62] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[63] Robin L.H. Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semi-definite programming. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Istanbul, Turkey, August 2014.

[64] Philipp Michel, Joel Chestnutt, James Kuffner, and Takeo Kanade. Vision-guided humanoid footstep planning for dynamic environments. In *IEEE/RSJ Int. Conf. on Humanoid Robots*, Tsukuba, Japan, December 2005.

[65] Koichi Nishiwaki, Joel Chestnutt, and Satoshi Kagami. Autonomous navigation of a humanoid robot over unknown rough terrain using a laser range sensor. *Intl. J. of Robotics Research*, 31:1251–1262, 2012.

[66] Johannes Garimort and Armin Hornung. Humanoid navigation with dynamic footstep plans. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3982–3987, 2011.

[67] LÃľo Baudouin, Nicolas Perrin, Thomas Moulard, Florent Lamiraux, Olivier Stasse, and Eiichi Yoshida. Real-time replanning using 3D environment for humanoid robot. In *IEEE/RAS Int. Conf. on Humanoid Robots*, pages p.584–589, Bled, Slovenia, 2011.

[68] Weiwei Huang, Junggon Kim, and C.G. Atkeson. Energy-based optimal step planning for humanoids. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3124–3129, Karlsruhe, Germany, May 2013.

[69] Joel Chestnutt, James Kuffner, Koichi Nishiwaki, and Satoshi Kagami. Planning biped navigation strategies in complex environments. In *IEEE/RSJ Int. Conf. on Humanoid Robots*, Karlsruhe, Germany, 2003.

[70] Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *IEEE/RSJ Int. Conf. on Humanoid Robots*, Madrid, Spain, 2014.

[71] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, UK, 2004.

[72] A Richards and J. How. Mixed-integer programming for control. In *American Control Conference*, pages 2676–2683 vol. 4, June 2005.

[73] Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2014.

[74] Russ Tedrake. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2014.

[75] Pat Marion, Robin Deits, Andrés Valenzuela, Claudia Pérez D'Arpino, Greg Izatt, Lucas Manuelli, Matt Antone, Hongkai Dai, Twan Koolen, John Carter, Maurice Fallon, Scott Kuindersma, and Russ Tedrake. Director: A user interface designed for robot operation with shared autonomy. *Under review*, 2016.

[76] Maurice Fallon, Scott Kuindersma, Sisir Karumanchi, Matthew Antone, Toby Schneider, Hongkai Dai, Claudia Pérez D'Arpino, Robin Deits, Matt DiCicco, Dehann Fourie, Twan Twan Koolen, Pat Marion, Michael Posa, Andrés Valenzuela, Kuan-Ting Yu, Julie Shah, Karl Iagnemma, Russ Tedrake, and Seth Teller. An architecture for online affordance-based perception and whole-body planning. *Journal of Field Robotics*, 32(2):229–254, 2015.

[77] Will J Schroeder, Bill Lorensen, and Ken Martin. *The Visualization Toolkit: an object-oriented approach to 3D graphics*. Kitware, 4th edition edition, 2008.

[78] Pat Marion, Roland Kwitt, Brad Davis, and Michael Gschwandtner. PCL and ParaView—connecting the dots. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 80–85. IEEE, 2012.

[79] Claudia Pérez D'Arpino. Optimus robot - shared autonomy. https://www.youtube.com/watch?v=oqeI6xzgqgA, 2015.

[80] Herbert Edelsbrunner, David G Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *Information Theory, IEEE Transactions on*, 29(4):551–559, 1983.

[81] Jyh-Ming Lien and Nancy M Amato. Approximate convex decomposition of polygons. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 17–26. ACM, 2004.

[82] Robin LH Deits. Convex segmentation and mixed-integer footstep planning for a walking robot. Master's thesis, Massachusetts Institute of Technology, 2014.

[83] Ashutosh Singh, Jin Sha, Karthik S Narayan, Tudor Achim, and Pieter Abbeel. Bigbird: A large-scale 3d database of object instances. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 509–516. IEEE, 2014.

[84] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007.