Director: A User Interface Designed for Robot Operation With Shared Autonomy

Pat Marion[1], Robin Deits[1], Andrés Valenzuela[1], Claudia Pérez D'Arpino[1], Greg Izatt[1]
Lucas Manuelli[1], Matt Antone[1], Hongkai Dai[1], Twan Koolen[1], John Carter[1]
Maurice Fallon[2], Scott Kuindersma[3], and Russ Tedrake[1]

[1]**Computer Science and Artificial Intelligence Laboratory**
Massachusetts Institute of Technology
32 Vassar Street
Cambridge, MA 02139 USA

[2]**School of Informatics**
University of Edinburgh
10 Crichton Street
Edinburgh EH8 9AB, UK

[3]**Paulson School of Engineering and Applied Sciences**
Harvard University
33 Oxford Street
Cambridge, MA 02138 USA

Abstract

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 motion planning and control. A well-designed operator interface presents the operator with enough data and flexibility to handle off-script operating scenarios robustly, without overwhelming the user with unnecessary information. By contrast, an unintuitive user interface can increase the risk of catastrophic operator error. With that 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 tasks, such as drilling a wall or climbing a staircase. These task sequences, developed *a priori*, make queries to automated perception and planning algorithms 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. We discuss the primary interface elements that comprise the Director and provide analysis of its successful use at the DRC.

1 Introduction

The DARPA Robotics Challenge (DRC) was a multi-year international competition focused on robotic technology for disaster response. While the DRC fueled many innovations in mobile manipulation technology, some of the most dramatic demonstrations were in the diverse and capable *user interfaces* developed by the teams to remotely pilot their robot through the competition tasks. In stark contrast to the slow and inefficient joystick interfaces commonly used by service robots such as the Packbot (Yamauchi, 2004) for

bomb disposal or visual inspection, these interfaces allowed the operators to efficiently command very high degree-of-freedom robots performing a series of locomotion and manipulation tasks all in succession in less than an hour. Bandwidth between the robot and the operator was intermittently restricted in order to encourage partial autonomy: a robot capable of operating with little or no human intervention could carry out tasks regardless of the state of its connection to the human operator. However, the focus on the competition was not on complete autonomy: a low-bandwidth always-on network communication link was provided, allowing some collaboration between the robot and the human operator. To be competitive in this format, teams had to deliver a robotic software system with considerable fluidity, flexibility, and robustness.

Our team adopted a philosophy of shared autonomy. Our goal was to design a system capable of completing tasks autonomously, but always able to fall back to a manual operation mode to allow a human to perform part of the task at multiple layers of granularity. Additionally, the autonomous behavior should be organized so that it is possible to resume the autonomous mode as early 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 sub-task 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.

We describe *Director*, a new graphical user interface and software framework that was used by Team MIT to pilot the Atlas robot in the DRC finals. Director interacts with our full robot system, the architecture of which we have described in a previous publication (Fallon et al., 2015a). In Section 2, we describe the features of the user interface that implement a shared autonomy system based on task sequences. Section 3 describes the programming models underlying the user interface and task system. We evaluate the performance of the interface and shared autonomy system during the competition runs at the DRC finals in Section 4.

1.1 Shared Autonomy

The concept of Shared Autonomy is based on the idea of integrating the inputs from a human operator with the computation of an autonomous system to produce the final behavior of the robot. This approach aims to maximize the use of the inherent advantages of each part of the system by relying on the autonomous system to compute and operate over the domain in which it outperforms the human operator (computational speed and accuracy for instance) and leaving other tasks that are out of the scope of the machine, such as cognitive tasks and supervision of the overall system, to the human operator. One example approach and implementation of shared autonomy is found in the field of teleoperation, in which a human operator controls the motion of the robot by its own physical motion as commands or through some physical input device. In addition to the correspondence problem (Nehaniv and Dautenhahn, 2002), there are challenges associated with the precision of the human motion needed to consistently achieve tasks such as grasping in this setup. Previous work has developed an arbitration system to blend the input of the human operator with the assistance in motion planning from the autonomous system using an arbitration function to produce a shared control law (Dragan and Srinivasa, 2012) (Dragan and Srinivasa, 2013) that aims to assist the operator in overcoming the limitations of the interface. More recently, this approach was adapted to teleoperation using a brain computer interface (Muelling et al., 2015)(Jain et al., 2015), in which the human input comes from brain signals instead of human motion. Other line of work has explored the performance effects of adding autonomous behaviors to a teleoperation system in the context of rescue missions (O'Brien et al., 2010).

Director is designed to support an approach to shared autonomy where the arbitration is performed by the human operator through a user interface. As opposed to continuously blending human input with autonomous control, our system is based on discrete sets of actions that are executed either by the computer or by the human as directed by the human operator. When the actions are computed by planning algorithms, they can be reviewed and approved by the operator before execution to guarantee nominal operation under novel circumstances. The interaction as arbitration also enables the operator to react when a particular situation falls outside the scope of the automated procedures by using lower-level interactions (more manual control). This approach is also better suited to the operation of robots remotely under time-delayed communications where direct teleoperation is not possible.

1.2 Approaches to task planning

High level tasks such as "walk through the door" are broken down into sequences of steps, each of which can be considered a task itself. For example, we might represent the sequence of subtasks as *find the door, walk to the door, open the door, walk through the door.* The *open the door* task can be divided further into the sequence *locate the door handle, reach to the handle, grasp, turn, push.* Thus, a complex task can be thought of as a task hierarchy, which can be represented as a tree, and the complete task is executed by stepping through each leaf task in the pre-specified order.

There is a long history of research focusing on task planning and decomposition strategies for autonomous robotic manipulation. Behavior trees were used for autonomous manipulation in the DARPA ARM challenge (Bagnell et al., 2012). Hierarchical state machines are implemented by the ROS SMACH library (Rusu et al., 2009; Bohren et al., 2011). Linear temporal logic can be used to synthesis task controllers (Kress-Gazit et al., 2007; He et al., 2015). Integrated task and motion planning (ITMP) combines symbolic task planners with continuous motion planners (Wolfe et al., 2010; Kaelbling and Lozano-Pérez, 2011; Srivastava et al., 2014). All of these are examples of general purpose, autonomous planning and execution frameworks. However, to operate a robot in the DARPA Robotics Challenge, teams were allowed to involve a human in the loop. We were inspired by the previously cited systems, but particularly interested in selecting an approach that allowed seamless integration of human authority into the plan and execute loop. One key aspect of such an integration is the ability to present the task plan as a sequence of sub tasks to the human operator with a visual representation that is easy to comprehend and control. We wanted our system to present the high level task hierarchy, as described in the previous door opening example, but the sequence should also interleave human operator tasks with robot tasks. Additionally, the operator should be able to pause, and skip over tasks, or resume a failed tasks after making a manual corrections (possibly by operating the robot in teleoperation for some period of time). We will describe the task autonomy design and user interface we came up with that allowed the human operator to work with the robot to carry out long, mixed-task missions as required by the DRC finals.

Robot task execution environments are available in other user interface systems. The Robot Task Commander is a visual programming language and IDE (Hart et al., 2014) designed to control the Robonaut-2 and Valkyrie platforms. The Affordance Template ROS Package for robot task programing (Hart et al., 2015) is an example of a planning framework based on affordance models that is integrated into the ROS software distribution. As another ROS based system (implemented using plugins to RViz), the Tartan Rescue DRC team described their operation interface that used task wizards to guide the operator through the interface steps required to operate the robot to complete a task (Stentz et al., 2015). The OpenRAVE environment provides a suite of motion planning algorithms, automation, and robotic applications (Diankov, 2010). Our user interface system shares similarities and capabilities with these systems, but it is distinguished by including a more powerful visualization system, integration with perception modules, and a scripting environment built into the user interface that enables rapid prototyping of perception, planning, and task execution behaviors.

1.3 Overview of Robot Capability and System

As a software track team, MIT competed in the DRC using the Atlas humanoid Robot supplied by DARPA and built by Boston Dynamics. With a particular research interest in dynamic control and bipedal locomotion, Team MIT developed a complete locomotion and whole-body manipulation system for low-level control of the robot that provided the advanced capability and reliability needed to complete the DRC tasks. In this section, we will describe at a high level how these capabilities shaped our approach and design. This paper complements our prior publication that describes our planning, control, and estimation algorithms in detail (Kuindersma et al., 2015).

Many of the tasks in the DRC required accurate positioning of end effectors for manipulation. After a series of calibrations, the robot could achieve repeatable reach execution to single centimeter accuracy with reliable trajectory tracking.[1] In March 2015, the robot's arms were upgraded to include 7 degrees of freedom (DoF) with an improved workspace but were limited to position tracking, as a result our manipulation was largely non-compliant with whole-body kinematic trajectories executed by the robot's controller. After testing a variety of robotic grippers preceding the DRC Trials, the team used the reliable Robotiq three fingered industrial gripper, whose underactuated three-finger design meant that manipulation would have limited controllable dexterity. Our motion planning algorithms supported collision-free motion planning in constrained environments. However, because collisions were rare, we found it was faster to remove collision constraints rely on visual inspection by the operator.

For locomotion we developed a stable and reliable dynamic walking gait with many advanced features such as toe-off (for stair climbing), heel and toe overhang (due to limited kinematic range) in addition to our whole-body vehicle egress strategy (described in Section 4). Due to the requirement for complete reliability, we used a conservative walking speed below our maximum of 0.45 m/s. After evaluation, we were confident to execute walking plans of 8–10 steps on the uneven walking terrain.

Finally, using the Carnegie Robotics Multisense SL sensor head, we created high precision 3D point clouds of the vicinity of the robot for autonomous object fitting and terrain estimation which required a 6 second sweep by the LIDAR sensor to collect data at our preferred resolution. This duration had a significant knock-on effect on our execution speed. High frequency LIDAR or active RGB would have had a significant positive effect on our approach.

1.4 Communication and Computation Layout

The robotic system is comprised of processes interacting with inter-process communication. The big picture organization of processes has not changed conceptually since the original design fielded in the Virtual Robotics Challenge (Tedrake et al., 2014). However, the final hardware upgrade of the Atlas prompted a new design of the mapping between processes and host machine. The upgraded robot carries on-board computation resources divided among three physical computers: *atlas0*, *atlas1*, and *atlas2*. In prior versions of the robot, all of our software processes ran off-board and sent low-level control messages to the robot via tethered fiber optic network. In the new design, processes are located either on the *robot side* (assigned to one of the on-board *atlas0*, *atlas1*, and *atlas2* hosts) or the *base side*, or operator station, also called the OCU (operator control unit). There is also a field computer, which sits within range of the robot's Wi-Fi radio and relays messages to the base side. The field computer does not host any processes besides message relay.

Processes communicate with message passing organized in a publish/subscribe model. Messages are broadcast to all processes within a broadcast community. The messaging system is based on the Lightweight Communications and Marshalling (LCM) library (Huang et al., 2010). A logical community may span multiple host computers and network interfaces. For example, processes on the robot side and base side may belong to the same community. However, during network interruption there will be a temporary partition-

[1] Although in Section 4 we will discuss our successful performance when robot damage lowered that precision.

ing within the community. The partitioning prevents messages from being transported between processes located on the robot and base side.

The on-board (robot side) processes include perception, planning, state estimation, control, and hardware drivers. The only processes that run on the base side are the user interface, planning server, and network shaper. The robot side processes include everything required to run the robot: in the case of a complete network disconnect between the robot Wi-Fi and field computer, the robot will continue standing, walking, or manipulating and then return to an idle wait state. The process organization is depicted in Table 1.

Control Computer	Perception Computer	OCU
Controller State Estimation Robot Driver	Sensor Drivers Perception Server Remote Planning	User Interfaces Motion Planning

Table 1: Computation was split between processes run remotely (i.e. on the robot) and interface and planning modules used at our operator control unit.

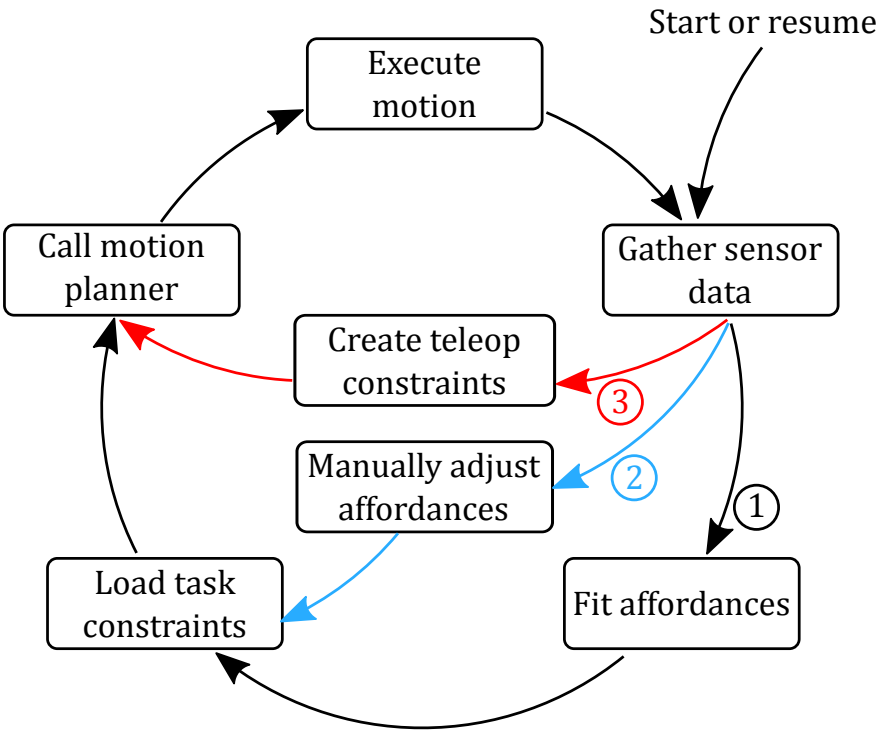

Figure 1: The operator workflow consists of a sense-plan-act loop augmented with operator input. During semi-autonomous operation, control proceeds along path 1, in which the operator needs only to supervise the state of the robot. Failures or errors at any stage may require the operator to switch to path 2, manually adjusting or creating affordances and their task-specific constraints. Further manual control is provided by path 3, in which the operator generates low-level teleoperation constraints such as specific end-effector positions or joint angles. After each executed motion, the operator has an opportunity to return to semi-autonomous operation along path 1.

2 User interfaces

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 test 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. The following sections will describe the two main user interface windows of Director that were used to control Atlas in the DRC finals: the task panel, and the application main window. During competition runs, the task panel is 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.

Figure 1 provides an overview of the task automation workflow. Task execution proceeds autonomously along the outer loop (1), but in the case of a failure the operator may proceed along the inner paths, providing input as (2) high level affordance adjustments or (3) a lower level teleoperation. Note that while proceeding in the autonomous execution loop, there may be steps where the system prompts the operator for input.

The software architecture of the interface, together with the appropriate routing of messages over LCM, provides the ability to adapt to different concepts of operations. In particular, it is possible to adapt the number of operators to the specific needs of the scenario at hand by changing the number of Director instances that are launched and connected to the stream of messages in the network. Using this approach, the system is scalable and allows having one or multiple operators working in parallel such as supervision of autonomous behaviors and interaction with the guided perception system and affordances. Each instance has the ability to be fully functional or can be restricted as convenient (for example, used as listener-only, which will provide complete visualization of the system but would not allow to execute any control action).

During the DRC finals competition, it was appropriate to have multiple operators to run the system. However, we note that the interface and architecture presented in this paper can be used in other types of missions that require the use of more limited resources that can be deployed in place in a very short time (Murphy, 2004), typically yielding to a single-screen OCU, such as the render safe procedures for explosive ordnance disposal (EOD).

Scripting: The user interface also embeds a Python programming environment. The entire interface is scriptable, and contains high-level programmatic interfaces to interact with the complete robot system. This is achieved through the software architecture: Director is a comprised of a set of core C++ libraries, and the majority of the library interfaces have bindings to the Python programming language. We have written a collection of Python modules that sit on top of the C++ libraries to provide a very high level interface to the entire robot system. Every action that can be performed in the user interface can also be executed from a Python script. For example, querying the walking planner and executing the result, or collecting point cloud data and invoking a model fitting algorithm. In addition to accessing the user interface elements from the Python environment, the user/programmer also has access to data objects. Sensor data such as images and point clouds, robot poses and trajectories, affordance models, frames and transforms, are all accessible and scriptable 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 automation routines that interfaced with the full robot system capabilities.

2.1 Director main window

The Director application main window is pictured in Figure 2. The main window contains a 3D visualization environment to draw robot 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

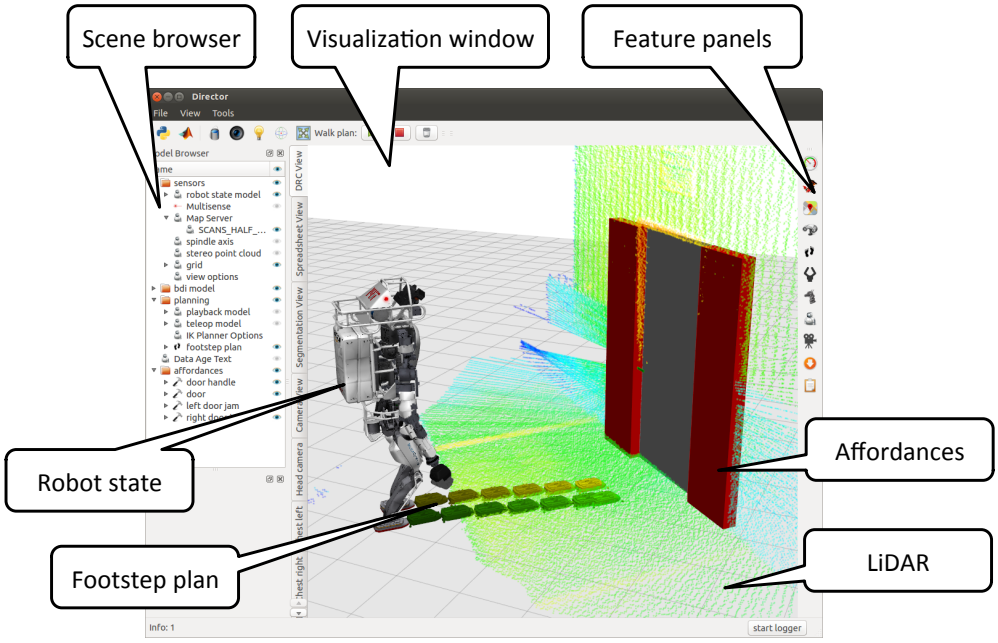

Figure 2: The Director main user interface.

Figure 3: The task panel interface.

health status of the system state. The interface also provides a teleoperation interface to support manual control by the operator.

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 (Schroeder et al., 2008). The key to 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 VTK-PCL-Python bridge library, we are able to prototype and debug new point cloud processing algorithms with seamless integration in the 3D visualization environment (Marion et al., 2012).

We also visualize models of the robot. Instances of the robot model are used to display the state estimation, the interactive teleoperation configuration, and animate manipulation plans. In Figure 2, we see a snapshot of a DRC Finals run: the robot stands with the door in front, the point cloud is drawn and an affordance model of the door is in view. 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.

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 minimal number of mouse clicks.

Teleoperation interface: One of the most frequently used feature panels is the *teleop panel*. Together with the visualization window, the teleop panel (shown in Figure 4) provides the operator with a rich set of controls to design whole-body manipulations 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 (Fallon et al., 2014). Together, the kinematic and quasi-static constraints allowed the operator to describe complex whole-body motions with changing contact states through the teleoperation interface.

2.2 Task panel

The task panel window is shown in Figure 3. The task panel contains a tabbed widget, with each tab holding the task plan for one of the eight tasks in the competition. The task panel can be initialized with other sets of panels, for example, panels for laboratory demos that are not relevant to the DRC competition. In the competition, the primary operator had the task panel opened full screen on one monitor, and the Director main window opened full screen on a second monitor. As long as there are no failures in execution, the task panel occupies 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.

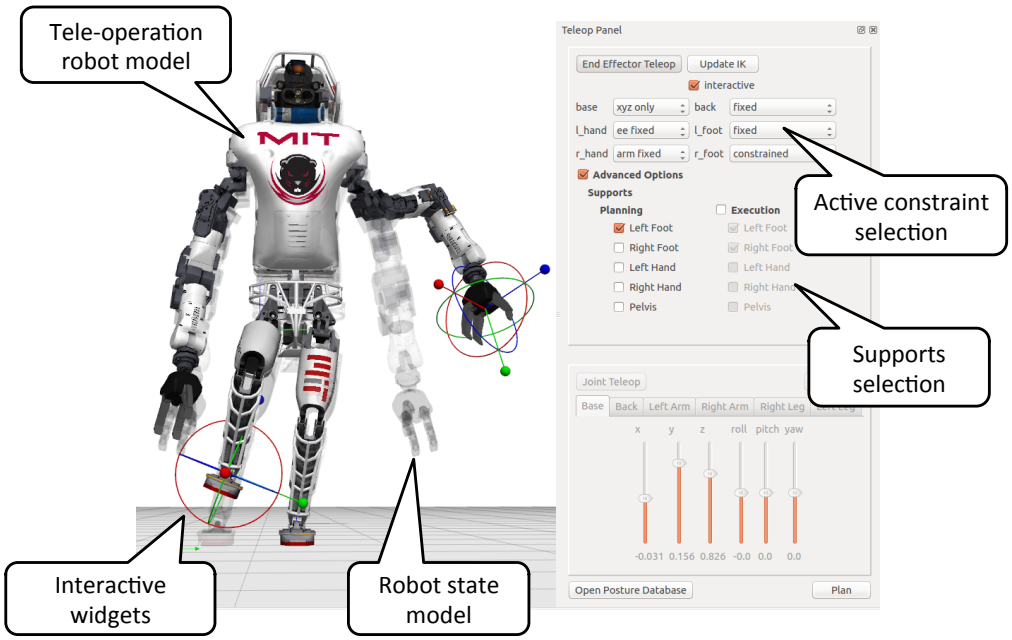

Figure 4: The teleop interface provides the user with a rich set of preset constraints that are adjusted with widgets in the visualization window. The robot's current configuration is shown as a translucent model, and its desired configuration is shown as the full opaque model. In this example, the operator 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.

When the system asks an input from the operator the requested input is clearly stated and designed so that it is not sensitive 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.

Layout: we describe the layout of the task panel user interface. Please refer to Figure 3. Tabs across the top switch between 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. Below, the interface were 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 (on the left side) were used to allow easy switching between the arm used in manipulation, choosing a valve radius or a fitting algorithm. 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 current situation. Section 4 describes such a situation

that arose during the Day 1 competition run where the manual controls were necessary to cope with the situation. Section 3.2 describes the programming model underlying the task panel user interface.

3 Programming models

This section describes the programming models we used to implement our approach to task autonomy, combining a human operator and a user interface with a task planning system. In addition to user interfaces, well designed programmer interfaces are a key component to successful shared autonomy. These interfaces allowed for easy and rapid extension of the interface by many members of the team during the development and testing of the complete system.

3.1 Affordance model

When a physical object had a specific meaning or was to be interacted with, it was specifically denoted as an *affordance*, which combined a 3D model of the object with metadata describing the specific meaning or modes of interaction. One such set of metadata is a set of 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 planner 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.

The location of an affordance or one of its constituent frames may be updated by a process living anywhere in the user interface's LCM community and synchronized between simultaneously running user interfaces. Because the affordances were synchronized in this manner, it was straightforward to parallelize piloting tasks across multiple operators when the need arose: for example one operator could guide the robot through a task queue for walking while another refined the positioning of a valve or drill using the latest perceived point cloud.

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 3D visual rendering applications. Affordances, for example, a type of object in the object model. The object model also stores a large variety of other object types, for example, data objects such as point clouds, terrain maps, motion plans, footstep plans. The 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.

3.2 Tasks

Each step in a task execution is an object of the Task class type. As an example, here is a typical sequence of task executions and the classes used: *fit drill, approve drill, plan reach, approve manipulation plan, execute plan, wait for execution, close gripper*. The name *plan reach* actually referring to a specific custom planning function, for example, 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 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 are built in pauses that give the user the opportunity to adjust an input to the system without having to intervene and pause execution explicitly. 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 robot status messages published 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.

Asynchronous task queue Task execution is brokered by an asynchronous task queue (ATQ). Each item in the queue is a task. A task is a standalone unit capable of performing some action(s) and returning success or failure. At any point in time, only the top task in the queue is active, and it may complete at some point in the future. If a task raises an error then the execution of the task queue is paused, and the task panel user interface displays this state to the user. When the task queue resumes it will attempt to re-execute the failed task from the beginning, unless a different task has been selected in the user interface. A program may construct and begin execution of any number of asynchronous task queues. For example, as described in Section 2.2, each tab of the task panel owns maintained its own ATQ, but the task panel ensures only one queue is executed at a time. We found that it was useful to leverage the ATQ model in other locations of the user interface as well, to perform auxiliary operations. Rather than adopting a thread model, many actions in the user interface are dispatched (queued) for asynchronous execution. We implemented a asynchronous task queue ourselves in our application library so as to maximize simplicity and control over the design, however the concepts are borrowed from the field of asynchronous programming, and implemented in production systems such as Celery (an open-source Python implementation), and often found in job schedulers and web servers.

3.3 Guided perception

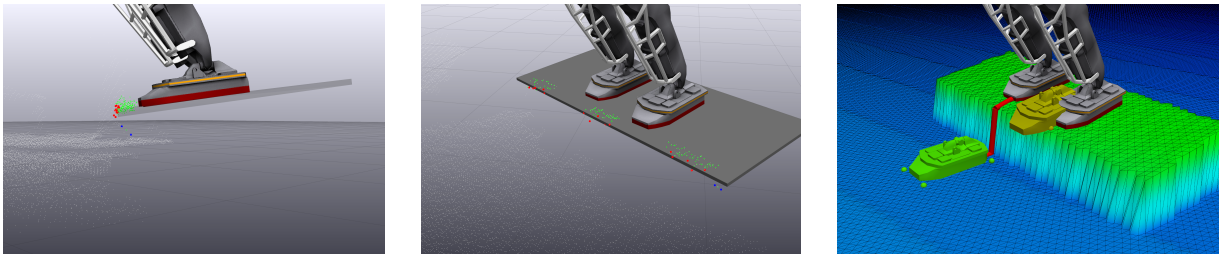

Figure 5: Fitting the running board platform used for vehicle egress. Left and Center show stages of the fitting algorithm. Red and blue points are candidate edge points with the red points being inliers. From the edge the pose of the platform affordance (in gray) is produced. Right shows the affordance being used to automatically place a precise overhanging footstep and a swing trajectory.

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 3.1. 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 camera images because it required less precision than point cloud annotations.

Some fitting algorithms may succeed with high probability without operator provided search regions, but operator input can still be useful to approve the 3D pose of automatically fit affordances prior to continuing with planning based on the affordance models. We can also provide guidance by designing task specific

perception algorithms, where some intuition about the task is encoded into the algorithm.

As an illustrative example, we discuss the fitting of a Polaris running board model to point cloud data during the vehicle egress task (Figure 5). During the egress task the robot steps out of the car from a running board platform 20 cm above the ground onto the ground as shown in Figure 7. The foot swing trajectory must be precisely planned relative to this platform. To accomplish this procedure, we modeled the platform as an affordance using the front edge.

Algorithm 1 Platform fitting algorithm

1: **function** FitPlatform(q, p)
2: $f_s \leftarrow$ stanceFrame(q)
3: $x, y, z \leftarrow$ axes(f)
4: $p \leftarrow$ removeGround(p)
5: $p \leftarrow$ cropToBox($p, f_s, [1, 1, 0.1]$)
6: $p_e \leftarrow$ computeEdge(p, y, x)
7: $l \leftarrow$ fitLineRansac(p_e)
8: $l_s \leftarrow$ projectToPlane(l, position(f), z)
9: **return** frameFromPositionAndAxes(midpoint(l_s), cross(l_s, z), l_s, z)

Algorithm listing 1 gives a sketch of the point cloud processing procedure to fit the pose of the platform affordance. We describe the steps of the algorithm in more detail to give the reader some idea of our approach to point cloud processing. Our processing algorithms combine task specific filtering and with existing techniques such as RANSAC model fitting and normal estimation as provided by the Point Cloud Library (Rusu and Cousins, 2011). The goal is to fit the edge of the running board with high accuracy, since the edge was to be used for footstep placement as described above. As seen in Figure 5 very few LIDAR point returns are collected from the platform due to occlusion by the robot body; only the front edge is visible. We describe each line of the fitting algorithm:

1. q is the robot configuration, p is a point cloud snapshot collected from the LiDAR sensor.

2. stanceFrame() returns a homogeneous transform at the midpoint between the two feet.

3. Extract the coordinate axes of the stance frame. We can assume that the Z axis is normal to the platform, and the X axis points roughly toward the edge of the platform.

4. Filter the point cloud to remove points on the ground, using a planar ground model.

5. Crop the input point cloud to a box of dimensions [1,1,0.1] meters, oriented and centered at the stance frame. These points are displayed in green.

6. Bin the points into bands roughly perpendicular to the edge. This computes a reduced set of candidate platform edge points, displayed in red and blue.

7. Fit a line model to the edge points. Inliers are displayed in red, outliers are blue.

8. Project the line into the platform frame (defined by the stance frame).

9. Return a homogeneous transform which defines the platform affordance.

The previously described algorithm is well suited to our shared autonomy model. It is written with some assumption of the initial pose, which is satisfied by the task sequence leading up to the invocation of the algorithm and the fit result is easily inspected by an operator. The fitting algorithms for other affordances are implemented with a similar approach.

4 Performance Evaluation at the DARPA Challenge 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 and user interface. Figure 6 presents a selection of task execution times collected during the two competition runs. On tasks where the shared autonomy system proceeded without interruption, execution times are consistent over both runs, but tasks that required manual operator intervention were slower: for example the door task on Day 1, and the terrain task on Day 2.

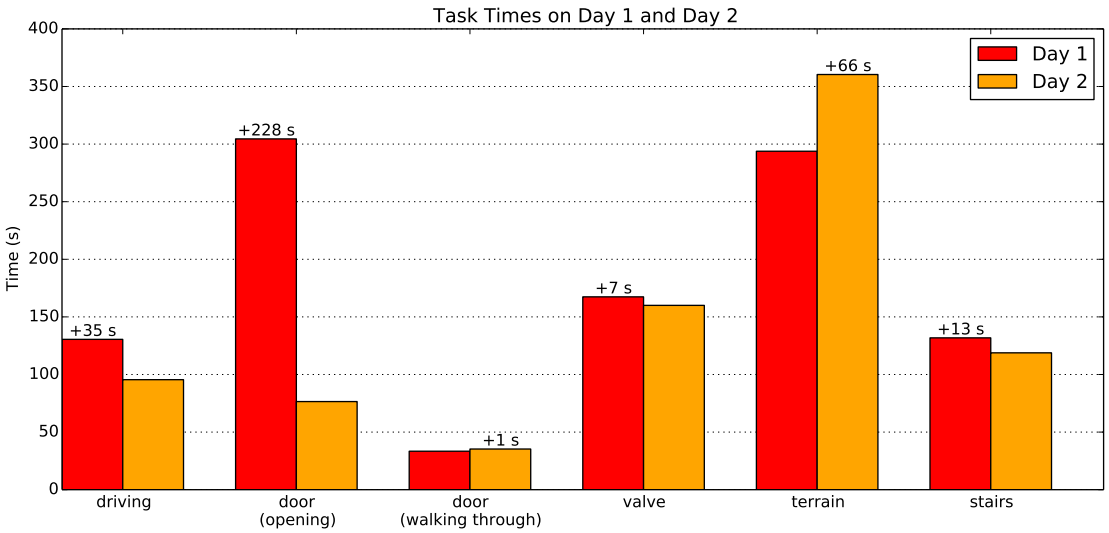

Figure 6: Timing comparison between tasks executed in competition days Day 1 and Day 2. Task completion success varied over the two days so we cannot directly compare all eight tasks; egress and drill are not shown due to lack of completion on at least one day. The door task required manual intervention of Day 1, and terrain required manual intervention on Day 2, and took correspondingly longer.

Driving and egress tasks: The first two tasks of the competition are driving and vehicle egress. To drive the vehicle the operator provided steering and throttle inputs based on video feedback (which had little delay). The stereo depth cameras were invaluable to inform the operator about vehicle speed and trajectory around obstacles. Driving was 35 seconds faster on the second day run, which we attribute to improved operator performance. After completing the driving course the operator changes task panel to the egress task. The egress procedure is highly automated and the operator's role is simply to approve automatically generated plans. These steps varied in complexity from opening the robot's grippers, to changing the active contact points used by the controller to balance.

During preparation both tasks were completed many times. Optimizing our semi-autonomous task sequence allowed us to halve the time for egress in the two weeks before the competition. However, a flaw in the design of our system meant that on Day 1 an important step which was required between the driving and egress tasks was not completed: disabling the ankle throttle controller. This resulted in the ankle being incorrectly positioned when moved from the pedal onto the foot well. The operators realized the error and attempted to correct it manually, but ultimately it led to instability when the robot attempted to stand up, leading to a fall from the vehicle.This issue was not observed previously because space constraints in our laboratory prevented the two tasks from being tested consecutively.

Before Day 2, a task was added to the sequence to prevent the error from occurring again. With this simple fix the egress worked exactly as planned on our second run.

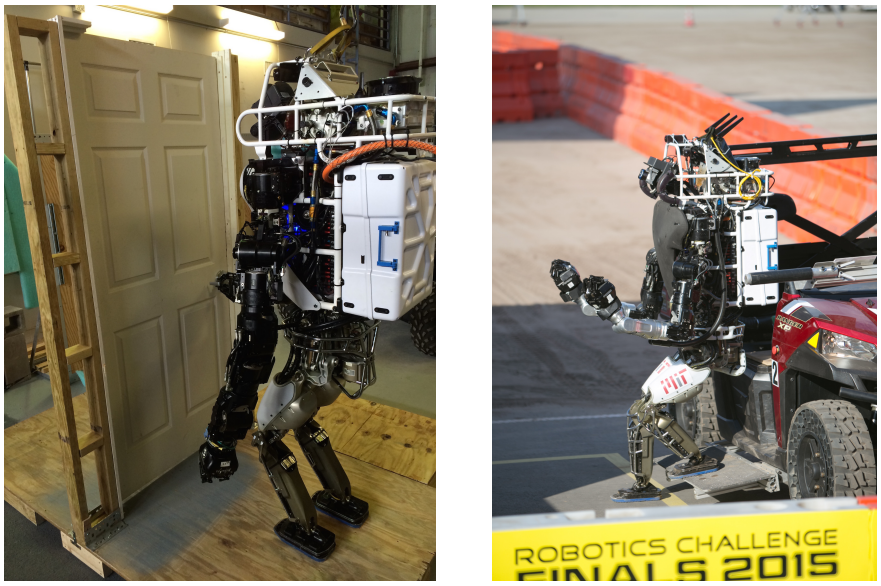

Figure 7: Atlas robot performing the door task in our laboratory (left). Egressing from the vehicle, task 2, at the DRC finals.

Door task: The door task involved opening and walking through an industrial doorway. The task interface encoded the sequence: walking to a carefully aligned location in front of the door; turning the handle; pushing the door open with a second hand; and finally walking through the door along a path carefully chosen to ensure clearance between the robot's wide shoulders and the door frame.

The fall from the car on Day 1 destroyed an actuator on our favored right arm. However, the task execution system exposed the flexibility afforded by our planning systems and affordance models, and allowed us to switch the handedness of all single-handed plans with a single toggle switch. It also allowed us to skip over stages that would not work with the broken wrist joint, such as pushing the door open, while still automating the other aspects. Figure 6 demonstrates that a much greater amount of time was required on Day 1 during the manipulation stage of the door task due to operator teleoperation, which was required because of damaged sustained to precision encoder sensors in the arm during the fall. However, the timing of the non-manipulation aspect of the door task (walking through the door) was consistent due to a return to the automated task squence. The entire door task was quickly executed achieved without deviation from the autonomy script on Day 2.

Valve task: The valve task was the first task after entering the building through the door. The network blackout rules begin immediately upon crossing the threshold of the door. Our recorded execution time for the valve includes the approach to the valve starting from just after the door. Due to the broken right wrist actuator, we turned the valve using the robots left end-effector on Day 1, but preferred the right end-effector on Day 2. Execution is similar, but the robot's planned stance location in front of the valve will depend on the handedness selected. The valve task sequence ran without operator intervention on both days, and execution time was consistent between the runs, with a small difference of only 7 seconds.

Drill task: The procedure to turn on the drill requires bihanded manipulation and was the most complex task. The steps include: pick up the drill from a shelf with one hand, turn it on with the other, cut a circle, knock out the circle to create a hole, drop the drill. Each sequence requires several planned trajectories and updates to the affordance models using perception data.

The task was skipped during the Day 1 run because of the broken wrist actuator meant that we could not

turn the tool on. On Day 2 our planned task sequence performed very successfully. We used stereo visual servoing to press the drill button by successively minimizing the distance between the drill button (held by the right hand) and the thumb on the left hand. The human provided the precise location of the button by clicking on a camera image.

We then walked to the wall and started cutting, however we failed to cut the wall deeply enough to disconnect the circle. After inserting the drill into the wall as a temperature sensor indicated that the wrist actuator was overheating requiring the operator to half operation to pause to allow it to cool. While we believe that this was unrelated to the missed cut, it meant that we ran out of time to re-attempt the drill task (which shut off after 5 mins).

The cutting trajectory included significant use of back motion as the shelf used to store the drills was close to the cutting wall. We believe that this execution was imprecise as a result and that a more compliant control strategy, such as (Sentis et al., 2010), would have achieved a more robust cut. In retrospect, comparing to approaches taken by other teams our bihanded approach was quite complicated and had a roll to play in our failure to achieve this task each day.

Finally when dropping the drill to move to the next task, we uncovered a bug in our system that was unknown despite significant amounts of randomized testing. The motion plan during this sequence caused the robot to twist its pelvis around a yaw of 180 degrees. This exposed a simple wrap-around bug (where 180 degrees became -180) which caused a control instability and a fall.

Surprise, terrain and stairs tasks: The surprise task was completed using a teleoperation mode to press a lever on Day 1, while on Day 2 we were behind time and skipped the surprise task (a plug insertion). We moved on to the mobility tasks of crossing an uneven terrain course and ascending a staircase. Our research group is specifically focused on locomotion and on each day we successfully completed the tasks. From our analysis we were the quickest of the teams to complete these tasks.

Unlike the manipulation tasks, where affordances are used as virtual representation of the physical objects to be manipulated, in the terrain and stairs tasks, affordances are used to represent the support surfaces for locomotion, such as the blocks in the terrain (shown in Figure 8) and the levels in the stairs. In this case, the guided perception module has a fundamental role in computing the segmentation of these objects from the point clouds and automatically fitting affordances to the relevant parts. The operator had the ability to perform modifications on the position and orientation of the affordances if needed. The task hierarchy consisted of fitting support surface affordances to the sensed data, loading pre-planned footsteps relative to those affordances, and then executing those footsteps. We had previously demonstrated the ability to autonomously plan footsteps over unmodeled terrain (Deits and Tedrake, 2014; Fallon et al., 2015b), but because the exact layout of the DRC terrain was known in advance, we chose to pre-plan footsteps for the given terrain.

Each subtask in the hierarchy also involved a check for operator approval, but during the competition almost no operator intervention was required. During the terrain task on Day 1 and the stairs task on both days, the operator interaction was limited to minor (roughly 1 cm) adjustments of the perceived positions of the locomotion affordances. During the terrain task on Day 2, our approach was more cautious and a little slower. Our operator observed a foot clipping a block while crossing the terrain and triggered the robot to take a recovery step (replanning online) which stabilized the robot before we continued. The operator then planned two manually-chosen footsteps and then resumed execution of the scripted tasks.

5 Conclusion

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 is focused on the development and field testing of

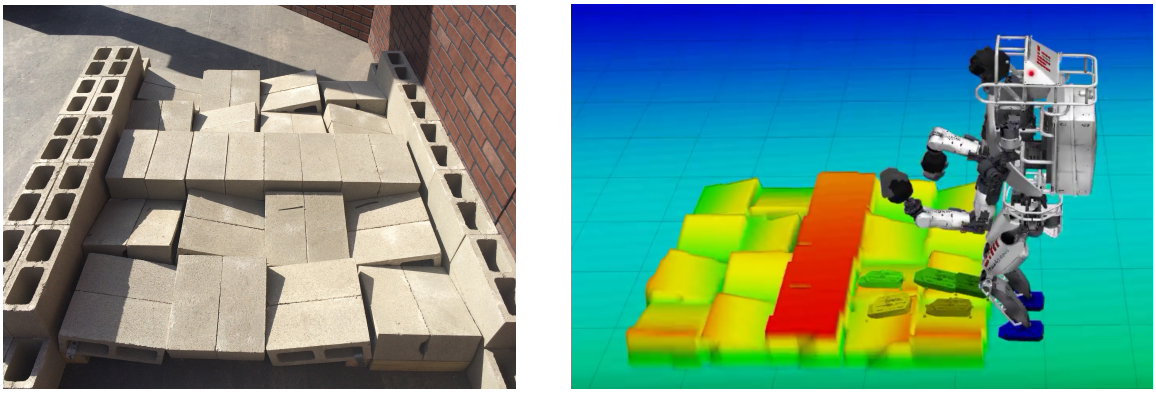

Figure 8: The terrain course at the DRC Finals (left). The terrain model as estimated by the perception system and represented to the planning system as a height map (right).

a user interface that supports the execution of complex manipulation and locomotion tasks by alternating autonomous behaviors represented in a task hierarchy with the operator's input for supervision of the system and teleoperation of the robot if necessary. The teleoperation mode counts with the ability to specify a reach set of constraints and whole-body behaviors that empower the operator with the ability to perform procedures using a high-DoF robot that were not possible with previous technology that has been deployed in the real mission, which is largely based on joint-by-joint teleoperation of low-DoF robots. A comparison of task execution in the field during the competition is limited to two runs but it shows a consistent indication that using a larger portion of autonomy allowed task completion is less time compared to manual intervention.

On both days of competition, the robot successfully completed the course in under 60 minutes, though not all tasks were completed for reasons discussed in the previous section. 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 lessons we learned.

We have released all of the software described in this article under an open-source license[2]. The software has been generalized to support a variety of robots and continues to be used by a growing community of users after the DRC project.

Acknowledgments

We gratefully acknowledge the support of the Defense Advanced Research Projects Agency via Air Force Research Laboratory award FA8750-12-1-0321, and the Office of Naval Research via award N00014-12-1-0071. We are also grateful to the team's many supporters both inside and outside MIT (listed at http://drc.mit.edu), including our families and friends.

We are also grateful to Boston Dynamics, Carnegie Robotics, the Open Source Robotics Foundation, Robotiq, iRobot Corporation, and Sandia National Laboratories for their support during the DRC.

Finally, we acknowledge our late colleague, advisor, and friend, Seth Teller, whose leadership and ideas contributed immensely to this work.

[2]http://github.com/RobotLocomotion/director.git

References

Bagnell, J. A., Cavalcanti, F., Cui, L., Galluzzo, T., Hebert, M., Kazemi, M., Klingensmith, M., Libby, J., Liu, T. Y., Pollard, N., et al. (2012). An integrated system for autonomous robotics manipulation. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2955–2962. IEEE.

Bohren, J., Rusu, R. B., Jones, E. G., Marder-Eppstein, E., Pantofaru, C., Wise, M., Mösenlechner, L., Meeussen, W., and Holzer, S. (2011). Towards autonomous robotic butlers: Lessons learned with the pr2. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5568–5575. IEEE.

Deits, R. and Tedrake, R. (2014). 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.

Diankov, R. (2010). *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute.

Dragan, A. and Srinivasa, S. (2012). Formalizing assistive teleoperation. In *Robotics: Science and Systems (RSS)*.

Dragan, A. D. and Srinivasa, S. S. (2013). A policy-blending formalism for shared control. *International Journal of Robotics Research*, 32(7):790–805.

Fallon, M., Kuindersma, S., Karumanchi, S., Antone, M., Schneider, T., Dai, H., D'Arpino, C. P., Deits, R., DiCicco, M., Fourie, D., Koolen, T. T., Marion, P., Posa, M., Valenzuela, A., Yu, K.-T., Shah, J., Iagnemma, K., Tedrake, R., and Teller, S. (2014). An architecture for online affordance-based perception and whole-body planning. *J. of Field Robotics*.

Fallon, M., Kuindersma, S., Karumanchi, S., Antone, M., Schneider, T., Dai, H., D'Arpino, C. P., Deits, R., DiCicco, M., Fourie, D., Koolen, T. T., Marion, P., Posa, M., Valenzuela, A., Yu, K.-T., Shah, J., Iagnemma, K., Tedrake, R., and Teller, S. (2015a). An architecture for online affordance-based perception and whole-body planning. *Journal of Field Robotics*, 32(2):229–254.

Fallon, M. F., Marion, P., Deits, R., Whelan, T., Antone, M., McDonald, J., and Tedrake, R. (2015b). Continuous humanoid locomotion over uneven terrain using stereo fusion. In *Humanoid Robots, 2015. Humanoids 2015. 9th IEEE-RAS International Conference on*. IEEE.

Hart, S., Dinh, P., and Hambuchen, K. (2015). The affordance template ros package for robot task programming. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 6227–6234.

Hart, S., Dinh, P., Yamokoski, J. D., Wightman, B., and Radford, N. (2014). Robot task commander: A framework and ide for robot application development. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1547–1554. IEEE.

He, K., Lahijanian, M., Kavraki, L. E., and Vardi, M. Y. (2015). Towards manipulation planning with temporal logic specifications. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 346–352. IEEE.

Huang, A., Olson, E., and Moore, D. (2010). LCM: Lightweight communications and marshalling. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan.

Jain, S., Farshchiansadegh, A., Broad, A., Abdollahi, F., Mussa-Ivaldi, F., and Argall, B. (2015). Assistive robotic manipulation through shared autonomy and a body-machine interface. In *Rehabilitation Robotics (ICORR), 2015 IEEE International Conference on*, pages 526–531.

Kaelbling, L. P. and Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1470–1477. IEEE.

Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J. (2007). Where's waldo? sensor-based temporal logic motion planning. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3116–3121. IEEE.

Kuindersma, S., Deits, R., Fallon, M. F., Valenzuela, A., Dai, H., Permenter, F., Koolen, T., Marion, P., and Tedrake, R. (2015). Optimization-based locomotion planning, estimation, and control design for atlas. *Autonomous Robots*. To appear.

Marion, P., Kwitt, R., Davis, B., and Gschwandtner, M. (2012). Pcl and paraview—connecting the dots. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 80–85. IEEE.

Muelling, K., Venkatraman, A., Valois, J.-S., Downey, J., Weiss, J., Javdani, S., Hebert, M., Schwartz, A. B., Collinger, J. L., and Bagnell, J. A. (2015). Autonomy infused teleoperation with application to bci manipulation. In *Robotics: Science and Systems (RSS)*.

Murphy, R. (2004). Human-robot interaction in rescue robotics. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(2):138–153.

Nehaniv, C. L. and Dautenhahn, K. (2002). The correspondence problem. In *Imitation in Animals and Artifacts*, pages 41–61, Cambridge, MA, USA. MIT Press.

O'Brien, B., Stump, E., and Pierce, C. (2010). Effects of increasing autonomy on tele-operation performance. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1792–1798.

Rusu, R. B. and Cousins, S. (2011). 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE.

Rusu, R. B., Şucan, I. A., Gerkey, B., Chitta, S., Beetz, M., and Kavraki, L. E. (2009). 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.

Schroeder, W. J., Lorensen, B., and Martin, K. (2008). *The visualization toolkit: an object-oriented approach to 3D graphics*. Kitware, 4th edition edition.

Sentis, L., Park, J., and Khatib, O. (2010). Compliant control of multicontact and center-of-mass behaviors in humanoid robots. *IEEE Trans. Robotics*, 26(3):483–501.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 639–646. IEEE.

Stentz, A., Herman, H., Kelly, A., Meyhofer, E., Haynes, G. C., Stager, D., Zajac, B., Bagnell, J. A., Brindza, J., Dellin, C., et al. (2015). Chimp, the cmu highly intelligent mobile platform. *Journal of Field Robotics*, 32(2):209–228.

Tedrake, R., Fallon, M., Karumanchi, S., Kuindersma, S., Antone, M., Schneider, T., Howard, T., Walter, M., Dai, H., Deits, R., et al. (2014). A summary of team mit's approach to the virtual robotics challenge. *IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China*.

Wolfe, J., Marthi, B., and Russell, S. J. (2010). Combined task and motion planning for mobile manipulation. In *ICAPS*, pages 254–258.

Yamauchi, B. M. (2004). Packbot: a versatile platform for military robotics. In *Proc. SPIE 5422, Unmanned Ground Vehicle Technology*, volume 5422, pages 228–237.