Preprint of the paper which appeared in Journal of Field Robotics,
Volume 32, Issue 3, pages 352-377, May 2015,
First published online 4 Dec 2014.

Human-Robot Teaming for Rescue Missions: Team ViGIR's Approach to the 2013 DARPA Robotics Challenge Trials

Stefan Kohlbrecher, Alberto Romay, Alexander Stumpf, Anant Gupta, Oskar von Stryk
Simulation, Systems Optimization and Robotics Group, CS Dept. Technische Universität Darmstadt
Hochschulstrasse 10, 64289, Darmstadt, Hesse, Germany
{kohlbrecher,romay,stumpf,gupta,stryk}@sim.tu-darmstadt.de

Felipe Bacim, Doug A. Bowman
Center for Human-Computer Interaction, Virginia Tech
2202 Kraft Drive, KWII Building (0106), Blacksburg, VA 24061-0106, United States
{fbacim,dbowman}@vt.edu

Alex Goins, Ravi Balasubramanian
Robotics and Human Control Systems Lab, Oregon State University
1891 SW Campus Way, Dearborn Hall, Corvallis, OR 97331, United States
goinsa@onid.orst.edu, ravi.balasubramanian@oregonstate.edu

David C. Conner*
TORC Robotics
2200 Kraft Drive Suite 2050, Blacksburg, VA 24060, United States
conner@torcrobotics.com

Abstract

Team ViGIR entered the 2013 DARPA Robotics Challenge (DRC) with a focus on developing software to enable an operator to guide a humanoid robot through the series of challenge tasks emulating disaster scenarios. The overarching philosophy was to make our operators full team members and not just simple supervisors. We designed our operator control station (OCS) to allow multiple operators to request and share information as needed to maintain situational awareness under bandwidth constraints, while directing the robot to perform tasks with most planning and control taking place onboard the robot. Given the limited development time we leveraged a number of open source libraries in both our onboard software and our OCS design; this included significant use of the Robot Operating System (ROS) libraries and toolchain. This paper describes the high level approach, including the OCS design and major onboard components, and describes our DRC Trials results. The paper concludes with a number of lessons learned that are being applied to the final phase of the competition.

1 Introduction

In the spring of 2012, the United States Defense Advanced Projects Research Agency (DARPA) proposed the DARPA Robotics Challenge (DRC)[1] to accelerate development and evaluation of disaster response robots that have the capability for early response and mitigation of disasters. This effort was partly motivated by the earthquake and tsunami that struck the Tohoku region of eastern Japan on March 11, 2011, and led to subsequent damage to the Fukushima Daiichi nuclear plant. The DRC is designed to mimic the tasks that might be required of a robot (Nagatani et al., 2013) to respond to the initial damage and avert subsequent catastrophes.

A major focus of the DRC is the development of an approach that leverages the complementary strengths and weaknesses of the robot system and human operator(s). While full bandwidth and update rate access to all sensory systems is available onboard the robot system, cognitive and decision making abilities of a human operator are still vastly superior for the foreseeable future. This is especially true for disaster scenarios, as only very limited assumptions about their structure can be made beforehand, in contrast to structured man-made scenarios like kitchens, where semantic knowledge of their structure can be leveraged for highly autonomous robots operating in them (Blodow et al., 2011). Team ViGIR was formed with a particular focus on developing the enabling technologies for the human-robot team.

There are a number of schemes for classifying the autonomy of a system(Bruemmer et al., 2002; Scholtz, 2003; Yanco, 2004; Desai and Yanco, 2005; Huang et al., 2007). In our system, the robot is never fully autonomous and therefore has a low score for *human independence*(Huang et al., 2007). The human members of the team function as *supervisors* who set high level goals, *teammates* who assist the robot with perception tasks, and *operators* who directly change robot parameters to improve performance(Scholtz, 2003); as these roles change dynamically during a set task in our system, we will use the term *operator* generically. Following (Bruemmer et al., 2002), we rarely operate in *teleoperation* where we directly control a joint value, and primarily operate in *shared* mode where the operator specifies tasks or goal points, and the robot plans its motions to avoid obstacles and then executes the motion only when given permission. Even when executing a footstep plan in *autonomous* mode, the operator still has supervisory control of the robot and can command the robot to stop walking at any time, and safely revert to a standing posture.

Team ViGIR entered the DRC as a 'Track B' team competing in the DARPA Virtual Robotics Challenge (VRC). Initially, Team ViGIR was composed of TORC Robotics[2], the Simulation, Systems Optimization and Robotics group at Technische Universität Darmstadt (TUD)[3], and the 3D Interaction Group at Virginia Tech[4]. With only 8 months from program kick-off to the first competition, the team focused on providing basic robot capabilities needed for the three tasks in the VRC. A short overview of our VRC approach is available in (Kohlbrecher et al., 2013).

While the tasks and requirements for the VRC were based on those anticipated in a real scenario, there were important differences: sensor noise was low and more predictable, simple friction models were used, there was no need for calibrating sensors or joint angle offsets for the robot, and the environments were known ahead of time. The dynamic model used for simulating the Atlas robot was available to the teams, enabling the use of model based control approaches without prior model/systems identification. In spite of these simplifications, Team ViGIR chose to take the long view and develop our human-robot interface (HRI)(Goodrich and Schultz, 2007) with an eye toward operation in more complex and unknown environments. The VRC results demonstrated that our approach and developed capabilities were competitive, enabling us to place sixth out of 22 qualifying teams that participated in the VRC competition. While some approaches, such as our technique for getting into the car, were not transferable to a real robot, the overall approach and interaction philosophy was readily transferable. This included our basic design concept for the operator control station (OCS), the footstep and manipulation planners, our approach to grasping, and our low bandwidth

[1] http://theroboticschallenge.org
[2] http://www.torcrobotics.com
[3] http://www.sim.informatik.tu-darmstadt.de/en/index/
[4] http://research.cs.vt.edu/3di/

communication software.

After our success in the VRC, Team ViGIR received the Atlas humanoid robot shown in Fig. 1a, which was developed by Boston Dynamics, Inc. (BDI). With only four months to modify our software and learn the capabilities and intricacies of the robot before the DRC Trials, Team ViGIR added representatives from the Robotics and Human Control Systems Lab at Oregon State University[5] to supplement our experience with real world grasping and manipulation. As a joint US-European team, with a nine hour time differential across the team, a crucial part of our successful participation in the DRC was establishing a workflow that enabled seamless cooperation over large distances. To achieve this, the team relied on the Redmine (Koch, 2010) open source project management application. Redmine provided both a wiki for documentation, an issue tracker as well as integration with the *Git* (Loeliger and McCullough, 2012) repositories containing source code. Frequent teleconferences were used to coordinate among team members.

As development time between initial kick-off and the DRC Trials was limited, development was split into reliable core modules that were crucial to the ability to perform the competition tasks and additional modules that could potentially improve performance but also carried a greater risk of failing due to environment conditions or limited time for testing.

There are a number of publications coming from DRC research, including other competitors in the VRC (Koolen et al., 2014; Tedrake et al., 2014), as well available summaries of DRC Trials results (Fallon et al., 2014a). There are also publications on specific system components like footstep planning (Deits and Tedrake, 2014; Stumpf et al., 2014), drift free state estimation(Fallon et al., 2014b), whole body motion planning (Dai et al., 2014), and user-guided manipulation (Alunni et al., 2013).

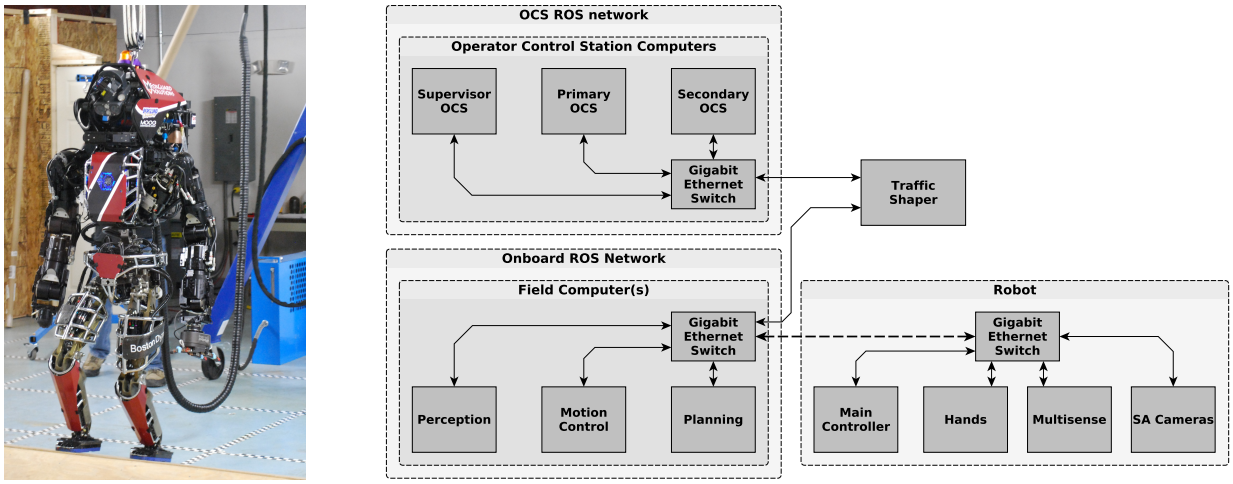
In this paper we present an overview of our system and some of our approaches that were developed, including some that were evaluated but could not be successfully integrated in time for the competition. Section 2 presents an overview of the system architecture with a focus on the hardware layout and communications. In Section 3, we discuss our philosophy and design of the the HRI. Perception and sensing of robot state and situational awareness is discussed in Section 4. Next, we discuss the robot control and planning systems in Sections 5 and 6, followed by a discussion of grasping in Section 7. The paper discusses our results at the DRC Trials in Section 8, offers lessons learned in Section 9, and concluding thoughts in Section 10.

2 System Architecture

The system hardware is composed of three major sub-systems as shown in Fig. 1: the robot, the "onboard" computers, and the OCS hardware. Team ViGIR uses the Atlas robot that includes a robot control computer and Ethernet switch that handles communications with the sensors and robot hands. The "onboard" or "field" computers run the robot control and perception software developed by Team ViGIR. In the future, these field computers will be carried onboard the robot itself, but for ease of development leading up to the DRC Trials DARPA allowed these to be separate computers connected to the robot via a 10 GB/second fiber optic network connection. During the DRC Trials, the onboard computers were connected to the OCS computers via a 1 GB/second network connection that passed through a network traffic shaper; the traffic shaper introduced communication restrictions intended to mimic the effects of poor wireless communications. All operator interactions with the robot occurred through the OCS hardware, with commands sent to the onboard software via the traffic shaper connection.

Early on, the team chose to base the system software on the open source Robot Operating System (ROS)[6] (Quigley et al., 2009) to take advantage of the available libraries and infrastructure. The system communications use ROS as the middleware, with two separate ROS networks being used during competition. One ROS network handles communications between the OCS computers, while the second ROS network handles

[5] http://mime.oregonstate.edu/research/rhcs/
[6] http://www.ros.org



(a) (b)

Figure 1: System overview: a) robot with tether connecting to the base station in the background, b) architecture diagram including the robot, Onboard, and OCS sub-systems. The dashed arrow shows the 10 gigabit fiber optic link between robot and field switch, while other arrows indicate 1 gigabit Ethernet links.

communications on the onboard computers. Section 2.4 describes the motivation for this choice, and the implementation of our Communications Bridge (CommsBridge) between the ROS networks. To be able to leverage latest open source software, we closely followed the ROS release schedule and used the ROS Hydro version during the DRC Trials. With ROS changing the underlying build system to *catkin*, the team uses a two workspace setup with one *rosbuild* workspace for packages using the legacy *rosbuild* build system and one workspace using the new *catkin* build system. Using this setup allows the most flexibility in using both old and new open source software for ROS.

The remainder of this section gives an overview of each major sub-system, with a focus on hardware layout.

2.1 Robot Hardware

The Atlas system is a hydraulically actuated anthropomorphic robot developed by Boston Dynamics Inc. (BDI). The robot has 28 actuated degrees of freedom (DOF), stands 1.88m tall and weighs approximately 150kg. There are 6 DOF per arm and leg, 3 DOF for torso motion, and one DOF for head pitch motion. For the DRC Trials, the robot was used in a tethered configuration as shown in Fig. 1, with the tether providing electrical power, coolant flow, and a 10 gigabit Ethernet connection .

The robot is equipped with a number of sensors. The main external sensor is a Carnegie Robotics[7] Multisense SL sensor mounted as the head. This sensor uses both a Hokuyo UTM-30LX-EW LIDAR mounted on a slip ring for continuous rotation and a stereo camera system that performs stereo disparity computation on a onboard FPGA. Both sensors are calibrated against each other using a calibration approach supplied by the manufacturer. Additionally, there are two situational awareness (SA) cameras mounted below the head with high field of view fisheye lenses. The robot provides a pose estimate based on an internal IMU and internal joint sensing.

For the DRC Trials, the robot was supplied with three options for hands (see Fig. 2a, 2b, 2c): a 3 fingered robotic hand with 5 DOF from iRobot[8], a 4 fingered robotic hand with 12 DOF and stereo camera system from Sandia National Labs[9], and a simple hook hand developed by BDI.

[7] http://www.carnegierobotics.com
[8] http://www.irobot.com
[9] http://www.sandia.gov

(a) (b) (c)

Figure 2: Robotic Hands: (a) iRobot (Odhner et al., 2013). (b) Sandia (Sandia National Laboratories, 2014). (c) Hook.

2.2 Onboard System

Onboard software has to provide the basic skills of the robot system required in the DRC, characterized by locomotion and manipulation in challenging environments, and employing a constrained bandwidth connection to human operators. Onboard software is thus developed with a focus on two requirements. The first requirement is operator independent onboard processing for autonomous and semi-autonomous functionality and the second is providing sensor data and interfaces to remote operators that allow them to gain situational awareness and interact with the robot over the constrained bandwidth link.

For the DRC, there is no strict limitation on computing power; during the DRC Trials, the onboard software ran on field computer machines located in the team garage. Our setup uses three desktop computers (Intel Core i7 3770, 3.4 GHz, 16GB RAM). Software modules are distributed among these machines according to functionality, with one machine each dedicated to motion control, perception, and planning. As the robot performs low-level closed-loop joint control using its internal main controller computer, a hard real-time computing system is not required for onboard software; a standard 64-bit Ubuntu 12.04 kernel is used for the Team ViGIR software.

Leveraging proven best practices within the ROS ecosystem, the robot is modeled using the URDF format and standard tools like the *tf*[10] library are used to provide a basic robot setup and model that could directly be used by many higher level components, like the motion planning system. The robot model is constructed at runtime using the *xacro*[11] macro toolset, to allow for flexibility (e.g. for choosing from available hand options on a per arm basis).

2.3 Operator Control System

The OCS consists of three computers (Core i7 3770 3.4GHz, 16GB RAM) arranged as two auxiliary operator stations, which used two 27" 1080p monitors side-by-side powered by a single-GPU NVIDIA GTX660, and the main operator interface station, which used four 27" 1080p monitors powered by a dual-GPU NVIDIA GTX690. The main station is placed in between the two auxiliary stations. The auxiliary station on the left is used as the main point of connection between the onboard and OCS computers, and allows the operators to monitor communication status at all times. The main station is used by our primary operator to plan tasks based on the 2D and 3D visualizations and issue commands to the robot. Finally, the right auxiliary station is used by our secondary operator to request and validate existing perception data used by the primary operator. The setup used in competition can be seen in Figure 3.

[10] http://wiki.ros.org/tf
[11] http://wiki.ros.org/xacro

Figure 3: Multiple operator OCS setup used during the 2013 DRC Trials.

Since we use ROS, we are able to take advantage of the several existing tools that it provides, mainly *rviz*[12] and *rqt*[13]. In the development of our main widgets (described in section 3.3), we use a combination of *librviz*, *Qt*, and *Ogre*. Leveraging the existing tools in librviz for visualizing 3D data communicated via ROS was very important given the short amount of time to implement the system before the DRC Trials. All 3D views in the OCS use existing and customized versions of rviz plugins (e.g., adding support to our own methods for picking geometry), in addition to completely new plugins that implement some of our OCS' unique features (e.g., templates). For the development of simple 2D widgets, we use rqt extensively; this allowed us to quickly prototype widgets during development that act as windows for specific controllers on the onboard side (e.g., footstep controller parameters).

2.4 Bandwidth-Constrained Communication

During the DRC Virtual Challenge and DRC Trials, a network traffic shaper was used to modulate the available communications between the robot onboard software and the OCS. This was intended to mimic the communication restrictions likely during a disaster response scenario. During the VRC, communication connectivity was good, but there was a strict bandwidth and total data budget that varied for different tasks. During the DRC Trials, the communications alternated every minute between good comms (1,000,000 bits/s, 50 ms latency) and bad comms (100,000 bits/sec, 500 ms latency). During bad communications, the system buffered excessive packets introducing significant delay in data transmission if the system attempted to transmit too much data. As only limited bandwidth between the OCS and onboard software is available, the capability to leverage this link to transfer useful data for the human operator to gain timely situational awareness is crucial. Additionally, during the initial VRC design there were discussions of dropping packets, therefore Team ViGIR designed the communications system to be tolerant of dropped packets and broken communication links.

As stated above our team chose to use ROS for our communications middleware. The ROS system uses a publisher/subscriber model with a centralized *roscore* to coordinate communications between ROS nodes. This is not suitable for use with the communications challenges defined above as the system cannot tolerate loss of communications to the centralized *roscore*. For this reason, the team chose to use two separate ROS networks for the onboard and OCS software and develop a custom communications bridge (CommsBridge); this setup was used during the initial VRC and throughout the DRC Trials.

The CommsBridge uses both TCP and UDP to communicate between the OCS and onboard software. For

[12]http://wiki.ros.org/rviz
[13]http://wiki.ros.org/rqt

an OCS command going to the onboard software, the OCS-based CommsBridge module subscribes to the relevant ROS topic, sends the data across the network, and republishs the data as a ROS topic on the onboard side; for the reverse, the onboard software subscribes to perception, state, and status messages, sends the data across the network, and republishs for use by the OCS. The CommsBridge is robust to communication dropouts, and automatically reconnects as needed if a process is restarted or communications are dropped. As the same topic names are used on both sides, the setup allows seemless testing as a single ROS network if desired.

For high-rate robot state information from the onboard software to the OCS, the CommsBridge uses UDP packets and a custom message packing that uses scaled integer representations in lieu of most floating point values, and custom data compression based on Google Protocol Buffers[14]. The data for robot position in the world, all joint positions (including fingers), and other state data is sent as a single UDP datagram. As only the latest robot state data is relevant for operator situational awareness, the CommsBridge system is tolerant to dropped packets. All control calculations, including planning and trajectory interpolation, are performed on the onboard side; thus, the operator does not require continuous velocity updates. The operator's situational awareness is based on intermittent position updates.

For operator commands, data sent only on transition, or large data that is not tolerant to dropped packets the CommsBridge uses TCP-based communications. The team developed a templated message handler that uses the message type, topic name, and direction (e.g. OCS_TO_FIELD or FIELD_TO_OCS) and automatically defines the appropriate communications for any valid ROS message. The approach uses the open source blob_tools[15] developed by TU Darmstadt to serialize the messages into "data blobs" and compress using the standard bzip2 compression library. On receipt of the TCP message, the data is uncompressed, deserialized into the appropriate data structure, and republished on the appropriate message topic.

Image data is handled by a special CommsBridge node denoted ImageBridge. On the onboard side, raw camera images are communicated using the ROS image transport package[16], and handled by special crop-decimate nodes that permited the operator to request variable resolution images. The operator can also request video data at specified frame rates using Theora[17] compression. Generally, the full image or video frame is displayed in a greatly reduced resolution while a defined region of interest (ROI) can be requested at higher resolution as shown in Fig. 4. A separate ImageBridge is created for both the full and cropped images for each camera feed. The ImageBridge forwards OCS image requests that specify the desired ROI, resolution, and frame rate to the appropriate crop decimate node, and sends the compressed image structures using the blob_tools package. ROS CameraInfo messages used by the image transport protocol are only communicated across the ImageBridge on change, but are automatically republished on the OCS side to permit proper visualization of the image in 3D representations.

Figure 4: OCS Image view showing variable resolution views for grasping drill during wall task practice.

[14]https://code.google.com/p/protobuf/
[15]https://github.com/tu-darmstadt-ros-pkg/blob_tools
[16]http://wiki.ros.org/image_transport
[17]http://wiki.ros.org/libtheora

3 Human-Robot Interaction

A key goal in our system design was to combine robot capabilities with the rapid and finely tuned perceptual and decision-making abilities of human operators in order to complete the challenge tasks quickly and efficiently. In order to achieve this goal, based on the principles of human-robot interaction (HRI) (Goodrich and Schultz, 2007), we designed a powerful User Interface (UI) that allowed the operators to communicate with and control the robot, maintain situational awareness, and make quick decisions. However, there are several challenges involved in designing a UI that provides this level of interaction between operator(s) and the robot.

3.1 Design Challenges

With an overwhelming amount of sensor data and information coming from the robot, our first major challenge was to decide what sort of information operators needed, as well as to determine when and how to display and interact with that information. Given ideal conditions for processing and communications, it would be easy to provide the user with all the processed and raw data coming from the robot at all times. However, it would also be difficult to understand and manage that information. Allowing the user the flexibility to select the data that is needed to perform each task (e.g., 2D map data to perform navigation or point clouds to perform fine hand manipulation), was one of our main goals.

Another challenge in designing the UI for the DRC was the limited bandwidth that was provided for communication between onboard computers and the operator interface. This meant that while there was a large amount of data being generated by the robot, only a small portion could be sent over to the operator station. This aspect of the DRC played an important role in the design of our UI, since we would not only have to provide flexibility in what data should be displayed at any given time, but also how much of that data would be transmitted (e.g., controlling resolution of images/point clouds).

In addition, because of the 3D nature of the tasks and data collected by the robot, operators would spend a large amount of time looking at and interacting with 3D visualizations. The location of the robot with respect to other objects has to be known when navigating in a real environment. The positions of the hands and fingers have to be accurately displayed when performing fine manipulation tasks. All tasks in the DRC required users to perform complex spatial judgments, and all of them required multi-DOF interactions with the robot and the environment. Viewing 3D data on desktop displays and interacting with 3D environments through 2D input devices is inherently difficult (Van Dam, 1997), so designing both the visualizations and interactions was a major challenge.

Finally, while the DRC tasks served as a reference and benchmark for our solution, we strived to provide maximum generality in our UI solution. Custom design of our UI for the specific tasks that we had to accomplish during the DRC Trials was a possibility (since we had the complete task specifications ahead of time), but we knew that this approach would not work in the DRC Finals, where tasks were less well-defined and in real-world scenarios, which might be completely unknown.

3.2 Design Goals

To overcome these challenges, we designed a set of graphical UI components (widgets) that allowed the operator to control the robot and visualize information about its current state and the world surrounding it. The main goal was to build a customizable UI with various widgets that could be changed and reorganized based on the task at hand, providing variable amounts of data and control, allowing operators to perform under any situation specified in the challenge. In order to achieve this, we first created rough and then detailed information design and interaction design storyboards for each of the tasks. This included scenarios and sketches (Rosson and Carroll, 2002) for the UIs with the features we envisioned, which helped us to determine requirements for completing each task with a minimal amount of bandwidth. Prototyping and

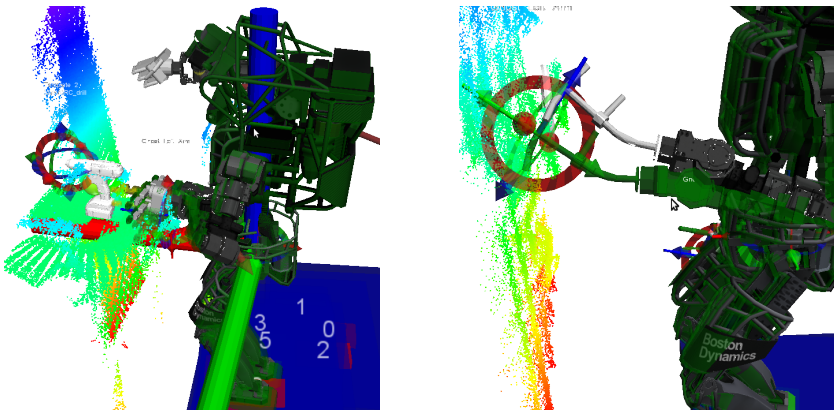
weekly meetings to evaluate the usability and functionality of the UI helped us to improve on the design.

In addition, while our initial design ideas would allow a single person to control every aspect of the robot and the tasks, we ultimately decided to design for multiple operators during the DRC Trials due to the complexity of the tasks and the limited amount of time to complete them. As shown in (Murphy and Burke, 2005), the use of multiple operators can increase overall robot system performance in search and rescue environments.

3.3 OCS Overview

A detailed description of our Operator Control Station (OCS) design is not possible here; we focus on two major features and the primary views/controls we developed. The first major feature is the use of 3D templates. Templates (3D models of important objects/features in the environment) allow the primary and secondary operators to annotate perception data with semantic information. For example, if the operator sees a known object in the point cloud (e.g., a tool), he can insert a template representing that tool in the 3D view at that location, thus informing other operators and the onboard systems about that object (Figure 5a). Based on the location of these known objects, the operator can then plan manipulation of the object using grasps and positions that maximize grip strength or reachability.

A second major feature, the "ghost" or simulation robot, is a transparent duplicate of the ATLAS robot visualization. The ghost robot allows the primary operator to plan and validate motions before executing them with the physical robot (Figure 5b). It is also color coded to give the operators feedback about the internal state of the onboard systems, such as collision checking for motion planning, to prevent unexpected actions. Both of these features can be used in and visualized at any of the views described below.



(a) (b)

Figure 5: Two major OCS features: (a) Templates. (b) "Ghost" robot.

The OCS used by the main operators contains three main components: *main view*, *map view*, and *camera view*. The *main view* is a widget that is primarily used for visualization of 3D data and fine manipulation control. It allows the operator to control end effectors, visualize the 2D and 3D reconstructions of the environment, annotate these visualizations with templates, and plan robot motion by controlling the ghost robot. The single 3D view can also be split into four 3D visualizations with different points of view and settings (orthographic/perspective), to facilitate spatial judgments and aid depth perception. The *map view* is a top-down orthographic view widget that is used for navigation and to request more information about the environment. The operator can select a region of interest in the environment by clicking and dragging to create a box selection, and then choose what type of data is needed (e.g., grid map, LIDAR/stereo point clouds). Fine control over the amount of data being requested helps in reducing the amount of information transmitted over the network and shown on the screen. Finally, the *camera view* allows the operator to request single images or video feeds with varying resolution from every camera on the robot, with up to four

images displayed at a time. Three-dimensional data can be overlaid on the images to validate the sensor data and prevent errors due to drift in position/orientation estimation.

The OCS also contains specialized components for controlling specific commands or to give flexibility to overcome unknown situations. The special widgets included the grasp control interface for each hand, which gave the operator access to pre-defined hand poses for templates placed in the environment, as well as widgets to provide individual joint control and planner configuration tools. The widgets and layout used in a manipulation task can be seen in Figure 6.

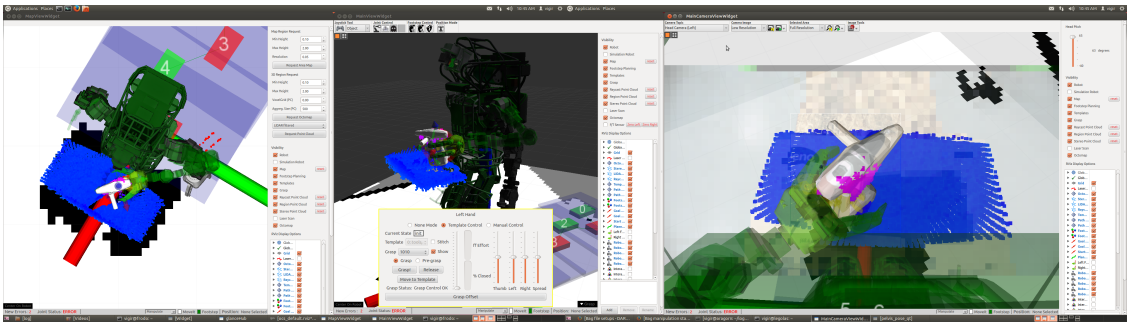


Figure 6: Overview of OCS layout used during a manipulation test, with the map view on the left, the main view in the center with grasp control popup, and the camera view on the right. All major controls are readily accessible through hotkeys or toolbar buttons that hide or display individual widgets; additional improvements are currently being incorporated to move the realization toward the initial concept and build upon lessons learned.

Finally, as mentioned in Section 3.2, one of our main goals was allow two or more operators to interact with the robot system using separate stations. In order to achieve this, we use a model-view-controller software pattern (Krasner et al., 1988): the model is instantiated as a set of ROS nodes that hold all the information about templates, the robot, images and user data; the views display the data to the operators; and the controllers allow the users to interact with and change the model. Thus, it is possible for multiple OCS instances to be active simultaneously. Since the two main operator tasks are requesting sensor data and commanding mission actions, we decided to separate these tasks per operator. While a primary operator evaluates perception data and controls physical actions of the robot, the secondary operator can then be responsible for other tasks, such as managing and requesting sensor data to provide situational awareness to the primary operator. While our team only had two active operators during the DRC Trials, our OCS implementation supports the addition of more operators that can supervise the actions of the main operators or even perform parts of the tasks themselves.

When designing OCS components, we followed a few guiding principles based on both best practices in HRI and our own usability evaluation of the UI used in the VRC. Our design process included weekly meetings and constant evaluation of the OCS. We wanted to minimize mouse movement by providing the ability to perform actions directly in the 3D environment, as well as providing keyboard shortcuts for the most used actions, such as requesting images or applying motions to the robot. All views can overlay multiple sensor data in same 3D environment to provide situational awareness, and the visibility of each piece of data could be toggled on/off to reduce visual clutter. Finally, we wanted to reduce the need for the operator to process detailed streams of text information. While detailed logs and complete information about errors and the state of the system is available at all times, we decided to create a widget that informs the operator about the state of different controllers at a glance, with a simple color scheme (green - successful, yellow - processing data, red - failed). This allows to quickly check the state of onboard systems.

During the limited development time prior to the DRC Trials, a number of features were being developed in parallel with the OCS development. The use of *rqt* and *rqt_python* allowed rapid development of a number of widgets required for proper configuration of specific components in our system (e.g., configuration of our

footstep planner parameters, and the glance hub mentioned above); unfortunately, a significant number of these widgets were not properly integrated into our design (e.g., selection of pre-canned poses, grasp control, and ghost robot configuration and control) due to limited resources. Thus, the version of our OCS used during the DRC Trials suffered from a major issue: clutter caused by the use of too many windows and buttons (as seen in Figure 7). While some of these windows were necessary to provide access to required functionality, the resulting clutter caused our OCS to have a very steep learning curve, and caused a big gap between our original intent with its design and the actual implementation. Moreover, while no formal usability studies were conducted to verify how big of an impact this had in operator performance, it is clear that many of these extra windows fail to comply to best practices in HCI and HRI (e.g., too many buttons that look exactly the same on the pre-canned poses widget). Because of this, properly integrating all existing features into our OCS design is one of our major goals going forward toward the DRC Finals.

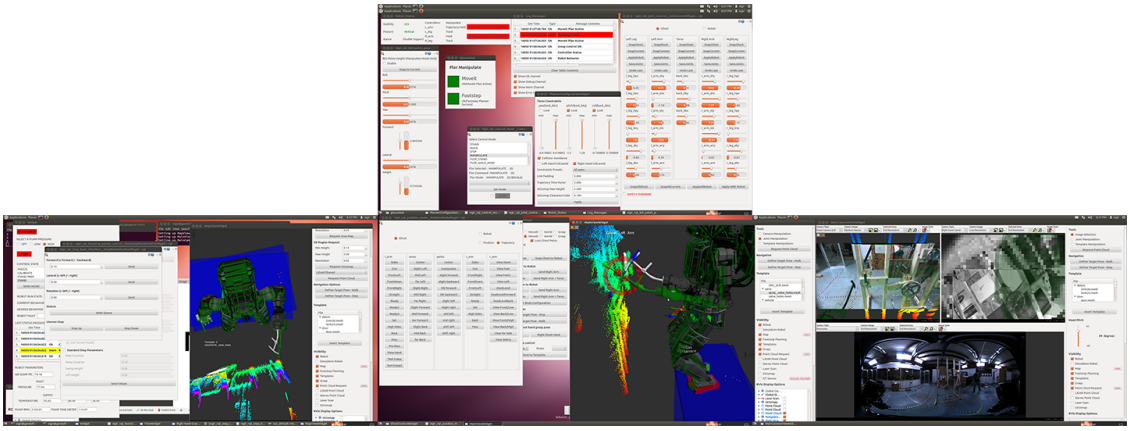


Figure 7: Overview of widgets used in the 2013 DRC Trials, with the map view on the left screen, the main view in the center, the camera view in the lower right, and all other OCS widgets spread throughout the available screen space. Compare this cluttered version with our current version shown in 6.

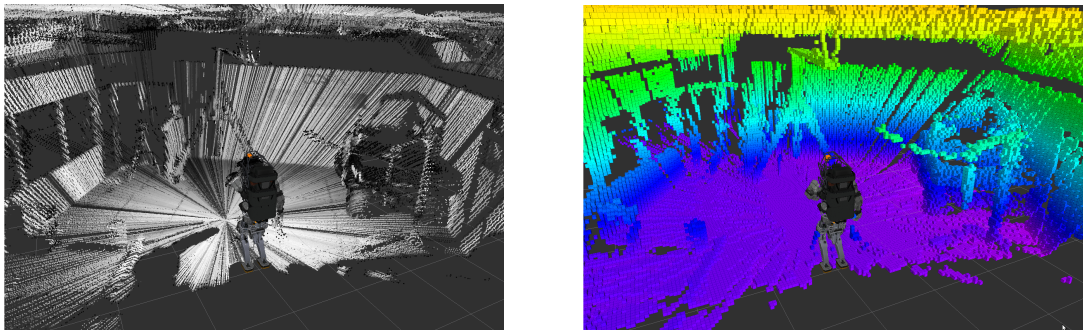
4 Perception

The perception system has to provide perceptual information to two different sinks: the onboard software components and the OCS. For onboard use, bandwidth constraints are not relevant and therefore information from all sensors can be integrated at full bandwidth. Data transfer to the OCS on the other hand is severely constrained, therefore the perception system must support selective data query and transfers.

4.1 Worldmodel Server

The Worldmodel Server component preprocesses, collects and aggregates sensor data and makes it available to both onboard and OCS system components. Leveraging established open source tools like Point Cloud Library (PCL) (Rusu and Cousins, 2011) and Octomap (Hornung et al., 2013), the worldmodel server allows queries of information about the environment with flexible level of detail and bandwidth consumption.

As described in Section 2.1, three dimensional sensing is provided by the sensor head, providing point cloud data both via the Hokuyo UTM-30LX LIDAR and an integrated stereo camera system. As the stereo camera system has a smaller field of view and is sensitive to lighting conditions, LIDAR data is used as the main source for creating a 3D geometry model of the environment onboard the robot. To achieve this, the planar scans of the LIDAR have to be preprocessed and aggregated, resulting in full 3D point clouds. The following preprocessing steps are employed: First, scan data is filtered for spurious measurements commonly called shadow points or mixed pixels that occur at depth discontinuities (Tuley et al., 2005) using the shadow point



(a) (b)

Figure 8: Sensor data processing: (a) LIDAR data with intensity information. (b) Resulting octomap representation.

filter available for ROS [18]. The filtered scan is then converted to a point cloud representation. During this process, the rotational motion of the LIDAR on the slip ring is considered and high fidelity projection is employed, transforming every scan endpoint separately. In a last step, parts belonging to the robot have to be filtered out of LIDAR data. To increase robustness against errors in kinematics calibration, a specialized robot geometry model uses simplified and enlarged collision geometries for self filtering purposes. LIDAR scans are saved to a ring buffer along with snapshots of coordinate frames used within the system. By employing this method, aggregate point clouds relative to different coordinate frames can be provided on request. A ROS API allows querying the world model via both ROS topics or services and flexibly retrieving region of interest point cloud or octomap data relative to various coordinate frames.

The primary onboard 3D geometry model is created using Octomap, a volumetric, probabilistic approach using an octree as a backend. Using this approach, the environment representation maintained onboard can be updated efficiently and in a probabilistically sound way. Even in case of changes in the environment or drift in state estimation, the environment model is updated accordingly and maintains a useful representation.

The Octomap environment model provides the main geometry representation and is used for multiple purposes. Using ray casting, distances to geometry can easily be determined. This feature can be used from within the OCS to perform ray cast distance queries against onboard geometry. In this case, only the ray cast information has to be transmitted to the robot and the distance information is transmitted back, utilizing very low bandwidth. The capability to request regions of interest (ROIs) of the environment model allows to transfer small ROI geometry over the constrained connection on operator demand and also makes geometry available to other modules, like the planning system. Similarly, it is possible to request 2D grid map slices of the Octomap representation, aggregating 3D data into a 2D grid map. Using compression, this representation is very compact and often sufficient for operators to gain situational awareness or for motion planning.

4.2 Object Detection and Pose Estimation

To perform manipulation tasks, object identification and pose estimation is required. For the DRC, we use a library of geometry templates for different objects as described in Section 3.3. Using the UI, the operator can insert a template into the 3D representation of the environment and adjust the 6DOF transform of the template. Using both camera imagery and 3D point cloud data from LIDAR and stereo camera, the template can be aligned to this sensor data. This 6DOF pose data is transmitted to the onboard systems and used by the planning and grasping control systems (see Section 7).

[18] `http://wiki.ros.org/laser_filters#ScanShadowsFilter`

An automatic object detection and pose estimation system using the Object Recognition Kitchen (ORK)[19] framework as a backend and adapting the LINEMOD (Hinterstoisser et al., 2011) approach for DRC tasks has been developed within the team. This was not used during the DRC Trials due to time constraints and challenges in adjusting the system to work under outside light conditions. We intend to further explore automatic pose estimation during the next phase.

4.3 Kinematics Calibration

Joint angle sensors of the Atlas robot exhibit a joint angle bias offset as well as a scale factor offset with respect to the true joint state. Without calibration of these effects, the joint kinematics configuration as measured by the joint sensors and the true configuration differ, making accurate manipulation difficult. For kinematics calibration a proven bundle adjustment approach previously used on the PR2 robot (Pradeep et al., 2014) available via the *calibration*[20] ROS package is adapted for use with the Atlas robot. To perform calibration, a calibration checkerboard is attached to the arm end effectors. A dataset comprising multiple camera views of the checkerboard using different arm joint configurations is then recorded. Due to the bundle adjustment approach employed by the calibration system, the transform between checkerboard and end effector does not have to manually specified, but can be estimated in a first calibration step, keeping the other arm kinematics parameters to be estimated afterwards fixed. A second calibration step is then employed to jointly optimize for the checkerboard transform as well as the kinematics parameters. Fig. 9 shows calibration results for the right arm. The reprojection error is significantly reduced after calibration. The cause for the remaining error will be investigated in future work.

5 Motion Control

Although the use of hydraulics in heavy machinery is quite common, there are relatively few examples of hydraulically actuated humanoid robot systems in comparison to the large number of electrically actuated robots used both for research and in commercial applications. While advantageous in terms of bandwidth and force magnitudes that can be generated, hydraulic actuation poses significant challenges for control of the Atlas robot system. Given limited development time, Team ViGIR chose to focus on position/velocity based joint control techniques using the BDI interface.

5.1 Friction identification

Friction effects in joints have significant impact on control performance, for example when applying torques for gravity compensation. In hydraulic actuators, friction is known to have significant impact (Lischinsky et al., 1999) and multiple approaches of different complexity can be used to compensate for it.

Fig. 10a-e shows the Stribeck curves for five of the six left arm joints. To measure friction, each joint was moved individually while leaving the remaining joints in a fixed position to avoid any coriolis and centrifugal effects. Using a PI-controller, each joint was moved with various, but constant velocities. In periods of constant velocity, the joint torque was measured via the built-in pressure differential sensors in the joints and averaged to cancel out sensor noise. Using model-based gravity compensation, gravity effects were partly compensated. All curves differ in shape, clearly showing that friction identification has to be performed for each joint separately. The error bars in the plots show the standard deviation for the measured friction, indicating high variance in some measurements.

In a second experiment, the valve current was commanded to constant values. The result of this experiment is shown for the left elbow joint in Fig. 10f. Although constant current values are applied, measured torques

[19] http://wg-perception.github.io/object_recognition_core/
[20] http://wiki.ros.org/calibration

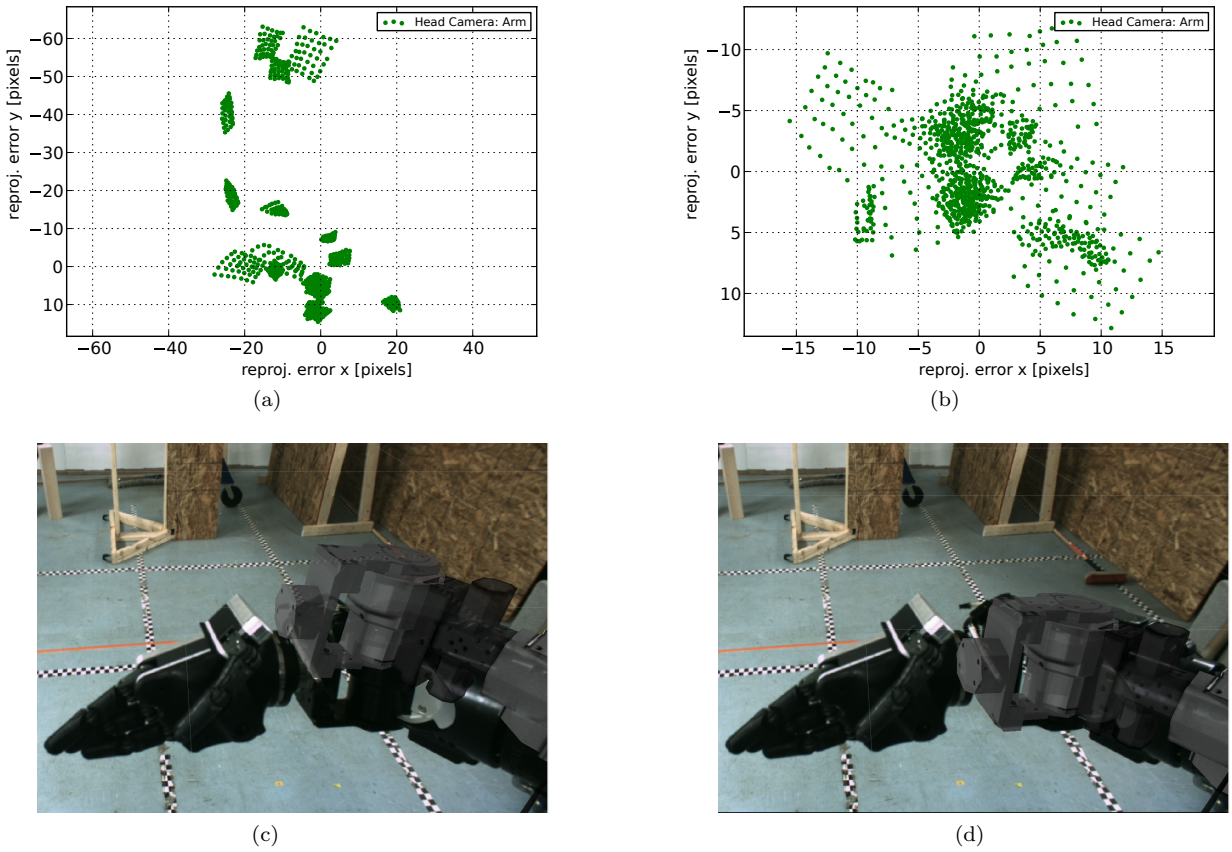


Figure 9: Kinematics calibration. Scatter plot of checkerboard corner reprojection error between forward kinematics based calculation of checkerboard corner positions and detections in camera images: (a) Before kinematics calibration (b) After calibration of joint angle offset biases and scale factors. (c) Example of reprojection of forward kinematics based arm configuration before calibration (d) after calibration

show a slope, illustrating the need to model actuator dynamics. The asymmetric response of joint actuation is also readily visible, with the same absolute value of applied current resulting in different measured torques, depending on sign of the applied current.

From the results of this section, we concluded a simple friction model based on the current joint velocity is not sufficient for effective friction compensation as the dynamic hydraulics effects must be taken into account. Friction forces have a tremendous impact on a pure torque based control policy; this led us to abandon attempts at model-based control for the DRC Trials as described in Section 5.4.

5.2 Control Modes

We chose early on to focus development on operator interactions and manipulation, while leveraging the basic capabilities provided by the Boston Dynamics API. These include behavioral control modes for maintaining balance in STAND and MANIPULATE modes, as well as basic WALK and STEP modes. Team ViGIR interfaces with these robot control modes through a custom RobotController program using an operator selectable set of control modes that determines what commands are accepted and processed by the robot control interface. While BDI supplied a USER mode that permitted complete control of all 28 joints in the robot, we chose not devote significant resources to developing whole-body controls prior to the DRC Trials.

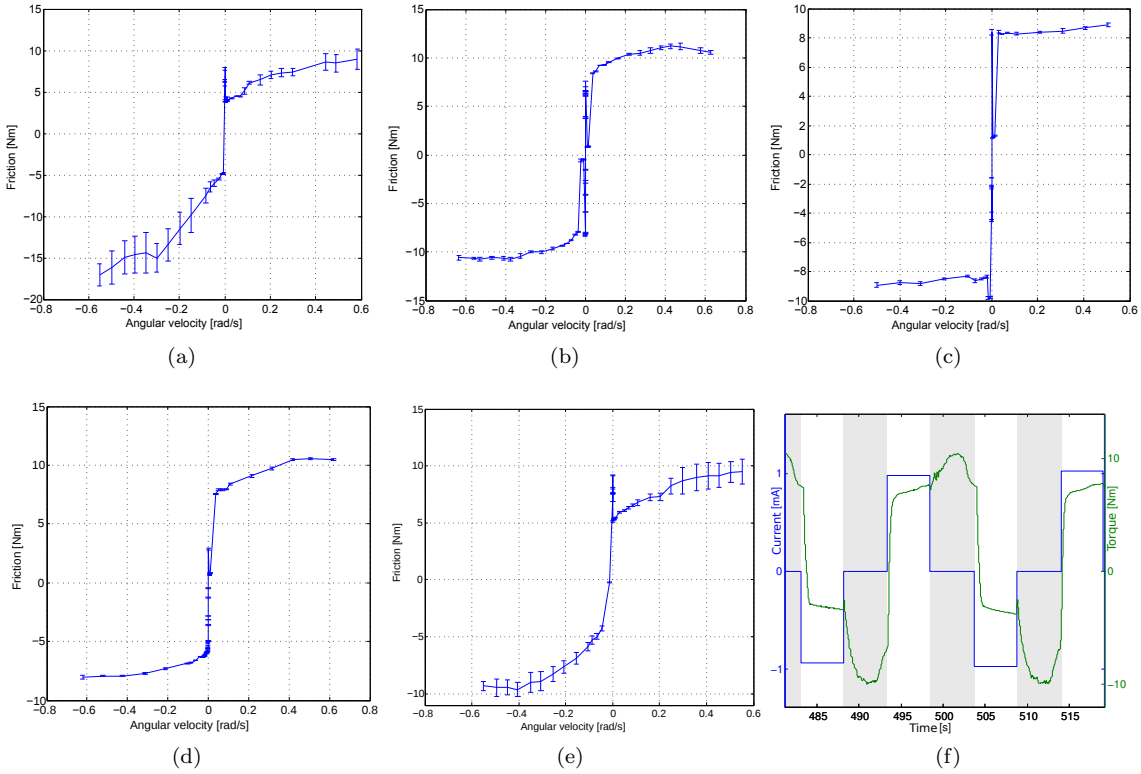


Figure 10: Friction analysis: (a)-(e): Experimentally determined Stribeck curves for left arm joints. (f) Valve current and torque for the left elbow joint. Only the parts of the plot with white background show torque sensor measurements with applied constant current. Dark background parts indicate motion of the joint using position control.

Thus, the USER mode was only used during the ladder task for the 2013 DRC Trials competition; although it was used more extensively during the VRC.

The WALK and STEP modes allow the onboard control system to send a sequence of footsteps defined in a world frame shared by the robot pose estimate. The RobotController accepts a footstep plan and sends the next four steps to be executed to the robot, which executes the steps using the robot's BDI-developed software to maintain balance. These control modes also provided the operator limited ability to command the upper body joints in the torso, arms, and head using joint commands.

The robot accepts joint position[21] or trajectory[22] vectors for joints grouped according to the appendage. The left/right arm/leg commands specify six joints each for 24 joints total, the torso specified three joints, and a separate head command accounted for the full 28 degrees of freedom for the robot. During the DRC Trials, we mainly depended on BDI behaviors for lower body control and balance; the leg appendage chain control was only used during the VRC and Trials ladder task. Coordinated motion of multiple appendages is obtained through the use of trajectories with a common time reference.

Commands to the robot hands are governed by the grasp controller, and are completely independent of the main robot control interface.

[21] http://docs.ros.org/api/sensor_msgs/html/msg/JointState.html
[22] http://docs.ros.org/api/trajectory_msgs/html/msg/JointTrajectory.html

5.3 Joint Control

The robot API permits control of individual joints of the robot upper body in all behavioral modes, and lower body joints in USER mode. The joint control API permits control of position, velocity, and force based on a proportional-integral-derivative (PID)-based control law that included some feedfoward terms[23]. We predominantly used position-based PD control with a force term used for gravity compensation and a feed-forward term based on commanded velocity.

The software system has two basic modes with position and trajectory joint angle commands. As the hydraulic actuators are capable of significant forces and fast motions in response to step changes in commanded position, the operator interface position commands are converted to trajectory commands from current position to desired positions over a variable time interval. Upon receipt of a new trajectory command by the controller interface, the commanded trajectory is stitched to existing trajectory commands, and smoothed. The smoothing process recalculates the internal joint velocities at each desired joint position in the defined trajectory vector so that the trajectory can be represented as a sequence of cubic spline segments that match position and new velocity at each trajectory point.

During execution of the trajectory, the controller evaluates the cubic spline for the relevant section based on current time and publishes the desired position and velocity for each joint. As the main controller of the robot performs closed loop control with hard real-time guarantees, a dedicated standard Kernel Ubuntu machine is used for the RobotController, sending joint command updates at 250-300 Hz to the robot main controller. Once the trajectory end point in time is reached, the controller holds the current joint position with zero desired velocity. If robot joints are under control of an internal robot control mode, such as leg joints during stepping, the robot control interface tracks the current actual joint position so that mode transitions are bumpless[24].

In general, the per joint control response was excellent. Difficulties in manipulation were due to kinematics calibration errors and kinematic limitations of the arm design. The former difficulties were partially overcome by improved calibration as described previously (cf. section 4.3); The latter difficulties require additional improvements to the physical arm and the inverse reachability and inverse kinematics system in the next phase (cf. section 6.2).

Another difficulty that was not addressed in this phase was the dynamic coupling between arm and torso motions due to the massive arms. The Rigid Body Dynamics Library[25] (Felis, 2012) is used to calculate the forces required to support the robot in a quasi-static manner for gravity compensation, but full inverse dynamics was not attempted at this stage. Preliminary testing, as previously discussed (see Section 5.1), showed that the model based force calculations were significantly undervalued due to model inaccuracies and the high friction inherent in the hydraulic actuators. In lieu of model-based inverse dynamics, the controls focused on sending smooth arm trajectories with relatively slow motions to minimize the impact of dynamic coupling, and used relatively stiff PD control on the torso joints to minimize perturbations.

5.4 Whole Body Control

In the BDI USER control mode all joints can be controlled by the operator. Using this cabability for whole body control (WBC) was identified as a promising approach for improving manipulation performance preceding the VRC competition. Using established methods (Sentis, 2007) as a foundation, a whole body force control framework based on the RBDL dynamics library (as already used for gravity compensation,

[23]The exact form of the control equation and parameters is proprietary to Boston Dynamics, and is governed by non-disclosure agreements.

[24]'Bumpless' is a term of art in the industrial control community used to denote a continuous transition of control output between automatic and manual control modes (c.f. http://www.mathworks.com/help/simulink/examples/bumpless-control-transfer-between-manual-and-pid-control.html).

[25]http://rbdl.bitbucket.org/

cf. section 5.3) was developed. Using this approach, arbitrary operational points (any points on the robot links) can be controlled in task space. Using a hierarchy, tasks are projected into the joint space Jacobian null space of higher priority tasks, guaranteeing that higher priority tasks are not violated.

The pelvis of the robot was modeled as an unactuated 6 DOF free floating base. Balancing was obtained by controlling the position of the robot center-of-mass (COM) and by projecting every motion in the constrained space of the supporting contact limbs. Using this approach, the simulated robot was able to stand statically stable in dual stance and also perform stable single stance motions in simulation, as shown in Fig. 11a.

To support manipulation tasks and to achieve compliance while being in contact with obstacles, a hybrid position-force policy similar to the one described in (Khatib, 1987) was used. Fig. 11b shows an example of a hybrid control application, with the robot pushing a cinderblock across a table.

While showing very good performance in simulation, the developed WBC system relies on a accurate dynamics model of the robot which was available for simulation in the drcsim simulator [26], but not for the real Atlas system. This can degrade performance of the used operational space approach (Nakanishi et al., 2008). As time constraints made detailed system identification infeasible, this WBC approach was not used in the 2013 DRC Trials.

6 Motion Planning

Due to high latency and low available bandwidth, approaches that require constant operator input and supervision for motion generation are both slow and potentially unsafe. For this reason, automated planning is employed onboard the robot system, allowing the planner to leverage the availability of all sensor data with no latency. Ideally, the operator just has to specify a goal pose, with the planning system generating a viable and collision free plan to that pose. Separate planning systems have been developed for locomotion as well as manipulation.

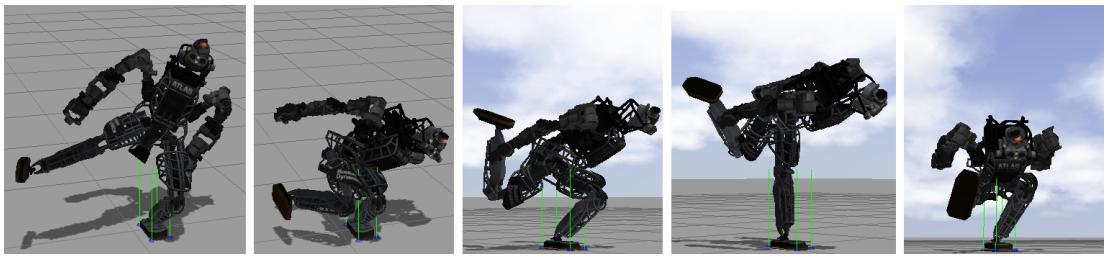
6.1 Footstep Planning

As described in Section 5.2 the Boston Dynamics API provides the capability for the robot to follow footsteps. Given a sequence of footsteps it generates and executes smooth trajectories for each foot placement which allows to decouple planning and execution level, also known as contacts-before-motion planning. For this purpose our approach is based on the already existing *footstep_planner* ROS package[27] which expands a graph and applies an Anytime Repairing A* (ARA*) search algorithm to obtain a sequence of footsteps (Hornung et al., 2012). For 3D planning on rough terrain, a suitable 3D world model, as well as suitable states, actions, transitions, and cost functions must be defined.

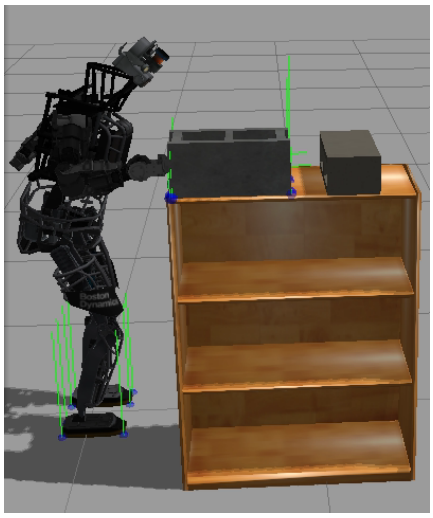
6.1.1 3D World Model

For 3D footstep planning a suitable world model is required which provides all data in an efficient way. The first step is pre-filtering the LIDAR point cloud to reduce noise in LIDAR data. Afterwards surface normal estimation[28] based on Principle Component Analysis (PCA) is applied (see Fig. 12a). The resulting normals are used to determine foot attitude for each step. A height map (see Fig. 12b) is generated which allows determining the foot height for each step and estimation of the ground contact percentage (see subsection 6.1.5 for further details).

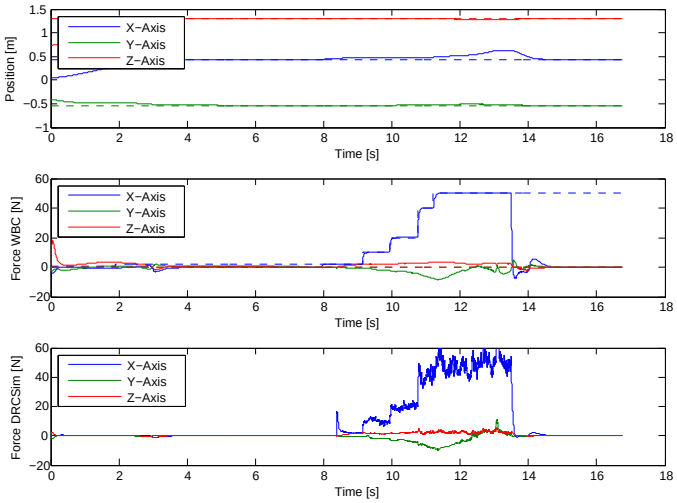
[26] http://gazebosim.org/wiki/DRC
[27] `http://wiki.ros.org/footstep_planner`
[28] For more details see `http://pointclouds.org/documentation/tutorials/normal_estimation.php`



(a)



(b)



(c)

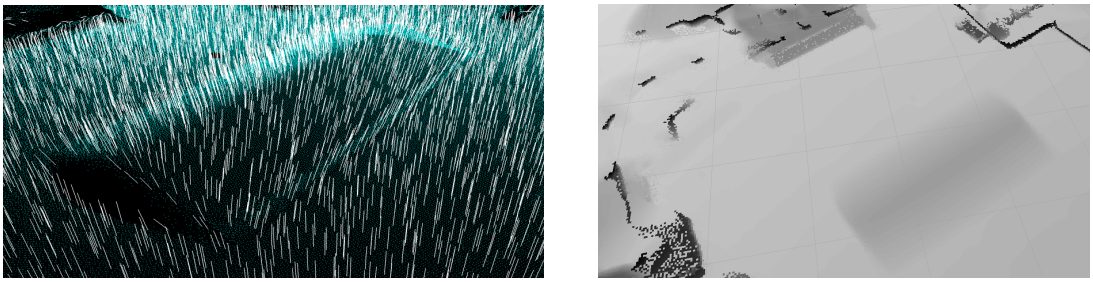
Figure 11: Whole Body Control: (a) Single leg stance motions (b) Atlas pushing cinderblock using hybrid position/force control. Along the X axis, a force control law is used, while the Y and Z axes are governed by position control (c) Position/force plot for the right hand. Solid lines are the sensed values; dashed lines are the commanded values. The upper plot shows the hand position, the middle plot the forces estimated by the WBC framework and the lower plot shows the forces provided by the simulated force torque sensors of drcsim. Starting at approximately 8 seconds, the robot begins pushing the block with stepwise increasing force, overcoming friction when applying over 50N. At the 13 second mark, the pushed block is stopped by the other block behind with no uncontrolled forces acting on the robot system.

6.1.2 States, Actions and Transitions

The foot state s is defined as global position and attitude in 3D space by $\mathbf{s} = (x, y, z, \phi, \psi, \theta)$. Here, ϕ, ψ and θ are the roll, pitch and yaw angle of the foot. The z, ϕ and ψ values are constrained by the underlying terrain, so all actions a can be given as a displacement vector $a = (\Delta x, \Delta y, \Delta\theta)$ relative to the stance foot s_0. All available actions are defined in the set of footstep primitives denoted as A. The original approach uses a manually defined set A, but in rough terrain scenarios the foot placement has to be flexible for the robot to be able to step over unstructured debris on the floor. Atlas is a human-sized robot having a large variety of possible footstep primitives, therefore it is not feasible to define such a set A manually. For this reason we define a reachability region $\mathcal{R}$ by a polygon next to the stance foot s_0 from which the footstep primitive set A is discretely sampled. Finally we define all possible transitions (s_0, s) by a function $s = t(s_0, a), a \in A$.

6.1.3 Cost Functions

In the original approach (Hornung et al., 2012), half steps are used for evaluation of cost functions. We found this representation to be insufficient for modeling cost functions of human-sized robots. Ignoring the



(a) (b)

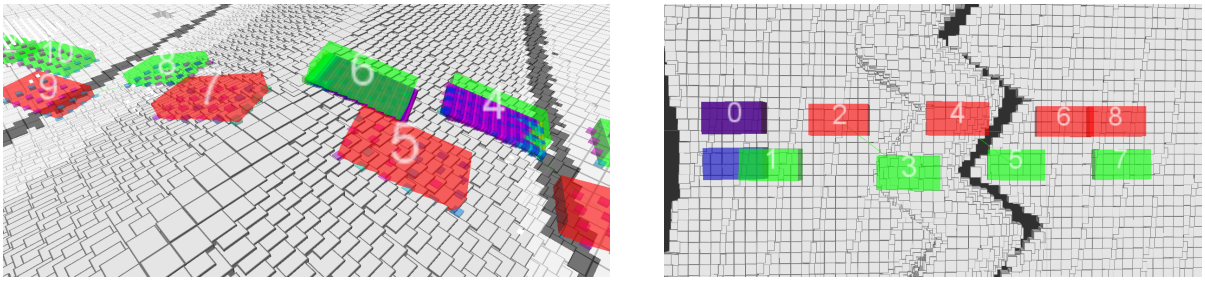
Figure 12: Most important parts of 3D world model: Normal estimation (a) and height map (b) of a ramp

source of the swing foot may lead to non-smooth footstep sequences, causing a fall. For this reason, in this work all cost are evaluated for full steps with s and s' denoting the source and target configuration for the same foot respectively.

The footstep planner has to minimize multiple, possibly competing costs e.g. shortest path and minimal fall risk. For this reason a hierarchical cost function was designed, modeling each criterion in a different layer. This approach was realized by using the Decorator Pattern (Gamma et al., 1994), enabling the flexible composition of cost functions. Let $c_i(s, s', c_{i+1}(s, s', ...))$ denote the cost function of the i-th layer given the result of the next layer's cost function. This recursion is stopped by the last layer which just evaluates $c_n(s, s')$. Each layer receives the result of the next layers and may reuse this value for internal purposes. In most cases cost functions are designed to just accumulate cost from the next layer, thus in common cases the resulting total cost may be illustrated as a weighted sum $c(s, s') = \sum_{i=1}^{n} w_i c_i(s, s')$. Using the 3D world model, the planner is able to generate sequences of footstep plans over rough terrain. Figure 13a shows a solution for the pitch ramp in which the soles of the feet are well aligned to the underlying 3D world model.

6.1.4 Risk Measurement

Using the reachability region $\mathcal{R}$ implies the risk of including bad footstep placements that lead to falls. For this reason a quantity denoted *risk* is required additionally to the cost. We are using cost and risk to distinguish between effort and feasibility of a step. Merging them in a single value would not prevent the planner from using bad steps in a plan when only a small number of step options are available. Analogous to evaluation of cost, each cost function computes the risk simultaneously and forwards it to the next layer. Feasibility of a step is finally determined based on the accumulated risk.

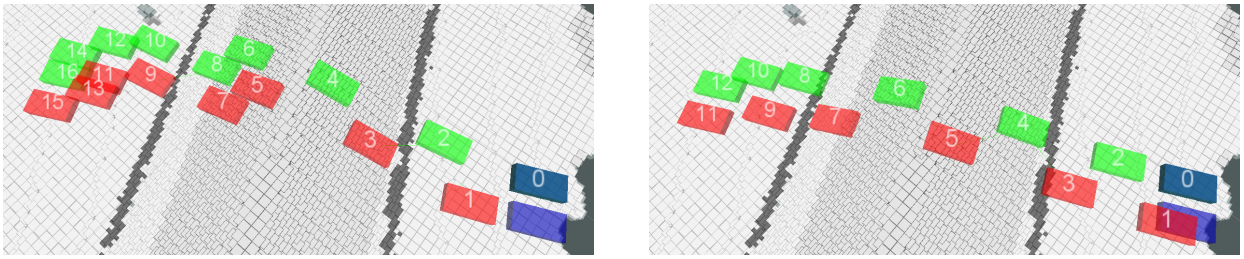


(a) (b)

Figure 13: Example solutions for 3D planning in rough terrain: (a) Solution for walking over the pitch ramp showing footsteps aligned with underlying geometry. (b) Plan for traversing over the chevron hurdle with some footsteps on the hurdle having less than full ground contact.

6.1.5 Ground Contact Estimation

The Atlas robot is not capable of stepping over a cinder block sized obstacle, so this task can be solved only by stepping on top of such obstacles. For ground contact estimation, edge detection based on grid maps generated from point cloud data was implemented first. While performing as expected, occupancy grid maps have the drawback of wasting space by encircling each obstacle with non-traversable cells. As a result, collision checks with occupancy grid maps are too strict because they enforce placing each foot avoiding obstacles completely. This results in longer paths and the ARA*-planner takes longer computational time trying to minimize this path length. The resulting plan in figure 14a is obviously an ineffective solution to travel across the pitch ramp. An alternative approach therefore estimates the percentage of ground contact for a given foot configuration which allows usage of foot configurations with less than 100% ground contact. The planner is thus capable of planning for overhanging steps while keeping them collision free without the usage of discretized occupancy grid maps, improving the result on the pitch ramp significantly (see figure 14b). In figure 13b another example is illustrated, where the steps 3 and 4 are placed on cinder blocks with portions of the feet overhanging the blocks, enabling to generate a straight footstep plan over them.



(a) (b)

Figure 14: Comparison of different collision check approaches: (a) Occupancy Grid Map (b) Ground Contact Estimation

6.2 Manipulation

For manipulation, motions need to be generated to move manipulators into desired configurations for grasping or other tasks. As it can reduce operator workload considerably, a desirable capability is automated collision avoidance, both with regards to self-collisions of the robot (e.g. arm coming in contact with torso) and collision with regards to collisions with environment. When performing manipulation in contact with the environment, planned motions must not lead to high internal forces acting on the robot. As described in section 5.4, force control was not a viable approach during the DRC Trials due to the limited time for proper system identification; for this reason, contact tasks were performed using position control with gravity compensation, making collision free and precise kinematic planning indispensable. As high latency limits the usefulness of otherwise promising approaches for teleoperation of end effectors (Leeper et al., 2013), we primarily employ planning to goal configurations. The Atlas arm kinematics proved to be challenging for manipulation, as the joint limits and composition of rotational DOFs in the arms result in limited reachable workspace. An important part of planning is selecting a pose of the robot pelvis relative to objects of interest in a way that allows the end effector to reach the object. For this, the Simox[29] open source inverse reachability approach described in (Vahrenkamp et al., 2013) has been integrated with our system.

We developed a custom manipulation planning package based on the MoveIt! (Chitta et al., 2012) framework available for ROS by using the low level MoveIt! API to provide additional specialized functionality required for the Atlas robot and DRC tasks. The system enables planning to goal joint configurations and to goal end effector poses. Two planning modes are available: The default mode is unconstrained planning, with joints free to move to reach the goal. Optionally, motion can be constrained to follow a Cartesian path between the start and goal end effector pose. In this case, waypoints are generated based on linear interpolation between

[29] http://simox.sourceforge.net/

start and goal position and orientations for waypoints are generated using spherical linear interpolation (Slerp) (Shoemake, 1985) between start and goal end effector quaternions. More complex constrained motions such as circular motion for turning a valve are generated by concatenating multiple short linearly interpolated Cartesian paths as shown in Fig.15.

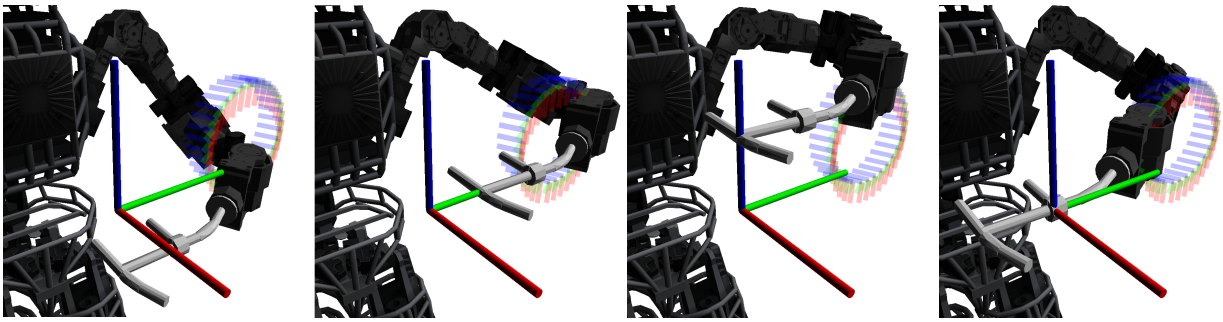


Figure 15: Clockwise circular path. The Hook rotates around the X axis(red) while the interpolated poses are shown for the last joint of the arm.

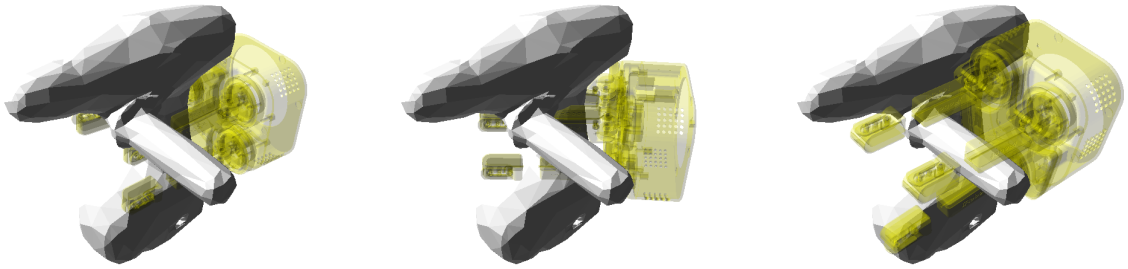
For obstacle avoidance, a region of interest of the worldmodel octomap as described in section 4.1 is used. As contact with the environment is required in many of the DRC tasks, collision checking between end effectors and the environment can optionally be switched off. For example, collision avoidance is needed to safely bring the robot hand into a position to pick up the drill. In order to grasp the drill, collisions between the palm and fingers of the hand and the drill handle must be allowed. With the challenging outdoor scenarios of the DRC, noise in sensor data that leads to geometric artifacts, preventing successful planning due to spurious collisions cannot be ruled out completely. To cope with such situations, collision checking with the environment model can also be disabled for the complete robot geometry; in this case the ghost robot changes color to warn the operator. Motion planning can be performed using either 6 DOF with the arms only, or by including the torso joints and using up to 9 DOF. As the 9DOF planning mode tends to result in higher control error or oscillation in some joint configurations,the operator can lock a selection of torso joints to restrict the planning space.

7 Grasping

For tasks requiring grasping operations, we use human-in-the-loop grasping and rely on the operator input to make crucial decisions. This strategy was chosen since for most objects, there are only one or two ways in which the object is to be grasped in order to complete a specific task. Thus, task specific grasps can be generated off-line and stored for future use, therefore removing the necessity for a robust online grasp generator. Each of the pre-generated grasps are associated and linked to a particular 3D geometry model (template) that the user can load in order to execute the stored grasps. After completion, the user can visually inspect the resulting grasp to ensure that the grasp executed correctly before proceeding with the rest of the task.

7.1 Template Grasps

All of the objects in the DRC Trials competition were well defined; therefore, we chose to create a 3D geometry template for all relevant objects, matching sizes and shapes. Each template contains information related to the object such as mass, center of gravity, and nominal orientation. Grasps are created off-line using the GraspIt! toolkit (Miller and Allen, 2004), taking into account parameters such as hand type, task, object size and relative location of the object in the environment. Once a reliable grasp is created, it is given an index number and the relevant data stored with respect to the template center of mass. The recorded data includes grasp and pre-grasp positions as well as a desired relative pelvis pose for the robot to maximize



(a) (b) (c)

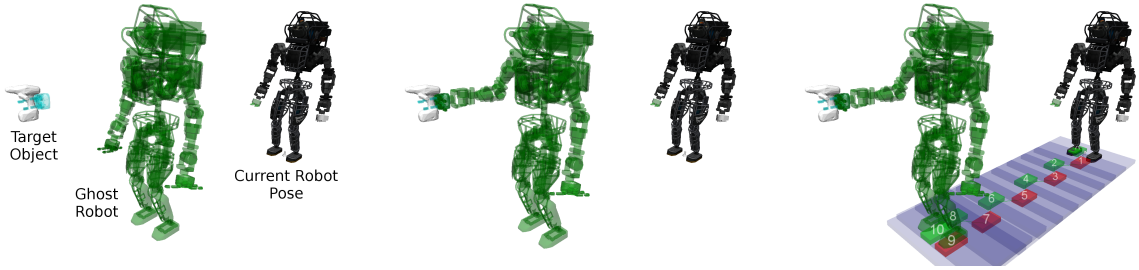
Figure 16: Precomputed Grasps visualized relative to drill template: (a) Front grasp. (b) 45 degrees grasp. (c) Handle grasp.

reachability. The pre-grasp location is typically created by moving the position of the hand approximately 10 cm away from the object in the direction normal to the palm. By creating a set of grasps offline, the cognitive burden on the operator is reduced because the number and types of choices are reduced.

During the task, the operator selects and loads the appropriate template into the main 3D view window. The operator aligns the template with perception data in the 3D view (see Section 4.2). Upon inserting the template, the associated grasp information is loaded and the operator is able to select and view all of the stored grasps. Stored grasps are visualized by showing a translucent hand overlaid on the template (Fig. **??**) to inform the user which grasp is currently selected.

7.2 Grasp controller

After placing the template and selecting the grasp, the grasp can then be tested for reachability using the ghost robot (Fig. 17a). If the grasp is reachable, the IK solution is displayed (Fig. 17b). If no solution is found, the ghost robot remains unchanged in its previous state. The user can then manipulate the robot pose, grasp orientation, template location, or torso constraints until the desired grasp is achievable. Once a feasible configuration is found, the user can send a footstep plan to move the real robot to the ghost robot pose. (Fig. 17c).



(a) (b) (c)

Figure 17: (a) Testing reachability, (b) Reachable position found and (c) Footstep plan to ghost pose.

Once the robot is correctly positioned with respect to the object, the user can then send the grasp to the grasp controller. The grasp controller then transitions through a series of states from the initial approach to the final grasp state. The grasp controller first enters the "approaching" state where it plans a collision free path from the current joint configuration to the IK solution for the pre-grasp position using the motion planner described in Section 6.2. Once the pre-grasp position is reached within a margin of error, the controller transitions into the "surrounding" state, moving the hand into the final grasp position near the

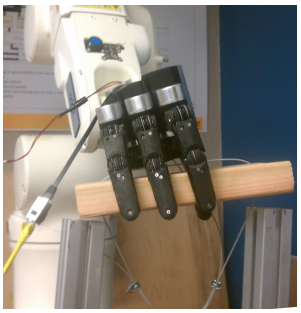
object. When the final grasp position is reached, the controller switches to the "closing" state, closing the fingers around the object so that all fingers make contact simultaneously. When the grasp is completed, the grasp controller switches to the "monitoring" state where it maintains the grasp and signals the user that the grasp is completed. If at any point during the process an error is detected, the controller will make the appropriate corrections and state transitions in order to complete the grasping process. Additionally, the user has the ability to abort the grasping process if there is a problem that cannot be handled automatically by the grasping controller. If this occurs, adjustments to the grasp must be manually corrected by the operator until the final desired grasp is achieved. This primarily was an issue with the wall task, since correct placement of the hand was crucial in order to operate the drill.

After a grasp is executed, the user can visually inspect the grasp to make sure that it matches the desired grasp. Since the final grasp may not match the initially planned grasp due to small movements of the object while grasping or slippage of the object after grasping, the user can modify the planned grasp to match the actual one by using a "stitch template" feature. Because variations between the planned and final grasp can affect manipulation quality, the object pose with respect to the hand must be accurately estimated. By rotating the template relative to the hand and positioning it to match the actual object orientation, the modified template transform can be used to update the actual grasp location. All future manipulations of the template will use this new transform, thus "stitching" the object to the hand and aiding the user in manipulation. This very important for the wall cutting task, as the cutting bit pose is used for planning cutting operations.

7.3 Hardware validation

Prior to selection of either the iRobot or the Sandia hands for use during competition, the robustness of the grasps produced by each hand was evaluated. Each hand was placed on the end of a six degree of freedom robot with a sample object anchored to the table below using a steel cable and force sensor (see Fig. 18a and Fig. 18b).

(a)



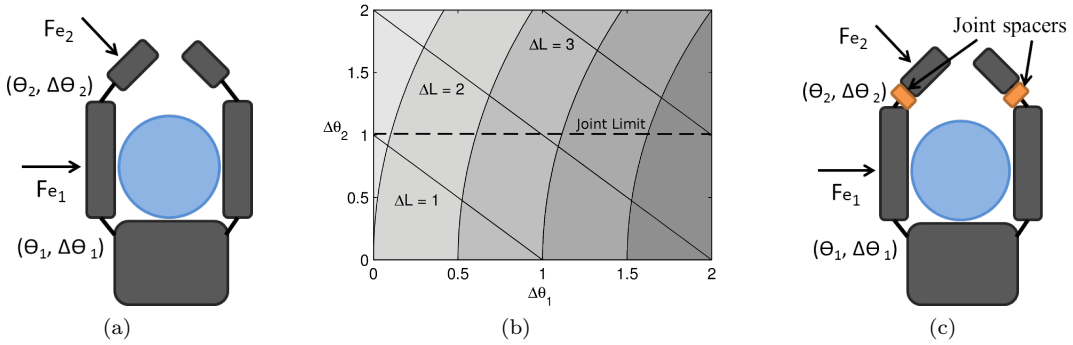
(b)

Figure 18: Testing setup with object anchored to table by a cable and force sensor. (a) iRobot hand (b) Sandia hand

The end effector was moved into a human selected ideal grasp location above the object. A series of grasp tests were then performed where the hand was commanded to the ideal grasp position with an increasing incremental offset to mimic position errors. After grasping, the arm was moved upward slowly until the object was dropped and the maximum force exerted prior to the grasp slipping was recorded. After analyzing the results, it was found that the iRobot hand was nearly twice as strong as the Sandia hand and was less likely to break. Despite the downside of reduced dexterity, we selected the iRobot hand because of its strength, reliability, and robustness.

During practice with the iRobot hands on the wall task, we noticed a significant amount of vibration while using the drill. It was determined that this was due to the underactuated nature of the hand and the lack

of having distal phalanges in contact with the drill (see Fig. 19a). Based on a static analysis of the fingers in contact with the drill, it was determined that for a fixed finger cable excursion length ΔL, the contact forces on the object would be reduced if the distal joint angle $\Delta\theta_2$ is increased and the proximal joint angle $\Delta\theta_1$ is decreased (see Fig. 19b), resulting in the loss of grip force and the object slipping.



(a) (b) (c)

Figure 19: (a) Grasping configuration with distal phalanges not making contact with object ($F_{e_2} = 0$). (b) Underactuation force response for given grasp configuration where darker region is higher contact force F_{e_1} with object. Increasing cable excursion length ΔL increases overall contact force given that there is a joint limit for joint two. (c) Image of the finger spacers added to create joint limit to increase grip strength.

This relaxing of the grip could be overcome by increasing the cable excursion length, but would result in permanent joint damage as the required force in the grip to restrain the drill would cause the distal joint to flex beyond the safe limit. To increase grip force in the proximal phalanges without damaging the distal joints, we added joint spacers which introduced a joint limit on the distal link (see Fig. 19c). With the spacers added, the distal joint would increase until the limit was reached, resulting in further cable length changes increasing $\Delta\theta_1$ and thus increasing the grip force F_{e_1} on the object without risking damage to the finger joints.

8 DRC Trials Results

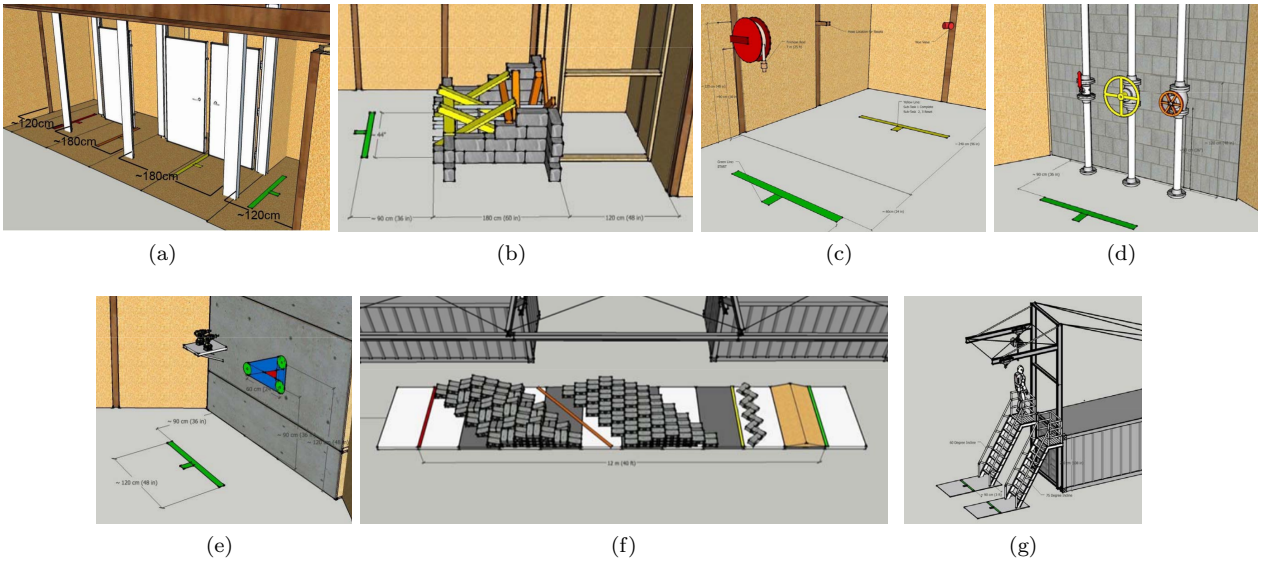
In this section, we describe the tasks of the DRC Trials, how Team ViGIR approached each task, and the results during the Trials. The approaches were based on lessons learned during testing with the Atlas robot, and an assessment of current system limitations. Supplemental video recordings of task performance for all tasks is available online[30]; we recommend viewing these after reviewing the descriptions below. After descriptions of individual tasks, we discuss lessons learned based on a timeline schematic that provides further insight into performance at the Trials.

8.1 Overview

The DRC Trials took place at Homestead-Miami Speedway on December 20-21, 2013. Teams competed in a total of 8 tasks, with only one attempt at each task allowed per team. The time limit for each task was 30 minutes. There was a maximum score of 4 points per task. Points could be scored by completing pre-defined subtasks. For instance, turning one of the three valves in the valve tasks was worth one point each. In case of robots getting stuck or falling, teams could call for an intervention, allowing restarting the robot at the start of the current subtask and incurring an automatic 5 minute penalty. If a team scored 3 points without a intervention in between, a fourth point was awarded. See (Krotkov, Eric, 2013) for a comprehensive overview of the DRC Trials rules. Team ViGIR performed five tasks (door, debris, hose, valve, wall) on the first day and two tasks (terrain, ladder) on the second. The driving task was not attempted by the team. Due to a

[30] http://www.youtube.com/playlist?list=PL5Ku_0Pgk5eKbi9dYWrNoX_j4t6tOr9d5

hardware issue, the robot had a major repair in the night before the first competition day, which limited our ability to perform kinematics calibration before competition start.



(a) (b) (c) (d)

(e) (f) (g)

Figure 20: Schematics of Tasks attempted by Team ViGIR at the DRC Trials: (a) Door (b) Debris (c) Hose (d) Valve (e) Drill (f) Terrain (g) Ladder. All schematics provided by DARPA.

8.2 Team ViGIR performance at the Trials

8.2.1 Door

The Door task consisted of crossing through three different doors. The first door was a "push" door, the second was a "pull" door and the third was a "weighted pull" door (see Fig. 20a). A point was given for crossing each door. Since turning a door handle requires low dexterity no robotic hand was used. The robot was equipped with a hook on each hand which was then used to turn the handles and move the doors (see Fig. 21a).

To open the door, our standard proceedure is to estimate the door handle pose using an object template. Based on this template, footstep plan is computed to move the robot into a manipulation position. Once reached, the IK positions of the arm to turn the handle are validated using the ghost robot and once the operator is satisfied with them, the robot arms are moved. With the door opened, the next step is to calculate a footstep plan taking the robot from the current pose into a pose across the door automatically. During lab-based testing prior to the DRC Trials, we made many successful attempts to walk through the door using autonomous and manual planning. During the DRC Trials door task, automated planning failed as happened occassionally during testing as the doorway is very narrow and noise in sensor data can lead to it appearing blocked. The operator switched to semi-autonomous planning for this reason, selecting footstep plans manually. Sending a manually selected footstep plan, the robot touched the door frame with the right hook hand, leading to a fall of the robot. A similar failure occurred in a second attempt after an intervention was called; at this point, time ran out and no points were scored in this task.

8.2.2 Debris

The Debris task consisted of having to remove debris from a doorway, in order to be able to walk through it. The debris consisted of 10 pieces of balsa wood of different dimensions and an aluminium truss (see

Fig. 20b). A point was given for removing sets of 5 pieces of debris and the last point was given for crossing the open doorway. The robot was equipped with a hook and an extended iRobot hand (see Fig. 21b).

Using an object template, the intended robot location with respect to the first debris piece is specified and is used to obtain a footstep plan to send the robot to this pose. Pre-Trials testing showed that removing debris pieces one by one was time consuming and error prone, as accidentally dropping a debris piece could introduce long time delays. When the grasp on the first piece failed when trying to remove it and it fell back into the debris pile, the operator team changed strategy and attempted to pull the whole truss out of the way using the hook hand. Even though more than 5 pieces of debris were moved, they were not located completely outside the rectangle specified in the rules before time elapsed; no points were obtained in this task.

8.2.3 Hose

The Hose task consisted of having to take a hose and attach it onto a wye by screwing it to a threading coupling. The hose was mounted on a reel separated from the wye by several meters so the robot had to grasp the hose and walk with it towards the wye (see Fig. 20c). A point was given for walking past a defined line with the hose, a second point was given for touching the hose coupling to the wye and the third point was given for attaching the hose to the wye. The robot was equipped with an iRobot hand to grasp the hose and align it with the wye, and a hook to turn the threaded coupling (see Fig. 21c).

Using the hose template and a footstep plan, the robot was sent to a position where the hose was picked up using the iRobot hand. While holding the hose, a footstep plan was then calculated to send the robot into a position in front of the wye. We scored the first two points, but were unable to attach the hose to the wye. The hook was used to turn the coupling a couple of times. Given that attaching the hose requires high accuracy, small misalignments caused the threads not to engage so that the hose fell to the floor when released. The primary challenges were the limited field of view of the cameras, the lack of tactile feedback and the self-balancing MANIPULATE mode (see Section 5.2) that prevented the robot from applying a constant force between the hose and the wye. According to video analysis, Team ViGIR was the fastest Atlas team to score the second point in this task.

8.2.4 Valve

The Valve task consisted of opening three different valves. The first valve was a lever valve, the second was an 18-inch diameter rotary valve and the third was a 9-inch diameter rotary valve (see Fig. 20d). A point was given for closing the lever valve 90 degrees and the other points were given for closing the rotary valves 360 degrees. The manipulability behaviours described in Sec. 6.2 provided the capability of turning the valves in one single circular movement of the arm. Since using robotic hands would imply a series of grasping/release step actions, the robot was equipped with a hook for each hand (see Fig. 21d).

Using the valve object template and footstep plans, the robot was sent into a position in front of each valve. The lever valve was turned using linear cartesian motion, while the rotary valves were turned using the circular path planning capability as shown in Fig. 15. Team ViGIR scored four points in this task including one point per valve and the bonus point, which was awarded for completing all three valve tasks without intervention.

8.2.5 Wall

The Wall task consisted of a half inch thick drywall panel with a right triangle pattern traced on the front which must be cut out (see Fig. 20e). The robot was set up with the iRobot hand on the left arm and a hook hand on the right arm (see Fig. 21e). Finger spacers were added to all but the trigger finger to increase grip strength but allow full motion of the trigger finger (see section 7.3). This was done so as to maximize the

force applied to the drill and to minimize the amount of vibrations in the drill while cutting laterally. The goal was to cut out the red triangle in the middle without cutting into any of the gray area surrounding the triangle. Cuts must be continuous from one green circle to the next. Two templates were created, one of the drill and the other of the triangle pattern to be cut out. The locations to stand relative to each template was calculated and stored prior to the trial. The Dewalt DCD980M2 drill was selected for the task for the reason that it would be easier to turn on and kinematically easier to perform the cutting operation within the field of view of the Multisense sensor. Since limited reachability of Atlas arm was already determined to be an issue, we opted to use the drill which would minimize this risk.

We scored zero points due to a slip of the grasp while performing the first vertical cut. Since the cut was being made downward, the resulting force was greater than the grip force could withstand, causing the hand to slip and lose grasp of the drill trigger. After the loss of the grasp on the trigger, we were unable to recover in time to complete the task. Loss of grip on the trigger could have been mitigated by starting the cut from the bottom and proceeding upward; however, selection of the other drill tool would have eliminated this issue and perhaps performed better as demonstrated by several of the other teams.

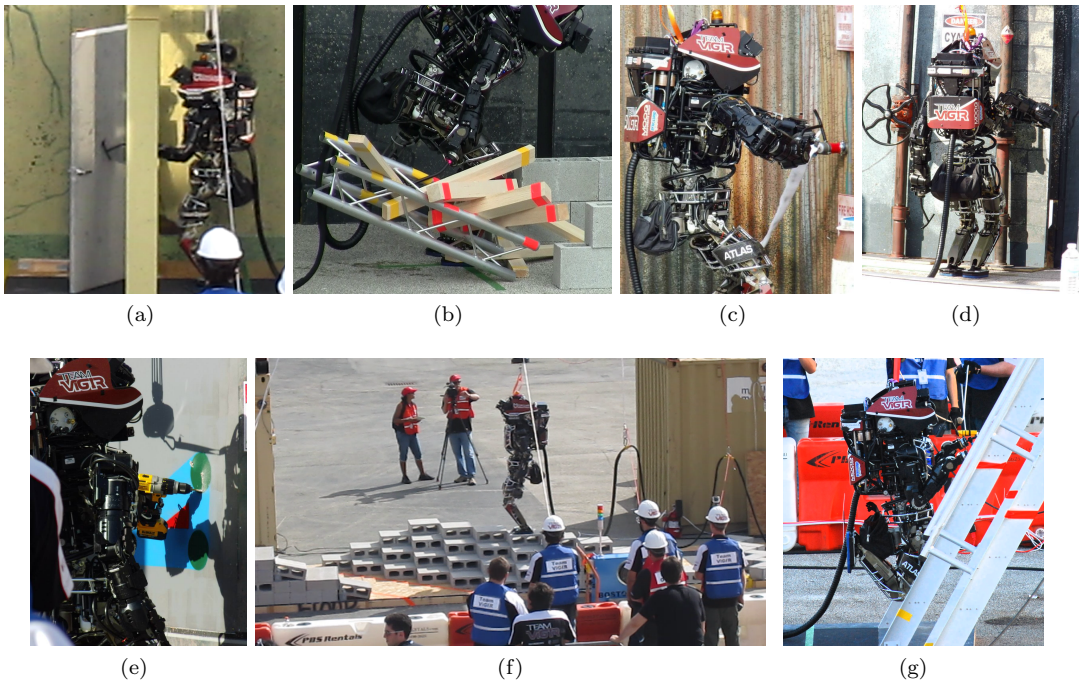


Figure 21: Participation in the DRC Trials: (a) Opening the first door (b) Pulling the truss out in the debris task (c) Attempting to connect the hose to the wye (d) Rotating the first valve (e) Using the drill (f) Stepping over cinderblocks (g) Robot on ladder with all support points above the floor.

8.2.6 Driving

The Driving task consisted of using the robot to drive a Polaris Ranger vehicle through a slalom course. During the Trials, the robot could be placed into the vehicle at the start, but had to drive the length of the course to receive a single point. Two additional points were earned by having the robot climb out of the vehicle, and walk across the finish line. Doing both parts successfully would have earned the team four points.

Given limited practice time, and the risk of hardware damage to the robot in this multi-contact situation, the team chose not to pursue the driving task and therefore scored zero points for this task.

8.2.7 Terrain

The Terrain task consisted of three parts with different requirements and difficulties (see Fig. 20f). The first part consisted of a pitch ramp and a chevron hurdle. The robot walked autonomously over the pitch ramp and adapted the foot attitude to the underlying ramp. The robot also autonomously planned and executed stepping over the chevron hurdle without any issues, and therefore scored one point.

The second part consisted of stairs with flat surfaces. The planner failed in this situation due to noise in 3D world model data. The flexibility of our UI allowed the operator to manually plan single steps up each block, and the robot successfully achieved the top step (see Fig. 21f). Unfortunately, during the initial step down the robot fell due to a weakness of the knee joint in this configuration. During our hurried second attempt after an intervention, the right robot foot caught a neighboring block on the step down and the robot fell a second time. In the end, we earned only one point from this task.

8.2.8 Ladder

The Ladder task consisted of having the robot ascend a ladder (see Fig. 20g). A point was given for having all the contact points above the first step, the second point over the fourth step and the last point with all contact points above the landing. Due to the high weight of the robot, it was equipped with a hook on each hand to be able to support itself while stepping (see Fig. 21g).

Using the ladder template, the robot location relative to the first step was previously calculated and used to obtain a footstep plan to send the robot into this position. The first point was obtained using the hooks to support the robot from the fourth step, then bending the knees in USER mode such that the knees rested on the first step, and thus had all the contact points above the first step. During a second try, the robot was commanded to do a walking step with a defined height. Although the movements of the robot looked promising, the robot fell while transitioning its weight to the upper foot.

8.3 Overall Trials Performance

Considering Fig. 22 and analyzing performance across all tasks, general observations can be made. The speed at which a task was performed was generally determined by the speed and proficiency of the primary OCS operator, who in turn relied on OCS and onboard systems. The secondary OCS operator took over the tasks of sensor data acquisition and template alignment, offloading work from the primary operator and accelerating operations due to parallelization between operators. The robot system was moving less than 50% of total task time, indicating that the bottleneck for task execution was on the operator side, with the robot waiting for commands most of the time. In only a few cases was the opposite true, with operators waiting for the robot to finish motion execution.

Analyzing task performance at the Trials, the following reasons for the operators being the bottleneck have been observed: The performance of robot state estimation, while sufficient for locomotion, was not reliable enough for performing automated grasping in the competition. Both internal (kinematics calibration) and external (pose estimation) state estimation errors contributed to the operator needing to close the end-effector positioning feedback loop manually, adjusting end-effector poses. A contributing factor was the previously mentioned robot repair in the night directly preceding the competition, which precluded proper calibration. As manipulation in contact was required for nearly all tasks and the team wanted to reduce the likelihood of failures and resulting interventions using position control, the speed of trajectory execution was reduced considerably compared to what the robot hardware is capable of.

The overall approach described in this paper proved to work well; however, the previously mentioned issues made task execution very slow. The task failures were due to insufficient operator training (door, debris), bad decisions about procedures (wall), or hardware faults (terrain), and were not fundamentally due to the

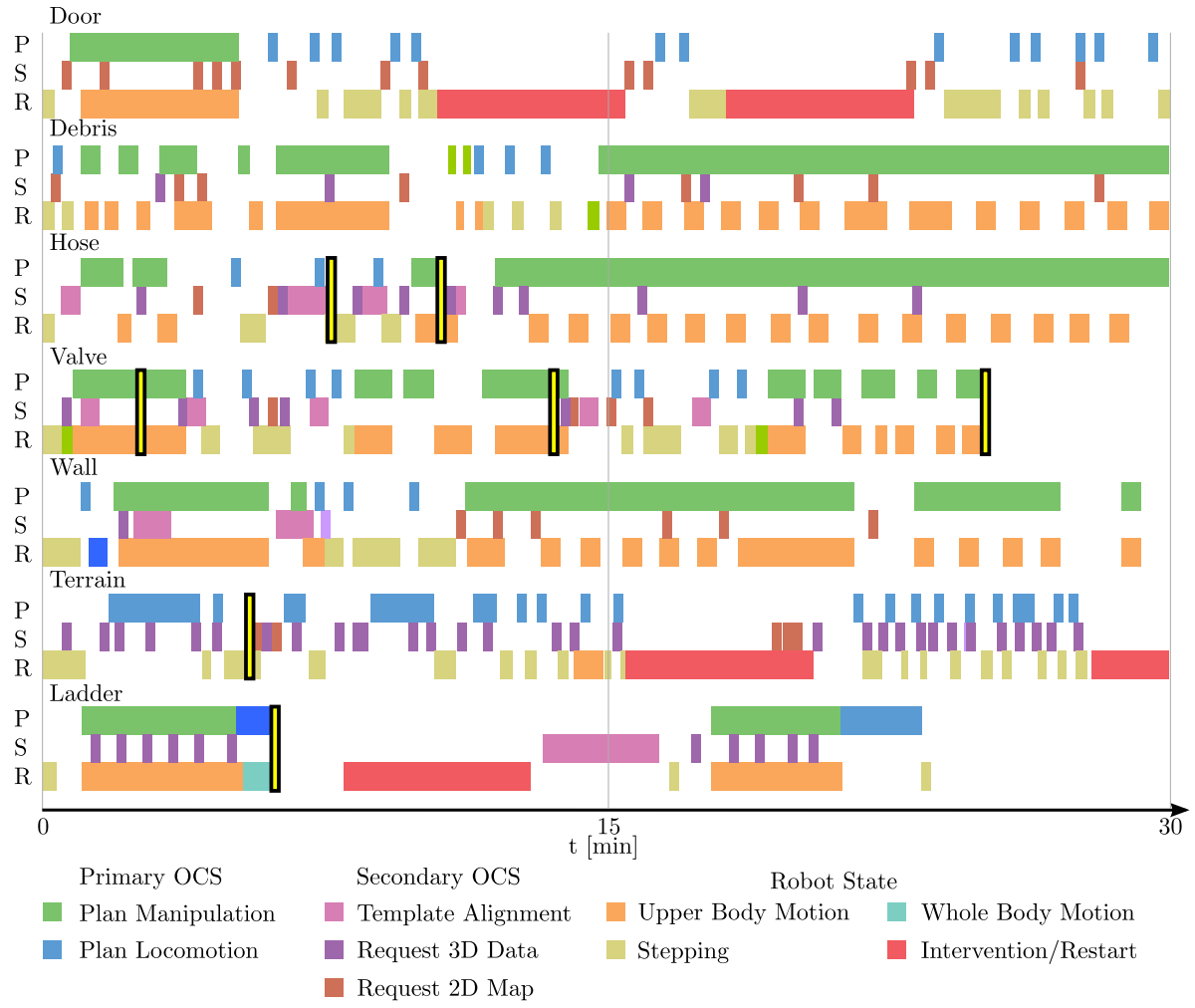


Figure 22: Overview schematic showing the time and task distribution among Primary OCS (P) , Secondary OCS (S) and robot state (R). Scoring events are marked using yellow bars with black border.

approach or software. In future work the following shortcomings noted in the Trials will be addressed, with the goal of increasing reliability and speed of task execution:

- Reliable and fast kinematics calibration is required for reliable autonomous manipulation.
- Drift compensation for robot pose estimation is required for locomotion as well as manipulation.
- Friction compensation will be investigated to enable low gain position control with gravity compensation or full force control.
- Automated pose estimation and grasp planning for objects of interest indicated by the operator(s)
- Whole body planning to leverage the high number of DOF and extending the reachability of the system.
- Whole body control to address the driving and ladder tasks.

9 General Lessons Learned

Beyond the technical discussion of performance at the Trials in the previous section, a number of lessons were learned over the course of the project. These include both successes that we can build upon, and failures that taught important lessons as we move forward to the next phase.

Team Coordination is the most important issue to be tackled in an extremely time constrained and complex project such as DRC participation. With Team ViGIR members being spread over more than 9000km, setting up tools and procedures was crucial. Using the Redmine project management system for issue tracking and documentation and coordinating via teleconferences weekly and additionally whenever needed, the team successfully tackled this challenge.

Complex Software Projects require all members of the involved team(s) to have easy access to software in order to be able to build and test software as well as follow conventions during development. Within Team ViGIR, great care was taken to guarantee this at all times by standardizing OS and ROS versions used; the installation, update and compilation processes were simplified using scripts. The power of the distributed version control system *Git* was utilized for our own software development as well as to quickly fork and extend upon existing open source software and use feature branch based development.

Simulation is an important tool to be able to validate and test complex robotics software. With an international team, this becomes even more crucial, as access to the real robot system is limited. The robot API available for drcsim and for the real Atlas robot diverged due to rapid development of the latter when the real robot became available. This meant that drcsim was only of limited use preceding the DRC Trials, a situation that caused development delays for the team.

Operator Training is crucial to overall system performance. Due to time and resource constraints as well as required expertise, the operator team was made up of core team members involved in onboard and OCS software development. This restricted the amount of time that could be dedicated to training. Given the infrastructure that is now in place, we plan to put an emphasis on operator training and practice in the next phase.

Multiple Operators Approach worked great as a "Wizard of Oz" speech interface, in which the main operator says, "May I have a point cloud?" and secondary operator does it. We plan to extend these functionalities to include new ways of communication and collaboration between operators, such as allowing the operators to mark and annotate point clouds. We intend to significantly increase the amount training and practice time where the operators function in defined roles prior to competition.

Interaction for Manipulation is a major challenge, as 6 DOF poses have to be specified in a fast and intuitive way. Using keyboard and mouse as a baseline approach, we will evaluate advanced methods for interactions as future work.

Understanding Complex 3D Structures was hard, especially because it was difficult to judge depth. We plan to evaluate the use of displays that provides higher levels of display fidelity (objective level of sensory stimuli (Slater, 2003)), such as a head-mounted display with stereoscopy and head-tracking, and how it can be integrated into the workflow of completing tasks with multiple operators.

Operator Control Station Layout is crucial for enabling interaction of the operator with the robot system. For the primary operator, our four screen setup worked well during competition; however, this setup could not be easily replicated on other machines for operator training.

User Interface design must be simple and intuitive to facilitate operator training. During development, numerous engineering widgets were created for testing and specialized controls. While the ability to rapidly prototype interfaces using *rqt* was crucial for enabling us to complete development, the resulting interface was overly cluttered and did not satisfy our design criteria. A major focus during the next phase is on streamlining the interfaces, and integrating control functions.

Multiple Perspectives are crucial to success. The limited FOV of the robot head and lack of a yaw joint on the neck limited perception. We found the cameras in the Sandia hands useful for providing another point of view during testing, but chose to use different configurations at competition for several reasons. We intend to revisit additional sensing leading up to the DRC Finals.

10 Conclusion

In this work, we present an overview of Team ViGIR's approach for solving complex rescue tasks using teaming between a robot system and human operators. The DRC scenarios are especially challenging because of severe bandwidth constraints and the use of complex human tools by a humanoid robot system. This paper provides an overview of many (but not all) of the important components in our onboard and operator control system. We describe our performance in the DRC Trials competition on a per task basis and related video data is linked. Lessons learned are discussed both from a single performance issue and from a "bigger picture" human factors/project management view point.

We plan on releasing all Team ViGIR software not bound by non-disclosure agreements as open source after the DRC Finals to facilitate knowledge transfer and comparison with work by other researchers.

11 Acknowledgments

This project was supported by the Defense Advanced Research Projects Agency (DARPA) under Air Force Research Lab (AFRL) contract FA8750-12-C-0337.

Team ViGIR would like to thank DARPA and its support staff for a well run Trials, the Open Source Robotics Foundation (OSRF) for their simulation support, and Boston Dynamics, Inc. for their support with the Atlas robot. The team would also like to thank the contributors and maintainers of the Robot Operating System (ROS), MoveIt!, GraspIt!, Octomap, rviz, rqt_gui, calibration, footstep_planner, Simox and Rigid Body Dynamics (RBDL) libraries.

In addition, the co-authors would like to thank our other team members who provided support during development: TU Darmstadt - Karen Petersen, Johannes Meyer, Dorian Scholz, Philipp Schillinger, Jochen Mück, Michael Stahr, Omid Pahlevan-Sharif, Mike Smyk, Sebastian Gauert; Virginia Tech - Tania Hamid, Lindsay Blassic, Jacob Shepphard, Alex Little, Tony Angel; Oregon State - Jordan Meader, Zhifei Zhang, TORC Robotics - David Anderson, Adam Bowling, Janette Brown, Kevin Collins, Jesse Hurdus, Andrew Lycas, Chris Sammet, and Ben Waxler.

References

Alunni, N., Phillips-Grafftin, C., Suay, H. B., Lofaro, D., Berenson, D., Chernova, S., Lindeman, R. W., and Oh, P. (2013). Toward a user-guided manipulation framework for high-dof robots with limited communication. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, pages 1–6. IEEE.

Blodow, N., Goron, L. C., Marton, Z.-C., Pangercic, D., Ruhr, T., Tenorth, M., and Beetz, M. (2011).

Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4263–4270. IEEE.

Bruemmer, D., Dudenhoeffer, D., and Marble, J. (2002). Dynamic autonomy for urban search and rescue. In *In Proc. 2002 AAAI Mobile Robot Workshop*, Edmonton, Cananda.

Chitta, S., Sucan, I., and Cousins, S. (2012). MoveIt! *IEEE Robotics Automation Magazine*, 19(1):18–19.

Dai, H., Valenzuela, A., and Tedrake, R. (2014). Whole-body motion planning with simple dynamics and full kinematics. *Under review.*

Deits, R. and Tedrake, R. (2014). Footstep planning on uneven terrain with mixed-integer convex optimization. *Under review.*

Desai, M. and Yanco, H. (2005). Blending human and robot inputs for sliding scale autonomy. In *Robot and Human Interactive Communication, 2005. ROMAN 2005. IEEE International Workshop on*, pages 537–542.

Fallon, M., Kuindersma, S., Karumanchi, S., Antone, M., Schneider, T., Dai, H., Perez D'Arpino, C., Deits, R., DiCicco, M., Fourie, D., et al. (2014a). An architecture for online affordance-based perception and whole-body planning. *Under review.*

Fallon, M. F., Antone, M., Roy, N., and Teller, S. (2014b). Drift-free humanoid state estimation fusing kinematic, inertial and lidar sensing. *Under review.*

Felis, M. (2012). RBDL - the rigid body dynamics library. `http://rbdl.bitbucket.org`. Accessed: 2014-03-07.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education.

Goodrich, M. A. and Schultz, A. C. (2007). Human-Robot Interaction: A survey. *Found. Trends Hum.-Comput. Interact.*, 1(3):203–275.

Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., and Lepetit, V. (2011). Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 858–865. IEEE.

Hornung, A., Dornbush, A., Likhachev, M., and Bennewitz, M. (2012). Anytime search-based footstep planning with suboptimality bounds. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 674–679. IEEE.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, pages 1–18.

Huang, H.-M., Messina, E., and Albus, J. (2007). Autonomy levels for unmanned systems (alfus) framework volume ii: Framework models version 1.0. NIST Special Publication 1011-II-1.0, NIST.

Khatib, O. (1987). A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation*, RA-3(1):43–53.

Koch, M. D. (2010). Utilizing emergent web-based software tools as an effective method for increasing collaboration and knowledge sharing in collocated student design teams.

Kohlbrecher, S., Conner, D. C., Romay, A., Bacim, F., Bowman, D. A., and von Stryk, O. (2013). Overview of Team ViGIR's approach to the Virtual Robotics Challenge. In *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, pages 1–2. IEEE.

Koolen, T., Smith, J., Thomas, G., Bertrand, S., Carff, J., Mertins, N., Stephen, D., Abeles, P., Englsberger, J., McCrory, S., van Egmond, J., Griffioen, M., Floyd, M., Kobus, S., Manor, N., Alsheikh, S., Duran, D., Bunch, L., Morphis, E., Colasanto, L., Ho Hoang, K.-L., Layton, B., Neuhaus, P., Johnson, M., and Pratt, J. (2014). Summary of team IHMC's Virtual Robotics Challenge entry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE.

Krasner, G. E., Pope, S. T., et al. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49.

Krotkov, Eric (2013). DRC Trials Rules. `http://www.theroboticschallenge.org/files/DRCTrialsRulesRelease7DISTAR22157.pdf`. Accessed: 2014-07-22.

Leeper, A., Hsiao, K., Ciocarlie, M., Sucan, I., and Salisbury, K. (2013). Methods for collision-free arm teleoperation in clutter using constraints from 3D sensor data. In *IEEE Intl. Conf. on Humanoid Robots*, Atlanta, GA.

Lischinsky, P., Canudas-de Wit, C., and Morel, G. (1999). Friction compensation for an industrial hydraulic robot. *Control Systems, IEEE*, 19(1):25–32.

Loeliger, J. and McCullough, M. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.".

Miller, A. and Allen, P. K. (2004). GraspIt!: A versatile simulator for robotic grasping. *IEEE Robotics Automation Magazine*, 11(4):110–122.

Murphy, R. R. and Burke, J. L. (2005). Up from the rubble: Lessons learned about HRI from search and rescue. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 49, pages 437–441. SAGE Publications.

Nagatani, K., Kiribayashi, S., Okada, Y., Otake, K., Yoshida, K., Tadokoro, S., Nishimura, T., Yoshida, T., Koyanagi, E., Fukushima, M., et al. (2013). Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots. *Journal of Field Robotics*, 30(1):44–63.

Nakanishi, J., Cory, R., Mistry, M., Peters, J., and Schaal, S. (2008). Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757.

Odhner, L. U., Jentoft, L. P., Claffee, M. R., Corson, N., Tenzer, Y., Ma, R. R., Buehler, M., Kohout, R., Howe, R. D., and Dollar, A. M. (2013). A compliant, underactuated hand for robust manipulation. *International Journal of Robotics Research*.

Pradeep, V., Konolige, K., and Berger, E. (2014). Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach. In *Experimental Robotics*, pages 211–225. Springer.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3.

Rosson, M. B. and Carroll, J. M. (2002). *Usability engineering: scenario-based development of human-computer interaction*. Morgan Kaufmann.

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.

Sandia National Laboratories (2014). Sandia hand. `http://www.sandia.gov/research/robotics/advanced_manipulation/Sandia_Hand.html`. Accessed: 2014-03-07.

Scholtz, J. (2003). Theory and evaluation of human robot interactions. In *HICSS '03 Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*.

Sentis, L. (2007). *Synthesis and Control of Whole-Body Behaviors in Humanoid Systems*. PhD thesis, Stanford University, Stanford, USA.

Shoemake, K. (1985). Animating rotation with quaternion curves. *ACM SIGGRAPH computer graphics*, 19(3):245–254.

Slater, M. (2003). A note on presence terminology. *Presence connect*, 3(3):1–5.

Stumpf, A., Kohlbrecher, S., Conner, D. C., and von Stryk, O. (2014). Supervised footstep planning for humanoid robots in rough terrain tasks using black box walking controller. *Under review*.

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. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Tuley, J., Vandapel, N., and Hebert, M. (2005). Analysis and removal of artifacts in 3-d ladar data. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2203–2210. IEEE.

Vahrenkamp, N., Asfour, T., and Dillmann, R. (2013). Robot placement based on reachability inversion. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1970–1975. IEEE.

Van Dam, A. (1997). Post-WIMP user interfaces. *Communications of the ACM*, 40(2):63–67.

Yanco, H. A. (2004). Classifying human-robot interaction: An updated taxonomy. In *Proc IEEE SMC*, pages 2841–2846.