2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

From Autonomy to Cooperative Traded Control of Humanoid Manipulation Tasks with Unreliable Communication: System Design and Lessons Learned

Jim Mainprice[1], Calder Phillips-Grafflin[1], Halit Bener Suay[1], Nicholas Alunni[1], Daniel Lofaro[2], Dmitry Berenson[1], Sonia Chernova[1], Robert W. Lindeman[1], Paul Oh[2]

[1] *Worcester Polytechnic Institute, {jmainprice, cnphillipsgraffl, benersuay, nalunni, dberenson, soniac, gogo}@wpi.edu*

[2] *Drexel University, dan@danLofaro.com, paul@coe.drexel.edu*

***Abstract*— In this paper, we report lessons learned through the design of a framework for teleoperating a humanoid robot to perform a manipulation task. We present a software framework for cooperative traded control that enables a team of operators to control a remote humanoid robot over an unreliable communications link. The framework produces statically-stable motion trajectories that are collision-free and respect end-effector pose constraints. After operator confirmation, these trajectories are sent over the data link for execution on the robot. Additionally, we have defined a clear operational procedure for the operators to manage the teleoperation task. We applied our system to the valve turning task in the DARPA Robotics Challenge (DRC). Our framework is able to perform reliably and is resilient to unreliable network conditions, as we demonstrate in a set of test runs performed remotely over the internet. We analyze our approach and discuss lessons learned which may be useful for others when designing such a system.**

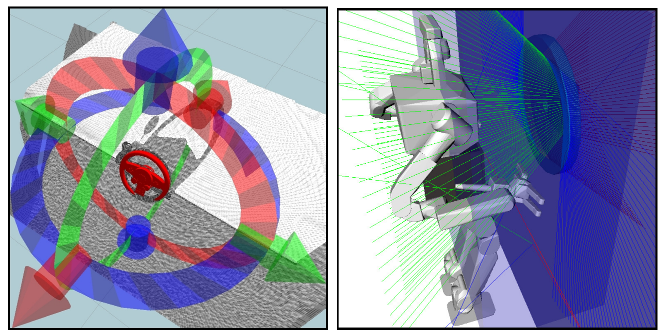

Fig. 1: The operator identifies and localizes a valve in a point cloud using an interactive marker (left), visualizes the motion planned to maximize the turn angle of the valve by testing multiple hand placements before execution (right).

I. Introduction

The 2011 tragedy of Fukushima has shown the limitations of state-of-the-art disaster response robotics. This led to the 2013 DARPA Robotics Challenge (DRC) Trials, where roboticists were invited to compete on eight tasks representative of those encountered in a disaster recovery scenario. To achieve those tasks, highly mobile and dexterous robots were required. Humanoids are especially well suited for the tasks, as they take place in human environments. Furthermore, to simulate a disaster scenario, the communication between the operator and the robot was made unreliable. This setup prevents operators from accessing sensor feedback at a high rate, which limits situational awareness, and thus they cannot accurately monitor the robot's actions in real time.

The task considered in this paper consists of manipulating an object whose size and location are not known *a priori* but whose general shape is known. Thus, we need a framework which is able to navigate the robot to a location where the object can be manipulated, grasp locations must be determined, and trajectories must be generated dynamically. In this paper, we focus on our implementation for the valve-turning task of the DRC, but we emphasize that our framework can be adapted to other humanoid manipulation tasks, as discussed in Section III-D.1.

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) award #N65236-12-1-1005 for the DARPA Robotics Challenge. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

Our framework is composed of four primary components: (1) an operator-guided perception interface which provides task-level commands to the robot, (2) a motion planning algorithm that autonomously generates robot motions that obey relevant constraints, (3) a data aggregation and "unreliable communication teleoperation" toolkit, which we have release as an open-source package [1], that limits the traffic on the data link and makes the system resilient to network dropouts and delays, and (4) an operational protocol that dictates how the team of operators must act to operate the robot.

Our implementation of the framework allows operators to specify the manipulation task in a straightforward manner by setting the pose and dimensions of the object to be manipulated in a 3D display of the point clouds acquired by the robot, as shown in Figure 1. The system then allows operators to plan a feasible trajectory, and once they have validated it using a previewing system (also shown in Figure 1), send it to the robot for execution. Even though this close monitoring of the motion planner could potentially be slower than a more autonomous approach, we believe the gains from exploiting the situation awareness capabilities of the operators enhance the overall reliability of the system. Also, this loss in efficiency is acceptable as our intent is not to solve manipulation tasks at high speed.

This paper does not focus on providing novel techniques for robot control. Instead, we provide a description of our system design and how the design changed from autonomy to teleoperation over time. We discuss what we learned through the design process that could be useful to others, including why some standard techniques could not surpass the ability of well-trained operators on critical tasks, such as localizing objects in point cloud data or monitoring errors in trajectory execution. We also discuss the performance and limitations of our motion planning and control approaches as well as alternative approaches for teleoperating the task.

II. Related work

Prior work in the area of disaster recovery robotics [2], [3] has revealed the need for better group organization, perceptual and assistive interfaces, and formal models of the state of the robot, state of the world, and information as to what has been observed. In this work, we focus on the problem of teleoperating a single humanoid robot remotely under unreliable communication.

A. Teleoperation of humanoids

Teleoperating a humanoid robot involves controlling actions such as head motions, grasping and manipulation of objects, and navigation. The case of teleoperating a humanoid robot is particularly difficult due to the high number of Degrees of Freedom (DoFs) and the constraint to maintain balance. A recent survey on teleoperation for humanoids can be found in [4], in which the authors list three types of challenges to teleoperate humanoids, two of which are relevant for disaster recovery: 1) physical properties of humanoids (e.g., high DoFs, morphology) and 2) operator-based (e.g., situational awareness, skill, and training). This paper addresses both types of challenges.

There are many architectures that attempt to tackle the problem of manipulation performed by a robot with a mobile base [5]. The problems present in these mobile manipulation tasks include where to place the robot in relation to the object to be manipulated [6], how to grasp the object [7], and how to plan the robot's movements [8]. To the best of our knowledge, there is no available framework for humanoid robots, unlike UAVs or rovers, that is tailored for operator-guided object manipulation in unstructured environments with limited communication to the robot.

B. Managing autonomy in teleoperation

Any robot to be teleoperated presents a certain level of autonomy. Managing this autonomy is crucial in the design of a teleoperation system. Supervisory control [9] defined by Sheridan [10] as a process in which "one or more human operators are intermittently programming and continually receiving information from a [robot] that itself closes an autonomous control loop through artificial effectors to the controlled process or task environment" provides a good framework to classify different control approaches. However, other terminologies and methodologies have since emerged to describe ways of managing the robot's autonomy, the three most relevant for our task are defined in [4]:

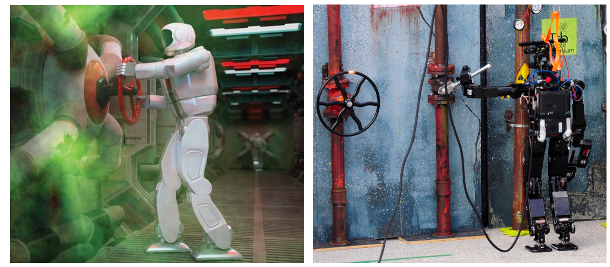

Fig. 2: Hubo2 Plus (left) in a simulated environment and DRCHubo (right) at DRC trials turning a lever valve

1) Direct control: The operator manually controls the robot; no autonomy is involved on the robot side. The robot is controlled in a master-slave interaction. For example, a direct-control approach is the control of each DoF of a manipulator using a joystick.

2) Traded control: In traded control the operator and the robot both control the robot's actions. The operator initiates a sub-task or behavior for the robot and the robot performs the sub-task autonomously while the operator monitors the robot. For instance, in [11] an interactive robot is teleoperated by selecting a state for the robot to act. Our approach is based on traded control in that the operator only specifies the target valve to initiate the sub-task, which is then performed by the robot using autonomous motion planning.

3) Collaborative control: This mode corresponds to high-level supervision where the robot is considered to be a peer of the operator. The role of the operator shifts from an operator who dictates every movement, to a supervisor who guides at a high-level. This approach is often used for unmanned systems controlled from a central command post [12], [13].

When a team of operators controls a robot in either of those modes the strategy is called "Cooperative". The method presented in this paper is a form of *Cooperative Traded Control*. In Section V we discuss our approach and compare it to alternative control strategies.

C. Teleoperation with low-bandwidth

Highly unstructured disaster environments, such as those we target in this work, provide a challenge where communication can be difficult due to the unknown properties of the building materials, making transmission and reception of signals unreliable [14]. The framework we present in this paper is meant to operate seamlessly with a latency between zero and five seconds, with packet loss and periodic dropouts that are analogous to communication in a demolished building.

Low-bandwidth communication covers a broad range of research, which can be categorized by the amount of delay that the system attempts to handle. Typically, these categories are roughly 0-2 seconds, 2-10 seconds, and greater than 10 seconds latency [15]. For instance, many surgical systems operate between zero and two seconds of latency sometimes across distant locations [16]. Latency greater than two seconds is typically found in research related to earth orbit or farther systems, such as Lunar robots [17], [15]. Latency greater than ten seconds extends even farther including the Mars rovers, which have a delay of many minutes [18].

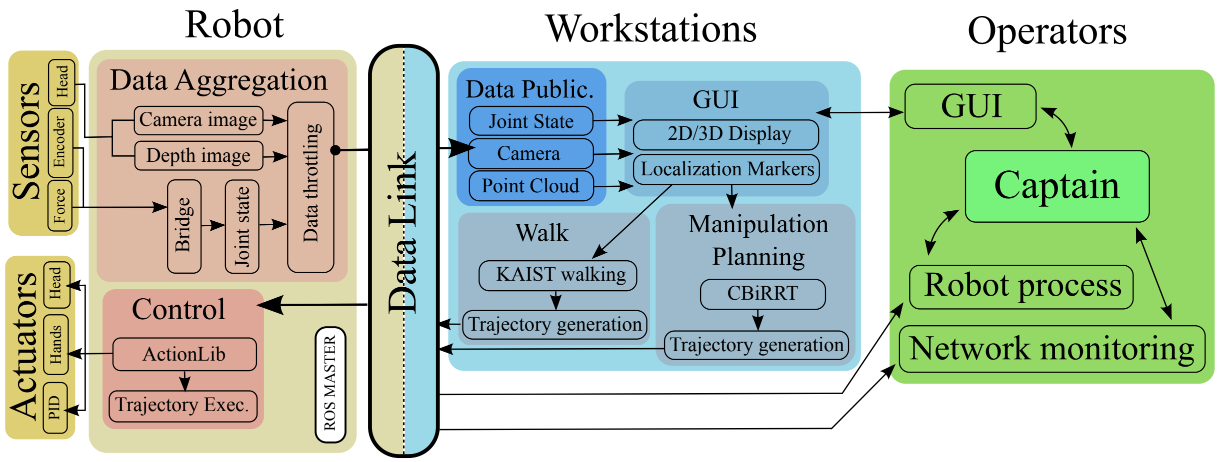

Fig. 3: System diagram showing data flow through the framework. Operator interaction is shown in green boxes.

D. Service-Oriented Architectures

The framework we present has been developed within a Service-Oriented Architecture (SOA) which consists of distinct software modules that communicate with each other [19], [20], [21]. We chose ROS [21] for its ease-of-use and visualization tool (RViz).

III. Framework Description

Our framework is intended for high-DoF robots. We have applied a preliminary version of it to PR2 and Hubo2 Plus robots [22]. In this paper, we describe an application to the DRCHubo robot, developed by the Korean Advanced Institute of Science and Technology (KAIST). DRCHubo has two 6 DoF legs, two 7 DoF arms, and a 2 DoF head. The hands each possess three fingers controlled by a single DoF. DRCHubo is equipped with a sensor head mounted with a tilting Hokuyo LIDAR for providing point cloud data and three cameras for stereo vision.

For legged and high DoFs robots such as DRCHubo, movements of the upper body can make the robot unbalanced. As a result, it is important to consider the center of mass when generating the motions. However, to be able to turn valves with high friction and stiction, the robot must use its maximal capabilities (i.e., using both arms). Thus, the hands must move in a coordinated manner. Our motion planning component, based on the CBiRRT algorithm [8], is able to account for those constraints given the pose and shape of the manipulated object.

A. Architecture

The framework's architecture, shown in Figure 3, shows the data flow through the system. Software modules running on the robot are shown on the left (yellow), while those running on the operator workstations are shown on the right (blue). On the robot, the data aggregation system reads sensor output and packages it into compact messages. The control system receives and executes trajectories.

On the workstations, the Graphical User Interface (GUI) displays data received from the data aggregation system to the operators. The GUI allows an operator to specify the object pose and dimension and send commands to the motion generation systems. Finally, the motion trajectories from the walking generation and the manipulation planning systems are sent to the robot through the data link.

The implementation of the framework relies on ROS [21] for the communication between modules, RViz for the user interface, and OpenRAVE [23] for the motion planning and pre-visualization of trajectories. The walking generation code is based on KAIST's Rainbow walking framework.

All data transmission between the robot and the workstation happens over ROS, however we have implemented our own data-link software to throttle data rates and limit communication to only necessary information.

B. Addressing Unreliable Robot-workstation Comms.

1) Data Aggregation: The primary function of the data-aggregation system is to reformat sensor data used on the workstation so that communication across the restricted datalink is limited. As shown in Figure 3, it simultaneously processes camera images, point clouds, encoder values, and force sensors allowing the framework to be highly modular and quickly adaptable to different robots, as well as being suitable for both real and simulated environments. The system can also be reconfigured during operation to handle changes in the available sensor data, such as selecting an alternate image or point cloud source, or changing the quality of data received.

2) Low-level control: For communication with Hubo's motor controllers we use Hubo-Ach, which is an open-source daemon process that uses a high-speed, low-latency IPC called Ach [24]. Hubo-Ach implements a real-time loop in which all of the motor references and state data are set and updated respectively. The bridge component of the data aggregation system combines joint angle and motor control sensor values read through Hubo-Ach and republishes them as ROS messages. The head sensors (i.e., camera and LIDAR for DRCHubo), are directly controlled through ROS nodes.

3) Data Link: Even with limited data over the link, communication between the robot and the workstation can break when operating under high latency if not handled properly. Hence we have developed an overlay of ROS's communication protocols that is resilient to unreliable networks (i.e., dropouts, and high latency), which we have released as open-source software [1].

The primary components of the teleoperation datalink system are rate-limiting repeaters and relays that allow fine-grained control over the network demands of our system. These relays, based on ROS's non-persistent service calls, replace the simple equivalents provided in ROS and implement automatic detection of network failures, notification and warnings to the operator, and automatic recovery of broken network sockets. For instance, the rate-limiting repeaters allow for data published at a "native" rate of 200Hz on-board the robot to be forwarded to the operator's workstation at a low rate of 10-20Hz, reducing network demands.

In the data throttling component, the user configures compression quality and publishing rates. Black and white camera images are compressed by setting the image resolution and compression quality. Point clouds produced from LIDAR scans are compressed using a voxel filter and a selection of point cloud compression algorithms including ZLIB and PCL's Octree-based compression [25]. Using this approach, we can reduce the data usage of point clouds by over 95%. To avoid flooding the limited data-link with unnecessary data, camera images are requested at only 0.5Hz and point clouds are individually requested by the operator. A new request is attempted only when the previous data request has completed, which means that new data requests are never delayed by old data still being transmitted. To provide transformations between various frames of the robot, an ROS TF tree is maintained on the robot using the current joint state. To limit unnecessary communication of those frames through the link, a separate tree is reconstructed on the operator workstation from joint state data, which requires only a small fraction of the data needed to continuously synchronize the TF tree.

Data	No Compression	ratio 1	ratio 2	Freq. used
Joint State	0.82 KB	-	-	10Hz
Camera	5120 KB	230 KB	4.97 KB	0.5Hz
Pointcloud	10 MB	$\approx$ 100 KB	$\approx$ 50 KB	On demand

TABLE I: Data sizes, compression ratios, and frequencies

Table I reports the data sizes and frequency used for publishing over the link. The compressed camera frame correspond to lower resolution (320x240) with JPEGs compression ratio for ratio 1 and ratio 2 of 50 and 20, respectively. The compression of point cloud corresponds to the compression algorithms mentioned above with no voxel filtering for ratio 1 and voxel size of 0.02m for ratio 2.

C. User Interface

When teleoperating the robot, the operator must be able to monitor the robot state as well as its surroundings to maintain situation awareness. Thus, our GUI, shown in Figure 4, provides monitoring capabilities through 2D camera images as well as a 3D display of the robot configuration and point cloud data. The user can switch what data is displayed on screen using RViz's built-in features. In addition to sensor data, the GUI displays the motion planner error conditions and the system's state through panels, text, and color codes.

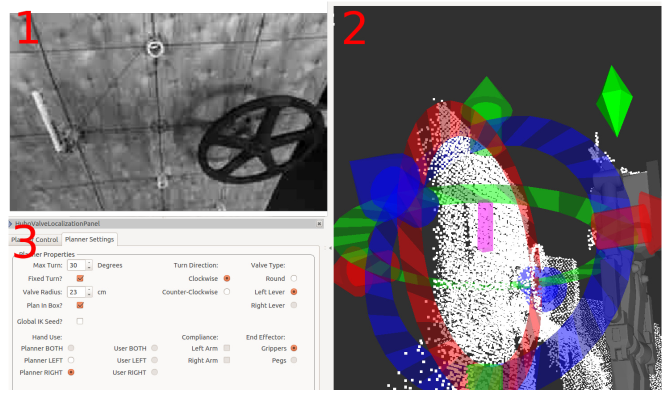

Fig. 4: Screen capture of the Graphical User Interface (GUI). 1) video feed of the lever and round valve, 2) display of point cloud and interactive marker, 3) planner-settings panel.

The operator controls the robot by specifying a set of parameters that are sent to the motion planner and the walking generation module (i.e., end-effector pose constraints, turn angle, etc). We use interactive markers [26] and control panels to determine and input those parameters. Distance and direction for the walking generation are determined using an interactive marker. Valve size, turn amount and which arm(s) to use for the manipulation task are determined using both an interactive marker and the control panels. Before querying the motion planner, all parameters can be verified at a glance by looking at the control panels.

Interactive markers (see Figure 1) provide six-DoF handles, three translation DoFs and three rotation DoFs, which enable the operator to quickly define a pose in a 3D display. Since we use interactive markers to simultaneously select and localize the object to manipulate, we avoid the use of complex object detection and localization algorithms, instead relying on the operator for these capabilities. The shape attached to the interactive marker can be a cuboid, a disk, or a triangle mesh. To specify the pose and dimensions of an object (e.g., lever or disk valve) the operator aligns the shape to point cloud data using the interactive marker. In the DRC, when localizing the valve for walking to a standing position in front of it, the pose estimate does not have to be as precise as for manipulation, usually only requiring one or two DoFs to estimate the walking distance. Hence, the average time to align the marker over one trial while operating the robot was 9.3 secs for walking and 41.6 secs for manipulation.

Once the object is localized and the planner parameters are selected, the operator can send a planning request. The main tab of the planner-settings panel provides basic commands for the motion planner and robot controller. The operator can command which sub-task to plan for (i.e., go to manipulation configuration, turn valve or go back to walking configuration) after which he/she can request a pre-execution visualization of the motion trajectory in a dedicated 3D GUI component (see Figure 1) by clicking a button. This visualization limits potential mistakes made by the operator as well as dangerous robot behavior. Another button lets the operator trigger the robot motion by sending the trajectory over the network to the robot controller. A second tab (see Figure 4) lets the operator specify the turn angle, turn direction (i.e., clock wise or counter-clock wise) and which arm combination to use (i.e., left, right or both).

In addition to operator input and feedback, the GUI controls the data flow over the unreliable link to the robot. The operator can request point clouds and turn on/off the camera image request loop. Thus, data from the robot is transmitted only when necessary to minimize communication.

D. Motion Planning and Execution of Trajectories

Once the object pose and dimensions are set by the operator, the operator can generate the robot's motion using the motion planning component. The paths produced by the motion planner are collision free and respect end-effector pose and balance constraints.

1) Motion Planning: For valve turning, each manipulation task involves three subtasks : 1) **Ready**: a full-body motion that sets the robot's hands close to the valve ready to grasp it, with knees bent, lowering the center of gravity to be more stable, 2) **Turn**: An arm(s) motion that grasps the valve, performs a turn motion, releases the valve and returns to the initial configuration (so it can be repeated without re-planning), 3) **End**: a full-body motion that brings the robot back to the walking configuration. While these motions are specialized for valve turning, the motion planner can be easily reconfigured to manipulate other objects by inputting a different set of constraints [8].

The motion planning component of the system is based on the CBiRRT algorithm [8], which is capable of generating quasi-static motion for high-DoF robots subject to end-effector pose and balance constraints. CBiRRT generates collision-free paths by growing Rapidly-exploring Random Trees (RRTs) in the configuration space of the robot while constraining them to constraint manifolds. Average planning times of CBiRRT for each subtask are reported in Table II.

Valve type	Ready	Turn	End
Lever	1.91 (1.11)	0.99 (0.39)	1.72 (0.91)
Circular	2.71 (1.12)	2.72 (1.55)	2.38 (1.72)

TABLE II: CBiRRT average and st. dev planning time in seconds for DRCHubo on the three valve turning subtasks.

In the valve turning task, the motion must obey constraints defined by the valve pose provided by the operator (see section III-C). The end-effector pose constraints are then specified by Task Space Regions (TSRs) [8] that CBiRRT uses to represent pose constraints. TSRs consist of three parts:

- T_w^0 : transform from the origin to the TSR frame w;
- T_e^w : end-effector offset in the coordinates of w;
- B^w : 6×2 matrix of bounds in the coordinates of w:

In our implementation for valve-turning, we have defined three tasks that the robot can perform: 1) turn a lever with the right hand 2) turn a lever with the left hand 3) turn a circular

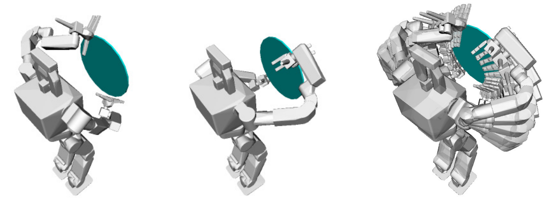

Fig. 5: (a) Start and (b) goal configurations maximizing the turn angle for a two-handed rotation of a round valve. (c) Full motion obeying end-effector pose constraints computed by CBiRRT.

valve with both hands. Each of these tasks corresponds to a specific TSR constraint definition.

In all cases, inverse kinematics [27] is performed to find a whole body configuration given the location and radius of the manipulated object. The TSRs are then defined according to the hand locations when grasping the object and the pose of the object. For instance, the TSR for one-arm lever motions is defined as follows:

$$T_w^0 = T_w^{valve}$$

where T_w^{valve} is the valve pose in the world.

$$T_e^w = (T_w^{valve})^{-1} * T_w^H$$

where T_w^H is the hand pose in the world when grasping the valve.

$$B^w = \begin{bmatrix} 0 & 0 & 0 & \theta & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

where θ is the desired rotation angle of the lever. When planning for full-body motions, we also define TSRs for the feet to keep their position and orientation fixed in the world. In order to perform large turns on the circular valve, we have implemented an algorithm that iterates through interpolated hand placements along the valve to find valid start and goal IK solutions that maximize the turn angle (see Figure 5).

2) Trajectory Execution and Control: The path generated by the motion planner is first re-timed using piece-wise linear interpolation before being sent over the data link to the control system. Trajectories are executed aboard the robot by feeding waypoints at 200Hz to the on-board controllers, which track the waypoints with PID controllers running at 1KHz. The operator is informed of the end of the trajectory execution by a monitoring system based on an ROS *Action Server* that returns success or failure if the robot has reached the end of the trajectory in the time constraints.

E. Human-Robot Interaction and Team Command

Due to the number of modules, the complexity of the system would generate too much cognitive load on a single operator. Thus, we have defined a multi-operator scheme that enables us to safely and accurately manage a task such as valve turning while being time efficient. In our multi-operator approach, each member is assigned a particular function (see Figure 6), and we make use of checklists and a "playbook", summarizing failure cases and possible strategic decisions to be made, to dictate the operators' tasks. An explicit chain of command is adopted to clarify responsibility and improve responsiveness in a failure case.

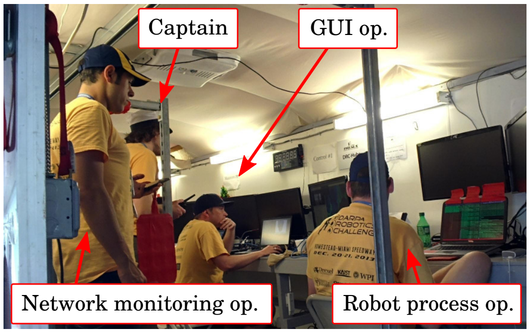

Fig. 6: Operator roles while teleoperating DRCHubo on the valve turning task at DRC trials.

The team roles were the following:

1) Captain: dispatches the different sub-tasks to the other operators and keeps track of the current strategy (e.g., the order in which to perform the environment scans, where to walk, and the manipulation tasks). This operator should have a good understanding of the system as well as precise knowledge of the fallback strategies stored in the "playbook".

2) GUI operator: prepares queries for the motion planner by localizing the objects (e.g., valves) using interactive markers, sends the trajectory to the robot after visualization (section III-C). This operator waits for approval by the Captain before executing planned motions.

3) Robot process operator: starts and monitors the software running aboard the robot (i.e., control and data aggregation systems). After launching three scripts in separate terminals, this operator monitors the terminals for errors and provides immediate diagnosis for the captain if any occur.

4) Network monitoring operator: Network communication with the robot can degrade dramatically, thus it is necessary to have an operator monitor the current network conditions using software that estimates the latency in real time. This operator can help the captain in his/her decisions to request a point cloud, which can overload the data link if made in a period with high latency.

As the operators' mental load is reduced using this co-operative control approach, adopting these roles enhances the robustness of the control process. Each operator is only responsible for a small number of tasks and the critical operations are monitored by at least two operators (i.e., the Captain and the operator responsible for the action). We also make use of a communication protocol where the name of the target operator is called prior to communicating to reduce the risk of mis-communication. For instance, in the valve turning task, when the captain asks if the robot computers have received the trajectory, stating the robot process operator's name avoids ambiguity in the request which could unnecessarily load the other operators.

Reducing the cognitive load of each operator enables acting in a safer and more effective manner. We believe team operation of a humanoid robot is especially efficient in scenarios where the robot must act under strong time constraints. Such operational modes are commonly found when operating large vehicles such as tanks or aircraft [28].

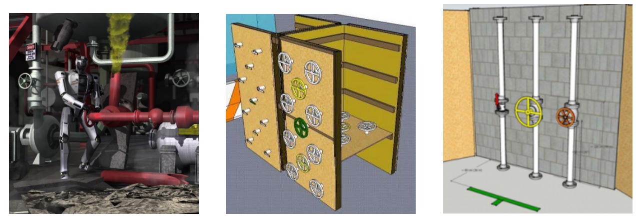

Fig. 7: Evolution of the task specs. provided by DARPA: **Left**: initial environment, **Center**: second specification with horizontal and vertical valve placements at different heights, **Right**: final task description with only vertically placed valves at a single height.

IV. Testing

We conducted tests over the course of two months, (eleven separate testing sessions) consisting of performing the valve turning task in an environment similar to Figure 7b. The tests were performed over the internet using a VPN connection, with the robot at Drexel in Philadelphia, USA, and the operator workstations at WPI in Worcester, Massachusetts, USA. The VPN connection incorporated delays and dropouts inherent to common long distance telecommunications. Each test was divided in two phases: setup time and run time. Depending on the time taken by setup, we ran between one and three trials per session. The average setup time, run time and results are reported in Figure 8.

We defined the score of our test to be similar to DRC conditions. However, since the task specification and score rubric published by DARPA changed multiple times our scoring rubric did not match the one used at the DRC trials. In the first phase of the tests we focused on turning a large circular valve three full turns. DRC rules were changed later to require only one turn on three valves in the setting presented in Figure 7c. Our rubric was:

- 1 point - Grasping the valve
- 2 point - One Turn
- 3 point - Three Turns

Figure 8 reports the evolution of the average setup and run times in minutes, as well as the average scores. During the first test sessions, our code was unstable and startup was largely manual, as illustrated by the high setup and run time and the low average points. However, after the first two test sessions the average number of points per run was 2.26, setup time was 21.6 minutes and run time 24.92 minutes. These results indicate that we were able to perform over one turn of the valve at each trial and shows the overall reliability of the framework. On test seven we could not score points due to hardware failure in the setup increasing our setuptime and preventing us from completing a valid run.

On the last run (i.e., test 11 not reported in Figure 8) we integrated the walking component from KAIST and adopted the final scoring rubric provided by DARPA. We aimed at turning three different valves in a setting similar to Figure 7c. On that test run the operational protocol as well as the software framework were the same as those used in the challenge. We scored four points by turning each valve a full turn within 30 minutes and not requiring intervention. At the DRC trials, we succeeded in turning the lever, however, the robot lost balance when we attempted to turn the second valve and we were unable to score the remaining points.

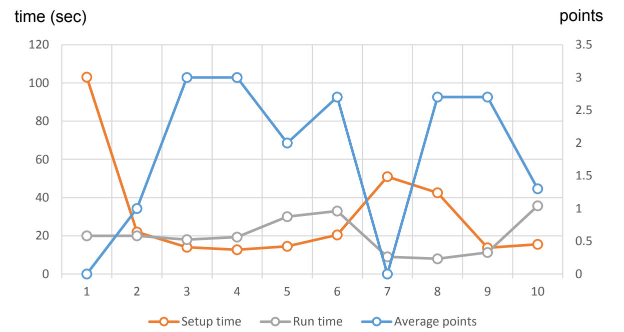

Fig. 8: Average setup time and run time in minutes (left) and average of points scored over 10 testing sessions (right). A test session comprise one to three tests.

V. Lessons Learned

The framework presented in Section III is the result of design and testing cycles in which significant trade-offs have been made to maximize its performance on the valve-turning task. The iterations of the DRC rules regarding bandwidth limitations and mock-up specifications acted as a moving target to our development. Initially, the vague requirements for operating the task, shown in Figure 7, encouraged us to implement a more autonomous approach. However, as the requirements became more precisely defined (e.g., no obstacles, only horizontal valves) and as autonomous techniques proved to be less reliable than direct input from the operators, we moved back from autonomy towards a more direct teleoperation approach. In this section, we discuss the reasons behind this shift for three core components of our system: motion planning, error detection, and object localization. We then discuss our cooperative traded control approach in relation to other teleoperation approaches.

A. Motion Planning

Motion planning is an essential component for robot autonomy. However, motion planning requirements depend largely on how much the workspace is structured (e.g., presence of multiple obstacles), physical and kinematic proprieties of the objects to be manipulated, and the capabilities of the robot itself.

1) Placement Planning: When the environment is unstructured, it is crucial to account for the robot's manipulation capabilities when selecting the placement location to perform a manipulation task. We pursued an autonomous solution to that problem based on reachability maps [29]. However, we found that a skilled human operator was able to determine a successful placement faster than the autonomous algorithm given a structured environment (i.e., with no obstacles, see Figure 7c). In this case, a simple estimate of distance to the valve from the point cloud data using an interactive marker by the operator was satisfactory.

2) Manipulation Planning: CBiRRT is able to account for obstacles as well as kinematic and balance constraints and thus can produce statically-stable motions. It was very effective throughout our testing. However, the algorithm does not account for the uncertainty on the manipulated object pose which is introduced by imperfect sensing. In the initial set of tests, the robot collided with the valve when performing the End operation, causing it to fall. Methods exist to account for uncertainty in sampling-based planning, however, we found that a simple solution based on adding way points in the reaching and extraction trajectories was sufficient. The way points are placed before grasping and after releasing the manipulated object, and the object's volume is augmented when planning motion to and from those way points. This procedure guarantees that the arms keep a minimal safety distance with the object to be manipulated. We found this solution effective enough to avoid collisions with the valves.

B. Error detection

Errors in trajectory execution (i.e., when the task is not performed as intended) can be identified by using the dynamic programming technique Dynamic Time Warping (DTW) to match executed trajectories against a library of known successful and unsuccessful trajectories [22]. DTW iteratively calculates the best alignment between elements of two or more time sequenced data [30] and produces a metric that quantitatively represents the similarity of those sequences to either the successful or unsuccessful class, which facilitates error detection during execution.

This error detection procedure was replaced by a simple monitoring of camera images by the operator. Indeed, this technique gives reasonable results on average (i.e., correct identification rate of 88%, the false positive rate was 10% and the false negative rate was 16% on the PR2), but even this high rate of performance can be easily surpassed by an experienced operator.

C. Object Localization

We avoided object detection because we did not have an appearance model of the object (i.e., color, the exact shape), however localization in a point cloud can be performed by the Iterative Closest Point (ICP) algorithm when given a good initial guess by the operator. ICP "snaps" the points on the surface of the object to the nearby points in the point cloud.

Despite our original plan to use ICP, the approach presented in Section III-C relies on the operator for both detection and localization of objects by aligning shapes to the point cloud data. In practice, operator localization of an object without using ICP was found to produce faster and more accurate results than using ICP. Indeed, ICP found the object pose quickly when given a good initial guess [22], however, due to the sparsity of the data the operator often needs to provide several initial guesses, making the process slower than specifying a precise pose directly.

D. Robot autonomy and teleoperation

The teleoperation method presented in Section III is an instance of *cooperative traded control* [4], which relies on an intermediate level of autonomy of the robot. This approach has two main advantages: traded control allows a more precise task specification, while cooperative control allows operators to maintain better situation awareness. We also experimented with a *direct control* approach as a fallback system for our framework, which requires less autonomy (i.e. substituting the motion planner for direct operator input from the GUI). As expected, this teleoperation mode was significantly slower than the control mode presented in Section III. From discussions with other teams competing in the DRC, it was indeed possible to use direct control for valve-turning, along with stored end-effector trajectories (e.g., a circle) to execute the turning motions (this is possible because the specifications for the valve locations were released not long before the competition). To perform the task with this strategy, the robot needs good balance control and arm strength to overcome the mismatch between the stored trajectory and the true valve pose and dimension – this is not the case of DRCHubo.

Another way to control the robot is by using *collaborative control* (similar to high-level supervision). Though this was our initial aim, as suggested by our experience, in such a system some critical components may be less reliable or slower than direct human operation. As a result, it is unclear if state-of-the-art techniques would enable the system to outperform a cooperative traded control approach.

VI. Conclusions

We have presented a framework for teleoperating humanoids robots to perform manipulation tasks with limited communication and demonstrated its effectiveness in a set of test runs. The framework consists of software modules running on-board the robot and on remote workstations which are connected through a custom data-link. Operators can specify the manipulation task using interactive markers to localize objects in point cloud data. Statically-stable trajectories are then planned which are collision-free and respect end-effector pose constraints. We have also defined a protocol to teleoperate the robot with a team of operators. We have analyzed our approach and discuss lessons learned designing our system which resulted in several steps back from autonomy. Our experience suggests that the key to designing an efficient teleoperation system is to identify where a well-trained operator can surpass the performance of software components to get the best of both autonomy and human expertise.

References

[1] C. Phillips-Grafflin et al., "Unreliable Network Communication Toolkit." http://github.com/WPI-ARC/teleop_toolkit, 2013.

[2] J. Casper and R. R. Murphy, "Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 2003.

[3] H. A. Yanco, J. L. Drury, and J. Scholtz, "Beyond usability evaluation: Analysis of human-robot interaction at a major robotics competition," *Human–Computer Interaction*, vol. 19, no. 1-2, pp. 117–149, 2004.

[4] M. A. Goodrich, J. W. Crandall, and E. Barakova, "Teleoperation and beyond for assistive humanoid robots," *Reviews of Human Factors and Ergonomics*, vol. 9, no. 1, pp. 175–226, 2013.

[5] A. Hornung, M. Phillips, E. Jones, M. Bennewitz, M. Likhachev, and S. Chitta, "Navigation in three-dimensional cluttered environments for mobile manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[6] F. Stulp, A. Fedrizzi, L. Mösenlechner, and M. Beetz, "Learning and reasoning with action-related places for robust mobile manipulation," *Journal of Artificial Intelligence Research*, vol. 43, no. 1, pp. 1–42, 2012.

[7] A. Miller and P. Allen, "Graspit! a versatile simulator for robotic grasping," *Robotics Automation Magazine*, vol. 11, no. 4, pp. 110–122, 2004.

[8] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *International Journal of Robotics Research (IJRR)*, vol. 30, no. 12, p. 1435 1460, 2011.

[9] W. Ferrell and T. Sheridan, "Supervisory control of remote manipulation," *IEEE Spectrum*, vol. 4, no. 10, pp. 81–88, 1967.

[10] T. B. Sheridan, *Telerobotics, automation and human supervisory control*. The MIT press, 1992.

[11] D. Sakamoto, T. Kanda, T. Ono, H. Ishiguro, and N. Hagita, "Android as a telecommunication medium with a human-like presence," in *International Conference on Human-Robot Interaction (HRI)*, 2007.

[12] A. A. D. Medeiros, "A survey of control architectures for autonomous mobile robots," *Journal of the Brazilian Computer Society*, vol. 4, no. 3, 1998.

[13] R. C. Arkin and T. Balch, "Aura: Principles and practice in review," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 175–189, 1997.

[14] C. Oestges, B. Montenegro-Villacieros, and D. Vanhoenacker-Janvier, "Modeling propagation into collapsed buildings for radio-localization-based rescue search missions," in *IEEE Antennas and Propagation Society International Symposium*, 2009.

[15] R. Burridge and K. Hambuchen, "Using prediction to enhance remote robot supervision across time delay," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[16] M. Lum, J. Rosen, H. King, D. Friedman, T. Lendvay, A. Wright, M. Sinanan, and B. Hannaford, "Teleoperation in surgical robotics–network latency effects on surgical performance," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2009.

[17] P. Pirjanian, T. Huntsberger, A. Trebi-Ollennu, H. Aghazarian, H. Das, S. Joshi, and P. Schenker, "Campout: A control architecture for multi-robot planetary outposts," in *Proc. SPIE Symposium on Sensor Fusion and Decentralized Control in Robotic Systems III*, 2000.

[18] J. Biesiadecki, C. Leger, and M. Maimone, "Tradeoffs between directed and autonomous driving on the mars exploration rovers," in *Robotics Research*, vol. 26, pp. 91–104, Springer, 2007.

[19] J. Jackson, "Microsoft robotics studio: A technical introduction," *IEEE Robotics & Automation Magazine*, vol. 14, no. 4, pp. 82–87, 2007.

[20] M. Sarabia, R. Ros, and Y. Demiris, "Towards an open-source social middleware for humanoid robots," in *International Conference on Humanoid Robots (Humanoids)*, 2011.

[21] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, 2009.

[22] N. Alunni et al., "Toward a user-guided manipulation framework for high-dof robots with limited communication," in *IEEE TePRA*, 2013.

[23] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," *Robotics Institute, Pittsburgh, PA, Tech. Rep.*, 2008.

[24] N. Dantam and M. Stilman, "Robust and efficient communication for real-time multi-process robot software," in *International Conference on Humanoid Robots (Humanoids)*, 2012.

[25] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), 2011.

[26] D. Gossow, A. Leeper, D. Hershberger, and M. Ciocarlie, "Interactive markers: 3-d user interfaces for ros applications," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 14–15, 2011.

[27] B. Siciliano and O. Khatib, *Handbook of Robotics*. Springer, 2008.

[28] G. W. Cooke, "U.S. Army Tank Doctrine." http://www.inetres.com/gp/military/cv/US_tank_doctrine.html, 2004.

[29] F. Zacharias, W. Sepp, C. Borst, and G. Hirzinger, "Using a Model of the Reachable Workspace to Position Mobile Manipulators for 3-d Trajectories," in *International Conference on Humanoid Robots*, 2009.

[30] P. Senin, "Dynamic time warping algorithm review," *Technical report, ICS Department, Univ. of Hawaii at Manoa, Honolulu, USA*, 2008.