Collaborative Autonomy Between High-level Behaviors and Human Operators for Remote Manipulation Tasks using Different Humanoid Robots

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

Spyros Maniatopoulos, Hadas Kress-Gazit
Verifiable Robotics Research Group, School of Mechanical and Aerospace Engineering, Cornell University
Ithaca, NY 14853, USA
{sm2296,hadaskg}@cornell.edu

Philipp Schillinger[*]
Robert Bosch GmbH, Corporate Research, Department for Cognitive Systems
70442 Stuttgart, Germany
philipp.schillinger@de.bosch.com

David C. Conner[†]
Capable Humanitarian Robotics & Intelligent Systems Lab, Department of Physics,
Computer Science and Engineering, Christopher Newport University,
Newport News, VA 23606, USA
david.conner@cnu.edu

Abstract

Team ViGIR and Team Hector participated in the DARPA Robotics Challenge (DRC) Finals, held June 2015 in Pomona, California, along with 21 other teams from around the world. Both teams competed using the same high-level software, in conjunction with independently developed low-level software specific to their humanoid robots. Based on previous work on operator-centric manipulation control at the level of affordances, we developed an approach that allows one or more human operators to share control authority with a high-level behavior controller. This *collaborative autonomy* decreases the completion time of manipulation tasks, increases the reliability of the human-robot team, and allows the operators to adjust the robotic system's autonomy on-the-fly.

This article discusses the technical challenges we faced and overcame during our efforts to allow the human operators to interact with the robotic system at a higher level of abstraction and share control authority with it. We introduce and evaluate the proposed approach in the context of our two teams' participation in the DRC Finals. We also present additional, systematic experiments conducted in the lab afterwards. Finally, we present a discussion about the lessons learned while transitioning between operator-centered manipulation control and behavior-centered manipulation control during competition.

[*]Presented work performed at Technische Universität Darmstadt [†]Presented work performed at TORC Robotics

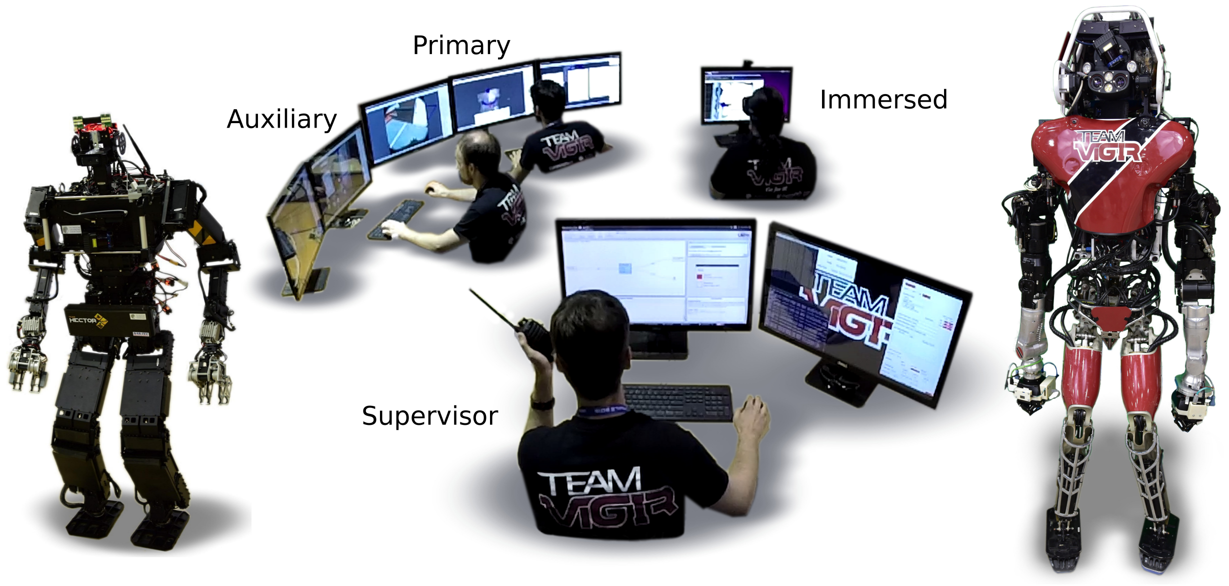


Figure 1: THORMANG "Johnny" developed by Robotis and used by Team Hector (left), Atlas "Florian" developed by Boston Dynamics Inc. (BDI) (right) and the Operator Control Station (middle) used by Team ViGIR. Both humanoid platforms use the same high-level software to perform remote manipulation tasks with human supervision. Operator roles are described in detail on Section 5.

1 Introduction

Executing robotic manipulation tasks in remote, and potentially degraded environments, presents challenging problems as shown in 2011 during robot operations at the post-disaster Fukushima Daiichi Nuclear Plant in Japan (Nagatani et al., 2013). The Defense Advanced Research Projects Agency (DARPA) motivated the development of robotic systems to tackle these problems with the DARPA Robotics Challenge[1] (DRC). The three main DRC events — Virtual Robotics Challenge in June 2013, DRC Trials in December 2013, and DRC Finals in June 2015 — focused in simulating disaster scenarios that required robots to perform mobility and manipulation tasks under severe communication constraints. The teams' development schedule was highly constrained as well, and the difficulty of the tasks was increased with each event.

The VRC was designed to be a pure software competition using three different simulated disaster environments. The mobility and manipulation challenges included traversing rough terrain, driving a vehicle, and manipulating a fire hose. For the DRC Trials, eight different mobility and manipulation tasks were to be performed by real robots in thirty minutes each. The DRC Finals was the last of these events and in contrast to the VRC and the DRC Trials, all tasks had to be performed in a single run. In addition, the robots had to be untethered. Each participating team's robot had sixty minutes to perform the eight tasks — driving a vehicle, egressing the vehicle, opening a door, turning a valve, breaking a wall using a tool, performing one of three surprise manipulation tasks, traversing rubble or uneven terrain, and climbing a ladder.

Team ViGIR[2] focused on developing high-level robot-agnostic software to participate in this challenge. Team Hector[3] implemented the same high-level software and contributed to its development. Both teams used highly advanced humanoid robots to compete in the DRC finals as shown in Figure 1. Each team developed their own low level control software for the specifics of each humanoid robot, but both teams used the same high-level software including mobility, perception, manipulation, and communication as well as the same graphical user interface. This high-level software was also used by Team Valor[4] at the DRC Finals.

[1] `http://www.theroboticschallenge.com` [2] `http://www.teamvigir.org` [3] `http://www.teamhector.de`
[4] `http://www.me.vt.edu/trec/team-valor`

The DRC events have resulted in significant advances in the area of robot control from a remote location using a human operator in the loop. However, the approaches developed for the DRC still have to face multiple problems requiring different areas of expertise. These areas spread from low-level system layers such as actuator control and data compression for communication to high-level system layers such as user interfaces and autonomous behaviors.

This paper focuses on work towards a shared autonomy manipulation control approach. By enabling abstraction of lower layers such as motion planning, higher level layers can be executed either by autonomous behaviors, human operators, or a collaboration of the two. The approach described in this paper thus focuses on three layers: Motion Planning, Object Template Manipulation, and High-level Behavior Control.

Humanoid robots are complex to control given the high number of degrees of freedom(Goodrich et al., 2013). For this reason, motion planning algorithms are required to generate joint-space trajectories based on Cartesian-space motions. To approach this problem, we based our motion planning layer on the *MoveIt!* (Chitta et al., 2012) motion planning library. *MoveIt!* provides open-source software for robot control, kinematics, motion planning, and manipulation among others. For the purposes of the DRC, we developed additional manipulation planning packages to provide functionality in the range from pure joint control to Cartesian trajectory control of the end-effector. Still, motion planning in a robot-centered basis makes it challenging to perform object manipulation motions. For this reason, we developed an object-centered manipulation control approach based on the concept of *affordances* (Gibson, 1977) on top of our manipulation planning layer. This manipulation control approach uses an implementation of *Object Templates* (Romay et al., 2014) as entities to represent real objects that can provide the remote robot with the necessary information to generate motion planning trajectories and manipulate such objects. Object templates are virtual representations of real objects in the form of 3D meshes that a human operator can manipulate using a graphical user interface that is part of our Operator Control Station (OCS), where the robot model and the sensor data from the remote environment is simultaneously displayed. Using object templates, an operator can aid the remote robot performing perception tasks by identifying objects of interest in sensor data. Once an operator identifies an object of interest, an object template can be inserted in the virtual environment and be overlaid over sensor data corresponding to the object identified. After object identification and overlaying of object templates have been performed, the robot can use the template pose to perform locomotion to approach the object, and grasping and manipulation based on the semantic information provided by the object template. The manipulation task can be supervised by a human operator commanding subtasks such as walking to the object, grasping it using predefined end-effector poses, and manipulating the object by executing the affordances provided by the object template.

Finally, our high-level behavior control layer enables task-level abstraction and autonomy. Our approach is based on modeling *high-level robot behaviors* as hierarchical finite-state machines, which we augment with data flow, adjustable autonomy, and elements of operator interaction (Schillinger, 2015). This high-level control layer has been implemented as the Flexible Behavior Engine (FlexBE), which comprises a Python framework for developing state primitives, an executive, and a graphical user interface (GUI) for rapidly composing state machines and supervising their execution. The development framework and the executive are an extension of the SMACH task execution framework (Bohren and Cousins, 2010). In terms of the three-layered approach, the key takeaway is that high-level behaviors can query object templates for information, such as grasp configurations and available affordances, in order to autonomously carry out manipulation tasks. They do so programmatically, as opposed to the operators' use of the OCS interface. Our overall approach is based on the idea that a remote manipulation task can be performed more efficiently if the system is flexible enough to allow execution of robot actions commanded from any of the three layers.

In this paper, we present results of remote humanoid robots using high-level behaviors to perform manipulation tasks based on the information provided by object templates. Developing this approach led to the following contributions:

- A remote manipulation control approach that allows abstraction to high-level system layers to command remote robots to interact and manipulate objects in the environment on an affordance level (Section 3).

- A high-level behavior control approach, based on finite-state machines, which accounts for data flow, operator interaction, and adjustable autonomy of execution (Section 4).
- An overarching principle, dubbed *collaborative autonomy*, which brings together our remote manipulation and high-level control approaches. Collaborative autonomy allows a human-robot team to interact and seamlessly share control authority during the execution of high-level tasks (Section 5).
- All of our software has been open sourced[5] and can be tested in simulation[6]. Some highlights include:
  - The Flexible Behavior Engine (FlexBE), which substantially extends the SMACH High-level Executive with several ways of operator interaction concepts like adjustable autonomy. In addition, it provides a graphical user interface for composing and executing these behaviors.
  - A manipulation backend for humanoid robots, which includes motion planning packages and the object template approach to perform manipulation tasks in an object-centered basis.

Team ViGIR entered the DRC as a "Track B" team and participated in all three events of the DRC. After Team ViGIR's success in the VRC (Kohlbrecher et al., 2013) and the DRC Trials (Kohlbrecher et al., 2015), additional research groups joined the team. For the DRC Finals, Team ViGIR comprised TORC Robotics[7], the Simulation, Systems Optimization and Robotics Group at Technische Universität Darmstadt[8], the 3D Interaction Group at Virginia Tech[9], the Robotics and Human Control Systems Laboratory at Oregon State University[10], the Verifiable Robotics Research Group at Cornell University[11], and the Institute of Automatic Control at Leibniz Universität Hannover[12].

Team Hector participated at the DRC Finals as a "Track D" team. It is composed of additional researchers and students from the Simulation, Systems Optimization and Robotics Group at Technische Universität Darmstadt. Team Hector qualified for the DRC Finals four months before the competition with a basic software approach. Given the short time to prepare for the competition, Team Hector focused on integrating Team ViGIR's existing high-level software with their humanoid robot "Johnny" and thereby prove the versatility of their software approach. Both teams successfully implemented, extended, and maintained the common software parts which contributed to faster development and quality maintenance.

The rest of the paper is organized as follows: In Section 2, we present an overview of related work, with a focus on remote manipulation of robots using human supervision and high-level control. In Section 3, we describe the core of the manipulation control approach including details on motion planning, object templates, and affordance-level interaction. In Section 4, we introduce our approach to high-level behavior control and its software implementation, FlexBE. In Section 5, we expand on the concept of collaborative autonomy and discuss how high-level behaviors and human operators interact in order to control a remote robot carrying out manipulation tasks. In Section 6, we present an evaluation of our results based on (i) the field performance of both teams during the DRC Finals and (ii) additional systematic experiments. In Section 7, we summarize the lessons that we learned developing our approach and competing in the DRC Finals. We draw conclusions in Section 8.

2 Related Work

Remote, partially-degraded, and uncontrolled environments present major challenges and a multitude of approaches have emerged to tackle these. These approaches are spread across a number of research areas; for the purposes of this paper, related work focuses on areas such as remote manipulation, high-level control, and autonomy.

[5] https://github.com/team-vigir
[6] https://github.com/team-vigir/vigir_install/wiki
[7] http://www.torcrobotics.com
[8] http://www.sim.informatik.tu-darmstadt.de
[9] http://research.cs.vt.edu/3di
[10] http://mime.oregonstate.edu/research/rhcs
[11] http://verifiablerobotics.com
[12] http://www.uni-hannover.de/en

2.1 Interaction Methods for Remote Manipulation Control

Of particular interest for interaction methods with remote robots is the ability to transfer human operator intent to the remote robot, converting it into actions in order to perform tasks in the environment. Considering a human operator in the loop for remote manipulation tasks has been proven as an effective approach to deal with challenges that uncontrolled and potentially degraded environments present, but it also presents new challenges.In order for a robot to perform manipulation tasks, it needs to acquire information about the objects to be manipulated, such as physical and semantic information. Having a human operator performing perception tasks, for example, can often provide more reliable information about the environment than can be obtained autonomously. This approach was widely used by teams in the DRC.

Team IHMC placed second in the DRC Finals. Their overall system design is based on the concept of Coactive Design (Johnson et al., 2011) which focuses on designing a system considering interdependency between participants in joint activity. Derived from this concept, they present an approach for an operator to control the behavior of a humanoid robot called *interactable objects* presented in (Koolen et al., 2013) and (Johnson et al., 2015). These interactable objects allow an operator to transfer his/her intent to the robot, for example, by selecting different grasp poses for the end-effectors, selecting different manipulation stance poses that allow the robot to reach the object, and also gives the ability to transfer information about how to perform footstep plans for locomotion with respect to the object of interest. To transfer information about how to manipulate objects in the environment, end-effectors can be linked to the interactable objects. This way, when the operator modifies the interactable object pose, the end-effector follows the pose and a trajectory is generated which can then be sent to the robot to execute it.

The Affordance Template ROS package developed by Team NASA-JSC (Hart et al., 2015) provides a human operator with a high level of adjustment and interactivity of the geometry information from the affordance templates used to represent real objects. This approach provides an interaction method where the human operator can adjust the scale of the templates. This adjustment also acounts for predefined waypoints defined in the frame of reference of the template. These waypoints are defined in an Affordance Template Description Format XML file and are used to generate potential poses for the robot's end-effectors. These templates are designed to provide robot-agnostic grasp poses and manipulation trajectories. Their interface provides the operator with the ability to command the robot to move through these trajectories, allowing execution in any direction.

In an approach presented by Team MIT (Fallon et al., 2015), CAD models of objects are used as environmental features that allow an operator to command robot actions based on the action possibilities that the real objects have. These templates provide information such as manipulation stances where the robot is able to reach the object and predefined potential grasp locations. The information contained in the templates is defined in an XML file which they call Object Template Description Format. In this file, objects are defined as a series of links and joints. In this approach, to generate arm trajectories, the operator is required to manipulate the object template and change its pose. The robot end effector follows the pose of the template and this information is used to define a desired end-effector trajectory. They present examples of performing plan motions with respect to a point of interest in the drill, such as the "drill bit" and selecting different points in kinematic chain links for motion planning. To our understanding, the approach presents no option to the human operator to select different points of interest from the objects grasped on the fly.

In all of these approaches, the human is required to interact and change the pose of the 3D objects to generate trajectories for the end-effector. In contrast, the approach presented here does not require that the human changes the pose of the object template, rather, it allows a human operator or a high-level behavior to request the affordance of the object which will then provide the necessary information to a motion planning backend that will autonomously generate the require motions.

Human supervision of remote manipulation tasks is not limited to rescue applications. For example, an approach to provide an astronaut with an interaction method to command a robot for space application has been presented by the German Aerospace Agency in (Birkenkampf et al., 2014). In this approach, a tablet-application for shared autonomy between a semi-autonomous robot and a remote operator is presented. This approach is based on previous work designed to provide a fully autonomous robot with symbolic and geometric information of the objects required to perform a manipulation task (Leidner et al., 2012). While this approach provides an interesting concept on how to ground symbolic information of a task into robot actions, it does not provide an interface to define these motions with respect to a point of interest in the object grasped. Another interesting approach is (Berenson et al., 2011), where Task Space Regions (TSRs) are proposed as a general representation for end-effector pose constraints. In their approach, articulated objects can also be manipulated by chaining multiple TSRs. Neither of these approaches consider the use of an abstract representation of the real object in a virtual environment; they use action information of the task that is then transformed into geometric information to generate the appropriate joint motions.

The approach presented in this paper for object manipulation allows a human operator or a high-level behavior to perform manipulation tasks using frames of reference in the object templates to define the affordances of the object, which provide information about the motions that are required to manipulate the object for a specific task. This means that no defined waypoints are used for end-effector targets; instead, the motion planning backend generates end-effector trajectories based on the affordances of the object. This manipulation interaction approach also allows dynamic selection of different known points of interest on the objects grasped or *object usabilities* that need to be considered for planning motions to achieve a specific manipulation task.

2.2 High-level Control and Autonomy

We compare our approach to high-level control with that of other teams that participated in the DRC. In addition, we discuss how our concept of collaborative autonomy relates to existing views of autonomy.

For their Valkyrie humanoid robot (Radford et al., 2015), Team NASA-JSC used the Robot Task Commander (RTC), a framework for defining, developing, and deploying robot application software for use in different runtime contexts (Hart et al., 2014). RTC is similar to our Flexible Behavior Engine (FlexBE) in that it is based on hierarchical finite-state machines (HFSM) for modeling control and data flow. In addition, both RTC and FlexBE have a graphical user interface (GUI) for facilitating the design of state machines. However, when running a high-level behavior with RTC, its GUI primarily acts as a read-only monitor for the current status of execution. FlexBE's main advantage in this regard is its focus on human-robot collaboration. During operation, FlexBE enables the operator to seamlessly switch between different levels of autonomy; from low, expressed by fine-grained decision approval as well as operator-triggered state transitions, to high, where the system only requests help from the operator if essential data is missing or unexpected errors occur.

Team IHMC's approach to operator-robot supervision and interaction was also based on the principles of Coactive Design. The operator executed sequences of autonomous scripted actions (Koolen et al., 2013), supervised their execution, and intervened if necessary. Their approach relies on expert developers programming these scripts. To our understanding, the scripts are executed in sequence, which is not as expressive as HFSMs. However, Team IHMC's approach proved highly effective in the DRC Finals.

Team MIT likewise used scripted "task sequences" to define their high-level control (Fallon and Marion, 2016). They did not *explicitly* employ task-level autonomy, and instead used these predefined scripts to choreograph the actions (DRC Teams, 2015). Instead, the operator assisted with perception and executed scripts for each task, which in turn passed objectives and constraints to the online planning system. As mentioned before, we believe that FlexBE HFSMs are more versatile than scripts. Instead, we conjecture that Team MIT achieved a very high degree of autonomy thanks to the tight integration between operator interaction, affordance-based perception and manipulation, and optimization-based whole-body planning (Fallon et al., 2015).

Team WPI-CMU employed autonomous execution with operator intervention when required. An example of their HFSM-based approach can be found in (Babu et al., 2015). To the best of our knowledge, the state machines and user interfaces they developed were custom-fit to the DRC Finals tasks. By contrast, FlexBE is a framework for composing, executing, and supervising HFSMs for any high-level task and any ROS-based robotic system. For example, it was used by Team ViGIR and Team Hector in the DRC Finals, and also by Team Argonauts to control their tracked mobile robot in the ARGOS Challenge.[13]

Team RoboSimian represented robot behaviors as asynchronous HFSMs (Hebert et al., 2015). In the DRC Trials, their behaviors were not at the task-level, but rather at the level of motion planning. They did compose their behaviors, but only in the form of operator-defined sequences. However, in their paper they mention that their system supports more expressive composition of behaviors.

In terms of the concept of autonomy itself, (Huang et al., 2007) have defined a generic model for autonomy levels for unmanned systems. Our approach would span what they define as actuator (teleoperation), subsystem (affordance-based manipulation), and system autonomy (high-level behavior control). A qualitative meta-analysis of autonomy in the DRC Trials has been conducted (Murphy, 2015). In terms of the analysis' nomenclature, Team ViGIR moved from "execution approval", where actions are delegated to the robot in small chunks, to "task rehearsal", where entire sequences of actions are considered. Our approach enables the switching between these two forms of delegation on-the-fly via an adjustable autonomy mechanism.

Our proposed *collaborative autonomy* is related to *collaborative control* as introduced by (Fong et al., 1999), (Fong and Nourbakhsh, 2004). Specifically, we view collaborative control as a broader definition, which encompasses two concepts of interest: collaborative perception and collaborative autonomy. Briefly, in collaborative perception the human operators assist the robot with perception tasks, e.g. detection of objects of interest, whether in semi-autonomous or fully autonomous operation. In collaborative autonomy, they assist with cognition, decision making, and even actions, such as object manipulation. We have elaborated on collaborative perception in our previous work (Kohlbrecher et al., 2015). In this paper, we will be focusing on collaborative autonomy,[14] which (Klien et al., 2004) have identified as one of ten challenges for making automation a "team player" in joint human-agent activity. Specifically, in this work, one or more human operators are collaborating with a high-level behavior to control a remote humanoid robot.

Collaborative autonomy is also related to the paradigm of supervised autonomy (Cheng and Zelinsky, 2001). However, we also allow for the online adjustment of the level of autonomy, along the lines of (Crandall and Goodrich, 2001). In addition, (Desai and Yanco, 2005) proposed sliding scale autonomy as an alternative to discrete autonomy levels. By contrast, our approach has the behavior control designer fine-tune the relative autonomy of the various steps of a high-level task *a priori* and then allows the operator to adjust the active behavior's autonomy on-the-fly during execution. To conclude, collaborative autonomy allows a human-robot team to carry out a task together; with the high-level behavior autonomously requesting operator input when required and the operator intervening, if deemed necessary, and then handing control authority back to the behavior.

3 Manipulation Planning and Interaction

For effective and reliable manipulation, the planning system has to provide planning capabilities that can be leveraged by different system components. We describe both the motion planning backend and the object template based frontend in the following.

3.1 Motion Planning

The motion planning and control system has to provide sliding autonomy capability, allowing for control options from full teleoperation to full autonomy, with object template based task-level control by the operator

[13] `http://argos-challenge.com/en` [14] The term has also been used in a different context, that of a human operating a team of unmanned vehicles.

in-between. This increases resilience in complex disaster response tasks, as different control paradigms can be switched seamlessly should the need arise.

For versatile manipulation planning, the planning backend supports joint space commands, cartesian target poses, and cartesian waypoints. The default planning backend we used for the DRC is based on the MoveIt! planning framework which is integrated into ROS. It allows for reliable planning of upper body motions utilizing full environment collision avoidance based on a real-time updated *octomap* (Hornung et al., 2013) model of the environment. This environment model is generated in real-time based on LIDAR data. All types of motion requests can be requested via standardized ROS action requests. Using an action interface allows the caller to monitor the planning and execution process. The caller thus provides a motion goal and then listens to feedback about progress towards the goal and retrieves the action result once succeeded or aborted.

We provide a ROS-action-based interface, therefore the planning backend implementation is decoupled from components using it as shown in Figure 2. The backend can thus be transparently modified or exchanged for another one.

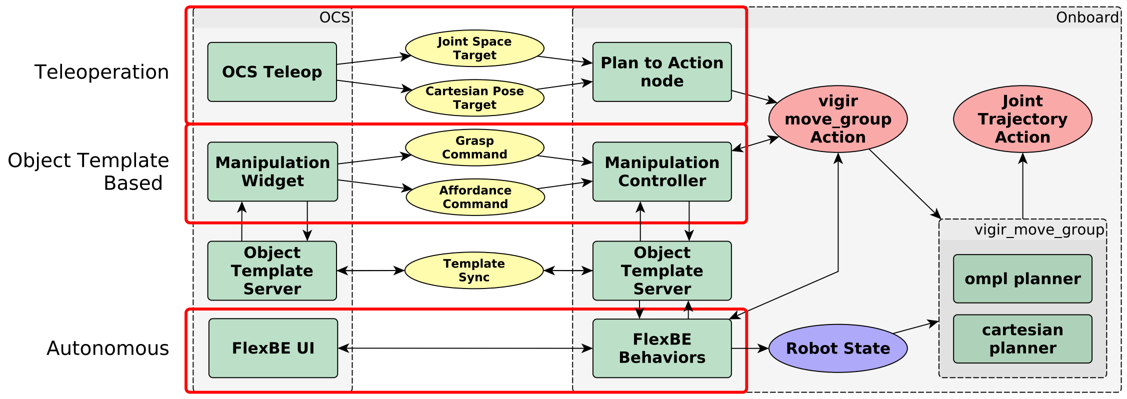


Figure 2: Overview of the manipulation planning and control system. There are three different options for performing manipulation tasks for the operator, from teleoperation to full autonomy. Even in full autonomy, the operator can influence state transitions at any time.

As an alternative backend, we investigated the Drake planning and control library (Tedrake, 2014), which allows for planning whole body motions. We did not used it for the experiments in this work, as it does not easily allow for using environment collision avoidance based on real-time sensor data. Locomotion and stability control for manipulation with the Atlas robot were performed using the provided API from BDI and Robotis API for the THORMANG.

3.1.1 Teleoperation

In teleoperation mode, the operator can command robot motion directly from the OCS. The most basic mode is single joint commands, which is only employed in case a failure in the system makes low level control necessary. The operator can also use joint target and cartesian pose target commands. In case of their use, the "ghost robot" is employed to visualize the goal configuration. Commands are sent as a compressed representation from OCS to the onboard side (robot side), where the actual planning based on the full robot and environment state is performed. A *plan to action* ROS node on the onboard side is used to translate the compressed motion plan requests coming from the OCS into the *vigir_move_group_action*[15] ROS action request.

[15] `https://github.com/team-vigir/vigir_manipulation_planning`

3.1.2 Object Template Centric Manipulation

In the object template manipulation mode described in further detail below, motion plan requests are generated based on object templates and affordances they offer. The manipulation controller component generates motion plan requests based on affordance commands and generates the appropriate *vigir_move_group_action* action request.

3.1.3 Autonomous Manipulation

In case of autonomous or semi-autonomous behavior guided manipulation, motion plan requests can be sent by any state machine as a *vigir_move_group_action* action request. Based on the action response returned, behaviors can then perform transitions autonomously.

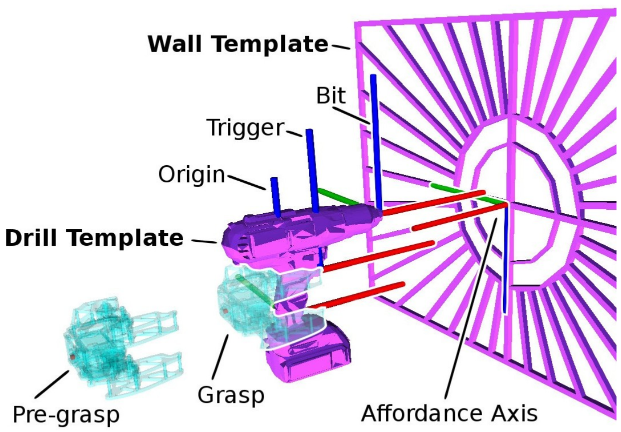
3.2 Manipulation with Object Templates

In remote robotic manipulation tasks with human supervision, the operator needs to communicate their intent to the robot so that this intent can be converted into action to perform manipulation tasks. This communication needs to be done in an efficient way to allow the robot to rapidly interact with the objects in the environment. In contrast to teleoperation and to the approaches discussed in Section 2.1, our interaction method called "Object Template Manipulation" allows the operator to rapidly communicate information of objects found in the environment and provides the remote robot with information about how these objects can be manipulated. The concept of object templates refers to the idea of abstracting the information of real objects in the environment into virtual representations as a means of aiding a remote robot with perception and understanding of manipulation information from these objects. These object templates are visualized in a virtual environment as 3D models that can be manipulated by a human operator. Object template information includes locomotion target poses to approach the object, predefined end-effector positions to grasp the object, and motion constraints to manipulate the object.

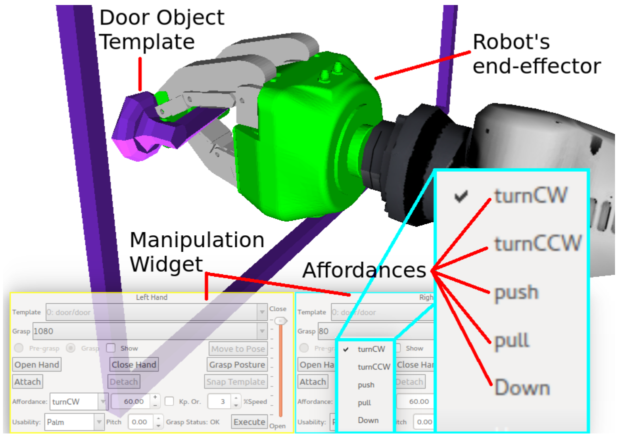
3.2.1 Object Template Definition

Our implementation of Object Templates (OT) was introduced in (Romay et al., 2014) as a means of interaction between a human operator and a remote semi-autonomous robot. Object templates are used to provide the remote robot with information about the objects of interest that the operator identifies on the sensor data acquired from the real world. The initial version of the OT approach used during the VRC considered only the 3D mesh of the object and potential grasp pose information (Kohlbrecher et al., 2013). After the development phase between the VRC and the DRC Trials, we evolved our OT approach to provide information about the functionality of the object. This brought the concept of *affordances* (Gibson, 1977) (Şahin et al., 2007) to the OT approach in order to be able to define information about how the object should be manipulated (Romay et al., 2014). Additionally we developed the concept of *usability*. Usabilities allow the operator to select points of interest in a grasped object so that this point can be used while executing affordance manipulation motions. These two concepts, affordances and usabilities, provide a unique capability compared to state-of-the-art approaches. From one side, affordances allow planning motions considering the potential actions that can be performed with objects. From the other side, usabilities define the frame of reference located on an object grasped by the end-effector which can be used, for example, to execute an affordance of another object (this attachment is assumed to as a rigid body transformation between the origin of the object template and the robot's end-effector).

Object template information is stored in a data structure that we call the Object Template Library (OTL). The OTL is divided into three blocks of information: the object library, the grasp pose library, and the stand pose library. The object template library contains physical and semantic information of the objects such as mass, CoM, inertia tensor, geometry mesh, affordances, and usabilities. The grasp pose library contains the list of potential end-effector poses with information about the grasp posture and pre-grasp poses for each grasp. The stand pose library contains the list of potential robot stand poses that allow the robot to reach the objects. The grasp pose library and the stand pose library have a relationship of many to one with the object library. Each object in the object library has a unique type that is used to relate one or many grasps to one object template as well as for stand poses. This distribution of information allows different robots and different end-effectors to be considered by this approach. For example, Team ViGIR created a stand library and a grasp library for the end-effectors mounted on Atlas and Team Hector created a different stand library and grasp library for the end-effectors mounted on THORMANG while both teams used the same object template library. This also considers situations, for instance, when a robot has different end-effectors in each arm. In this case, different grasp libraries can be used by the same robot. A graphical example of the components of an object template can be seen in Figure 3.



(a) Components of object templates. Drill Template and Wall Template. Pre-grasp and grasp shown as a translucent "ghost" hand. Affordance frame for the Wall Template and Usability frames for the Drill Template (origin, trigger, and bit).



(b) The object template of a door being grasped by the robot's end-effector. The Manipulation Widget is shown for both hands. The affordances combo box is zoomed in to show the available motions of the door, e.g., turn Clockwise (CW) or turn counter-clockwise (CCW) as well as pushing and pulling, among others.

Figure 3: Main information of object templates. Object template components (left) and manipulation widget (right).

These components are defined as follows:

Definition: Stand pose. *A stand pose* $P_S \in \mathbb{R}^3 \times \mathbb{SO}(3)$ *is a position relative to the object template frame of reference that indicates a potential location for the robot's pelvis that allows the robot to reach the object.*

Definition: Grasp. *A pose* $P_G \in \mathbb{R}^3 \times \mathbb{SO}(3)$ *provides a potential target for the robot's end-effector to grasp the object.*

These grasps are calculated offline for a particular type of end-effector used and are specifically designed for the task objective. An object template can have multiple grasps and they are also differentiated from left and right end-effectors.

Definition: Pre-grasp. *A pre-grasp is a pose* $P_{PG} \in \mathbb{R}^3 \times \mathbb{SO}(3)$ *that is calculated using an approaching vector* $V \in \mathbb{R}^3$ *and a distance* $D \in \mathbb{R}$ *from the grasp* P_G.

The pre-grasp provides a potential safe pose for the robot's end-effector before the actual grasp P_G of the object. Transitioning between pre-grasp and a grasp is performed in a straight line defined by the vector V.

Definition: Grasp posture. *The grasp posture is a tuple of joint values for the fingers of the robot's end-effector that are required for the designed grasp* P_G.

Definition: Affordance. *An affordance in the context of this approach is defined as a motion constraint that the end-effector of the robot needs to follow in order to manipulate an object. This motion is defined by an axis in a frame of reference* $A \in \mathbb{R}^3 \times \mathbb{SO}(3)$ *and it can be either a linear motion, a circular motion, or a combination of both motions if a screw pitch value* $P \in \mathbb{R}$ *is considered.*

We designed these affordances to be grasp-agnostic, so only the frame of reference of the end-effector is considered for planning motions. This allows the robot to have any type of end-effector and still be able to generate the required motions to perform the task.

Definition: Usability. *A usability is a point of interest defined with respect to the origin of an object template. This frame of reference* $U \in \mathbb{R}^3 \times \mathbb{SO}(3)$ *is used to augment the robot's end-effector to consider this point of interest during motion planning.*

As an example, consider Figure 3a. Once the drill is grasped, the usability of the drill bit defines the end-effector for planning relative to the cutting affordance of the wall template.

3.2.2 Object Templates Implementation

Object templates were implemented in an Object Template Server (OTS). The OTS is responsible for loading and providing object template information to any client that requested it. For example, the Main View widget will request 3D geometry mesh information from the object template library to display, as well as finger joint configuration while displaying potential end-effector poses to grasp such object. Other clients such as the Manipulation Widget (see Figure 3b) could request grasps, affordances, and usabilities from the OTS. Additionally, the use of the OTS by autonomous behaviors is decribed in Section 5.

Given the communication constraints for the DRC, the OTS was required to provide information for both the OCS side and the onboard side. On the OCS side, the OTS provides information to all the widgets that use object templates. It also manages the instantiated object templates that the operator has inserted in the 3D environment. To replicate the same status in the onboard side, another instance of the OTS is created on the onboard side. The OTS on the onboard side is responsible of managing object template information to be considered for motion planning, e.g., as collision objects or attached collision objects to the robot. Both OTS were kept synchronized through the Communications Bridge that allows communications under constrained communication conditions. Any object template insertion in the OCS side triggers an object template insertion in the onboard side by forwarding a compressed version of the same ROS message; this works in the same way for object template removal. In case of a synchronization issue, both OTS can be re-synchronized by resetting the instantiated object template information. The architecture of the OTS can be seen in Figure 2.

3.3 Manipulation Workflow with Object Templates

The workflow of subtasks in an OT based approach requires that the operator identify the sensor data in the OCS that corresponds to the real object. After object identification has been performed, the following steps are required to perform manipulation with object templates (see Figure 4).

1. The operator inserts the template designed for the object of interest using the OCS. The operator manually aligns the 3D model of the template to sensor data that corresponds to the real object. In the current state of our approach, this step is the only one that is performed exclusively by the human operator. Any of the following steps can be executed either by a human operator or by any high-level behavior.

2. With the template in place, the stand pose of the template can be requested from the OTS and a footstep plan can be requested in case the robot is not able to reach the object with the end-effectors.

This footstep plan is reviewed by the operators and if sane, it is autonomously executed, otherwise, further planning and intervention can be requested.

3. Once the robot is in a position that the object can be reached by the end-effector the pre-grasp pose can be requested from the OTS and the execution of this step will take the robot's end-effector into a safe predefined position before approaching the object.

4. Then, the fingers need to be set to any "Open" configuration if available (for safety reasons, in our approach all robots start with fingers in a closed configuration).

5. Afterwards, the grasp pose is requested from the OTS and the execution of this step will take the robot end-effector into a position ready for grasping the real object.

6. The grasp posture of the fingers can then be set so that the robot takes control of the object (for non-prehensile grasps e.g., turning the valve with a stick, this step can be omitted).

7. If the robot has grasped an object that can be moved around in the environment, the object template can be attached to the end-effector. It is then considered during collision checking while planning motions and will be moved appropriately.

8. Then, usabilities of the object template can be selected and used for motion planning assuming this attachment as a rigid body transformation between the origin of the object template and the robot's end-effector.

9. Finally, the affordances of the object template can be requested from the OTS and be executed so that the robot performs the required arm motions to achieve the manipulation task.

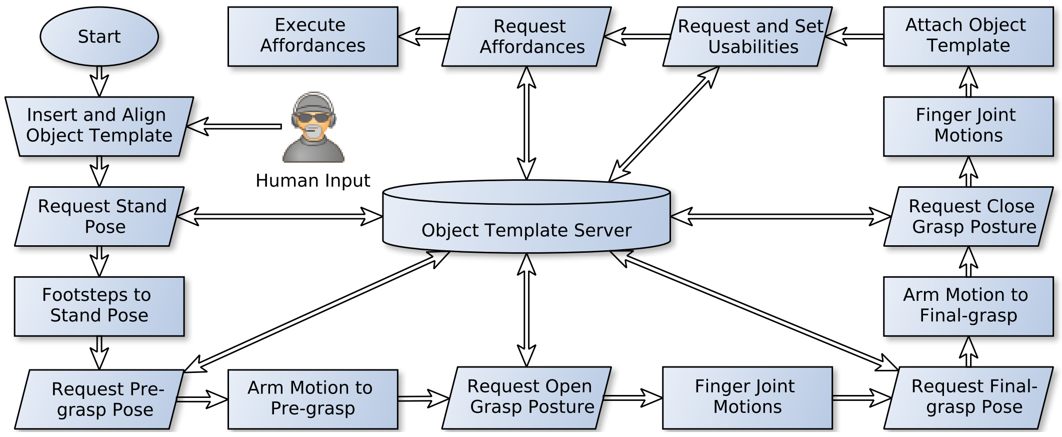


Figure 4: Workflow of a manipulation task using object templates. Trapezoids represent manual input that is always performed by a human operator. Parallelograms represent object template data requests to the OTS that can be done either by a human operator or by a high-level behavior. Rectangles represent processes executed by the remote humanoid robot.

4 High-level Behavior Control

The design philosophy behind our high-level behavior control approach is based on the challenges that a humanoid robot would face while carrying out remote search and rescue operations in collaboration with human operators. These challenges were exemplified in the rules of the DRC Finals. On the one hand, the sixty minute time constraint meant that we could not rely solely on the human operators to make every decision and perform every action; we would have to introduce significant autonomy. On the other hand, we also wanted to leverage the human operators' cognitive and perceptual capabilities. This gave rise to the concept of *collaborative autonomy*, which is the topic of Section 5. Furthermore, the degraded communications between operators and the robot motivated interaction between them at a higher level of abstraction. We achieved a significant level of abstraction via the use of object templates and affordances. Here, we are interested in abstracting the interaction between the operators and the robot at the task level.

We also accounted for the extensibility and generalizability of our high-level control approach. For example, we wanted our approach to not be specific to the eight DRC Finals tasks. Rather, it should be a framework for specifying and executing any high-level robot behavior. Furthermore, we wanted the approach to be compatible with any robotic system, with Team ViGIR's Florian and Team Hector's Johnny being two examples. Finally, we wanted to give team members the ability to quickly iterate on the creation and testing of high-level robot behaviors.

Taking these considerations into account, we model high-level robot behaviors as state machines, whose states abstract the various lower level system capabilities, such as those related to manipulation planning. Two of our contributions make collaborative autonomy possible. First, we introduce the notion of individual state machine transitions requiring a certain level of autonomy or *autonomy threshold* in order to be executed autonomously. By adjusting the autonomy level, the human operators have more or less control over the execution of a high-level behavior. Second, we imbue our high-level controllers with the ability to make requests to the human operators. Such requests include, but are not limited to, decisions and data (e.g., object templates and locomotion goals). We refer the interested reader to (Schillinger, 2015) for a detailed coverage of our approach to high-level control.

In this section, we first introduce terminology that supports our discussion of high-level behavior control. Then, we give an overview of the Flexible Behavior Engine (FlexBE), which is the software implementation of our approach to high-level behavior control. Finally, we discuss the interaction between high-level behavior control and manipulation planning and, in particular, template and affordance-based manipulation.

4.1 Behavior Control Definitions

We summarize the basic concepts that form the building blocks of our high-level behavior control approach, which is based on the model of hierarchical finite-state machines.

Definition: Behavior. *A behavior B specifies how the robot is meant to carry out a high-level task, including interaction with its environment and the human operators. A behavior is realized by the implementation of a corresponding state machine B_{SM}. Furthermore, each behavior defines a set of parameters B_P, where each $p \in B_P$ is specified when the execution of that behavior begins.*

Behaviors coordinate a set of existing, lower level, robot and system capabilities (e.g., grasping or footstep planning) via the states of a behavior's state machine.

Definition: State Implementation. *A state implementation*[16] *$s \in S$ interfaces system capabilities with high-level behaviors. Specifically, each state interacts with a single such capability. Each state defines a set of outcomes s_{Oc}. The execution of a state is terminated by returning an outcome, $oc \in s_{Oc}$, abstracting the result of its corresponding action, i.e., the result of activating a lower level system capability.*

A state is active while its corresponding action is being executed by some system component. Depending on the result of this action, an outcome is returned, leading to a transition in the enclosing state machine.

Definition: Hierarchical Finite-State Machine (HFSM). *A state machine $SM = (S_{SM}, t_{SM}, SM_{Oc})$ composes a set of states S_{SM}, where each state $s^{(i)} \in S_{SM}$ is either the instantiation of a state implementation $s \in S$ or a lower level state machine SM'. The state machine's outcomes, SM_{Oc}, indicate the termination of the state machine's execution. When a state $s^{(i)}$ returns a specific outcome $oc^{(i)} \in s^{(i)}_{Oc}$, the corresponding*

[16] We will omit "implementation", whenever the meaning is unambiguous, for the sake of brevity.

transition function of the state machine, $t_{SM} : S_{SM} \times s_{Oc} \rightarrow S_{SM} \cup SM_{Oc}$, defines which state is executed next or which outcome of the state machine is returned.

In addition to the control flow (logic), a state machine also coordinates the data flow. Data flow is relevant because the input data of an action corresponding to a certain state typically depends on the output data of previous actions. For example, a state interfacing with a manipulation capability may require an object template as input data. The template was the output data of some previous action/state in the control flow.

Definition: Userdata. *Each state machine SM additionally defines userdata D_{SM}, represented by a set of key-value pairs, where $f : K_{D,SM} \rightarrow V_{D,SM}$ maps each specific key $k \in K_{D,SM}$ to its respective value. States define userdata keys they need, s_I, and provide, s_O.*

Passing data from state A to state B with $s^{(A)}, s^{(B)} \in S_{SM}$ is thus defined as $f(k_{I,sB}) \mid k_{I,sB} = k_{O,sA}$ where $k_{I,sB} \in s_I^{(B)}$ is an input key of $s^{(B)}$ and $k_{O,sA} \in s_O^{(A)}$ is an output key of $s^{(A)}$.

Definition: Autonomy Level and Threshold. *During behavior execution, the current autonomy level is denoted a_{SM}. In a state machine SM, each outcome of a state $oc^{(i)} \in s_{Oc}^{(i)}$ defines its own autonomy threshold $a_{oc}^{(i)}$. If $a_{oc}^{(i)} \geq a_{SM}$, then the corresponding transition $t_{oc,i} = t_{SM}(s^{(i)}, oc^{(i)})$ is blocked.*

In other words, the autonomy threshold and the current autonomy level define a *guard condition*, $a_{oc}^{(i)} < a_{SM}$, on the transition $t_{oc,i}$. Transitions that satisfy their guard condition are executed autonomously. All transitions that do not satisfy their guard condition, i.e., if $a_{oc}^{(i)} \geq a_{SM}$, are suggested to the human operator and require explicit manual confirmation in order to be executed. The autonomy level a_{SM} can be adjusted at any time during behavior execution.

Definition: Behavior Input. *Providing data to a behavior is modeled as a special action p_{ID}. Input data D_i of p_{ID} specifies the required type of data while D_o provides the result. This result is evaluated and provided as output userdata $k_{O,sID}$.*

The behavior input functionality allows the operator to provide data — only available at the control station — to the remote robot. More importantly, states implementing this input functionality allow the behavior to autonomously initiate this data transfer by sending a request to the operator, who then responds with the corresponding data.

Definition: Behavior Modification. *Modifying a behavior B, i.e., its state machine B_{SM}, corresponds to adding or removing elements to or from S_{SM}, changing the properties of any $s(i) \in S_{SM}$, or changing the transition function t_{SM}.*

Behavior modifications can be used by the human operator to alter the structure of the active behavior at runtime, e.g., in response to unexpected situations during system operation. This functionality was not used during the DRC Finals and therefore we will not expand on it further in this paper. We refer the interested reader to (Schillinger et al., 2016).

4.2 Flexible Behavior Engine (FlexBE)

Our high-level control philosophy described above was implemented in FlexBE,[17] the Flexible Behavior Engine (Schillinger, 2015). FlexBE consists of two main parts. The back-end, i.e., the engine itself, is a behavior development and execution framework which evolved from the SMACH high-level executive (Bohren and Cousins, 2010). Most notably, the execution paradigm was changed to be non-blocking and thus, be able to incorporate operator interaction like adjustable autonomy or manually triggered transitions in parallel to automated behavior execution.

[17] `https://github.com/team-vigir/flexbe_behavior_engine`

The front-end to the framework and executive is FlexBE's graphical user interface[18] (GUI), which comprises four views: Behavior Dashboard, State Machine Editor, Runtime Control, Configuration. Two of these views, the Editor and Runtime Control, are depicted in Figures 5 and 6, respectively. Not only is the GUI itself an important aspect of the front-end, but also its way of bandwidth-efficient communication with the back-end in order to realize a robust robot-operator interaction.

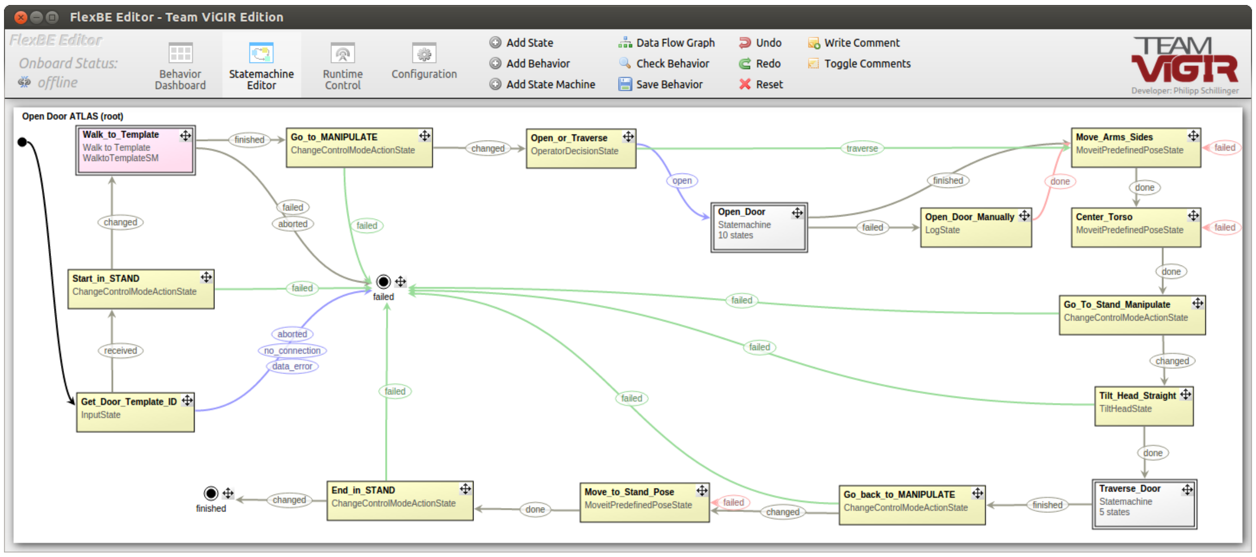


Figure 5: A hierarchical state machine implementing a behavior for the "Door" Task. The initial state is indicated by the black arrow originating from the top left. The behavior has two outcomes, `finished` (bottom left) and `failed` (center). Yellow states are parametrized state implementations. Gray states are state machines. Purple states are other behaviors embedded in this one. The color of each transition corresponds to its autonomy threshold: `Off` (gray), `Low` (blue), `High` (green) and `Full` (red).

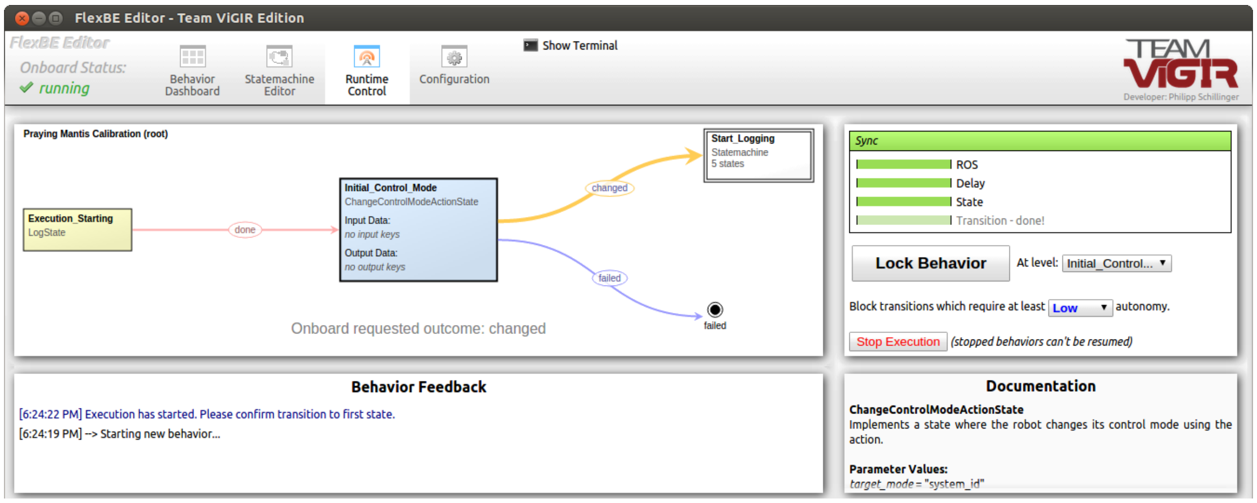
Developers create the state implementations S and then use FlexBE's State Machine Editor to design and parametrize hierarchical state machines B_{SM} that implement high-level behaviors[19] B. The design of a state machine SM includes, among other activities, the parametrization of the state implementations $s^{(i)} \in S_{SM}$, the specification of the behavior logic by connecting the outcomes $oc^i \in s^{(i)}_{Oc}$ of each state $s^{(i)}$ to other states, thus defining the transition function t_{SM}, and the selection of an appropriate autonomy threshold $a^{(i)}_{oc}$ for each transition $t_{oc,i}$. An example of a fully designed behavior is depicted in Fig. 5.

In order to integrate a new robotic system with our high-level behavior control approach, developers have to create new state implementations S, which will let their behaviors interface with their lower level system capabilities. Some state implementations are robot-agnostic (e.g., a decision state or a state that displays a message). Moreover, Team ViGIR and Team Hector were able to use the same state implementations since they were also using the same software system. If the same state implementations are used, then entire high-level behaviors may also be reused for the control of the new robotic system.

During system operation, the human operator uses FlexBE's Runtime Control to start, monitor, direct, and stop behavior execution (Fig. 6). A capability that is of particular interest to this paper is that FlexBE allows the human operator to adjust the current autonomy level a_{SM} on the fly. In addition, the operator can choose to override the behavior's suggested transitions. For example, in Fig. 6, the behavior is indicating an outcome and suggested transition (highlighted), but the operator can click on the other outcome in order to force a different transition.

Finally, it's worth noting that there are actually two state machines running every time a behavior is executed (c.f. Fig. 2). This design was motivated by the need to monitor and direct execution remotely

[18] `https://github.com/team-vigir/flexbe_chrome_app` [19] `https://github.com/team-vigir/vigir_behaviors`



Onboard Status: running

Onboard requested outcome: changed

At level: Initial_Control...

Block transitions which require at least Low autonomy.

(stopped behaviors can't be resumed)

[6:24:22 PM] Execution has started. Please confirm transition to first state.
[6:24:19 PM] --> Starting new behavior...

ChangeControlModeActionState
Implements a state where the robot changes its control mode using the action.

Parameter Values:
target_mode = "system_id"

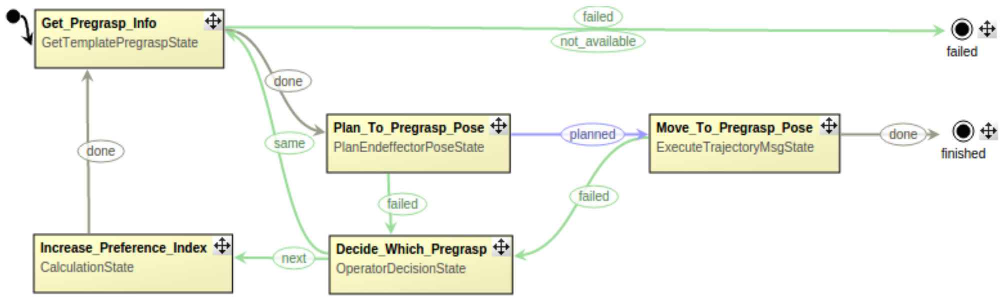
Figure 6: During system operation, the human operator oversees and directs behavior execution. The active state s (blue) is shown at the center of the Runtime Control view. To its left is the previously executed state. To its right are the possible state outcomes, $s_{Oc} = \{\texttt{changed}, \texttt{failed}\}$, and corresponding transitions, $t_{\texttt{changed}}$ and $t_{\texttt{failed}}$. Once an outcome has been returned by the state, the corresponding transition is highlighted. Here, the behavior is not transitioning autonomously because the current autonomy level ($a_{SM} = \texttt{Low}$) is not strictly greater than the transition's autonomy threshold ($a_{\texttt{changed}} = \texttt{Low}$).

and under degraded communications. The on-board state machine, B_{SM}, is the one controlling the robot by activating system capabilities, performing calculations, etc. In FlexBE's Runtime Control view, the operator is monitoring the execution of the behavior mirror, $B_{SM}^{(m)}$, which is a "ghost" version of B_{SM}. FlexBE keeps the behavior mirror synchronized with the on-board behavior. The synchronization goes both ways; autonomous transitions taken on-board are reflected in the mirror and manual transitions forced by the human operator are communicated to the on-board behavior.

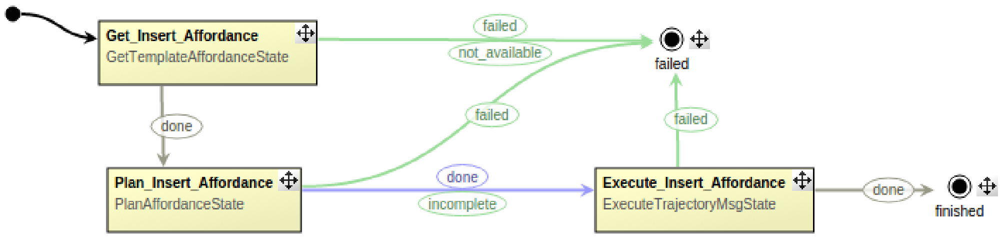
4.3 Behaviors and Object Templates

In Sections 3.1.1 and 3.2 we discussed how the operator can use teleoperation and object templates in order to command robot motion. We now show that high-level behaviors can also leverage our manipulation planning subsystem. This introduces another layer of abstraction and enables task-level autonomy. In brief, our system design allows us to carry out the template-based manipulation workflow depicted in Fig. 4 programmatically. The steps of the workflow are implemented as states and the workflow itself is realized as part of a HFSM. Examples of using object templates and affordances in high-level behaviors composed in FlexBE are provided in Figures 7a and 7b, respectively. The state implementations do not carry out manipulation-related computations themselves. Rather, they interface with manipulation system subcomponents, such as the (on-board) object template server and the *vigir_move_group_action*. The outcome of each state depends on the response of the corresponding subcomponent, e.g., success or failure.

Of particular interest is the first step in the workflow, the object template insertion and alignment. Since we leverage the cognitive abilities of the operator for object detection, the behavior requests the operator to identify the object of interest and then insert and align the appropriate template. As we alluded to in Section 4.1, this interaction is realized via the behavior input action. This is a key component of our system, which we discuss further in Section 5.2 in the context of behavior-driven requests for operator input.



(a) Given an arm side parameter $p_{\texttt{side}} \in B_P$ (left or right), a template ID (userdata), and a pre-grasp ID (userdata), this state machine requests the pre-grasp pose P_{PG} with the desired ID and a motion plan for the (left or right) end-effector to move to P. It then executes that motion plan. This particular state machine also includes logic for retrying after failures.



(b) Given an arm side (left or right), a template ID, and an affordance (here, "insert"), this state machine requests the desired affordance A and a motion plan for the (left or right) end-effector to perform A. It then executes that motion plan.

Figure 7: Two state machines that exemplify the use of object templates and affordances by behaviors.

5 Collaborative Autonomy

Throughout this paper, we have been referring to the concept of *collaborative autonomy.* In this section, we provide a definition and discuss its advantages.

Definition: Collaborative Autonomy. *Given a high-level task and a team that comprises a robotic system and any non-zero number of human operators, the team exhibits collaborative autonomy if the robotic system carries out the task autonomously, when capable of doing so, initiates requests to the human operators when necessary, and can respond according to their input at any time. Such autonomy-driven requests include requests for data, requests to perform actions, requests for the operators' permission or confirmation, and decisions that the system wants the operators to make.*

Collaborative autonomy can be illustrated as in Figure 8, where a robot (represented by the glider) is controlled by a high-level behavior (represented by the trainee in the front seat), who is supervised by a human operator (represented by the expert in the back seat). In the best case scenario, the high-level behavior is going to perform the complete task by itself. However, the human operator is there to provide support in form of information or take control, when necessary. The high-level behavior knows the basics of operating the device, e.g., how to steer the glider left or right. The type of input that the high-level behavior requests is at a task-level, e.g., "Is this the correct angle for landing?" If the high-level behavior is not able to complete part of the task, the human operator can (temporarily or permanently) take control, e.g., "I cannot land, please do it for me." Moreover, the operator (backseat expert) can also intervene at any time.

In the context of this paper, the robotic system mentioned above is a high-level behavior controlling a humanoid robot in order to carry out a manipulation task (e.g., one of the DRC Finals tasks). The definition above does not specify the number of human operators, as long as there is at least one. In the DRC Finals, Team ViGIR competed with four operators, while Team Hector competed with three. In order to make the

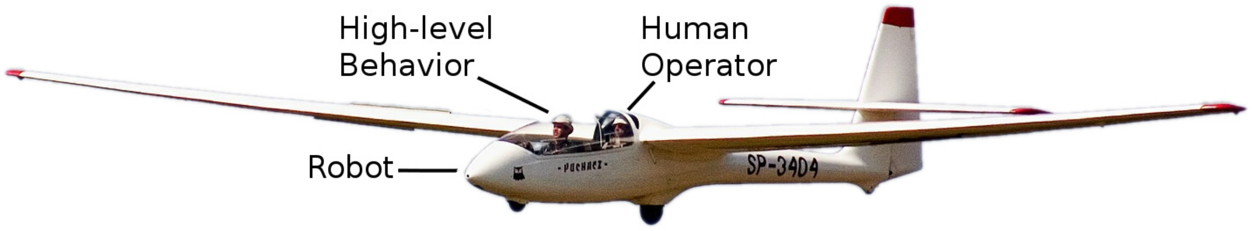


Figure 8: A metaphor for collaborative autonomy.

notion of collaborative autonomy more concrete, we will focus on Team ViGIR. However, the concept readily applies to different settings. For example, a single operator can take on multiple roles.

5.1 Operator Roles

Team ViGIR used multiple operators for both the DRC Trials and Finals. The individual operator stations were separate instances of the same user interface that shared data between operators. Thus, if one operator requested a point cloud, the same point cloud would be visible on all stations. This allowed the operators to coordinate verbally with one another, for example, where one operator could make a request to another operator to gather more sensor data or verify execution status (Kohlbrecher et al., 2015); this reduced the cognitive load on any one operator. For the DRC Finals, Team ViGIR used four operators with well-defined roles: Supervisory, Primary, Auxiliary, and Immersed operators. Team Hector defined almost the same operator roles except the immersed operator, as Johnny's sensors do not deliver long range 3D data like Atlas. The operators' relative positions in the control room is depicted in Fig. 1 (center). Table 1 describes the main responsibilities of each operator.

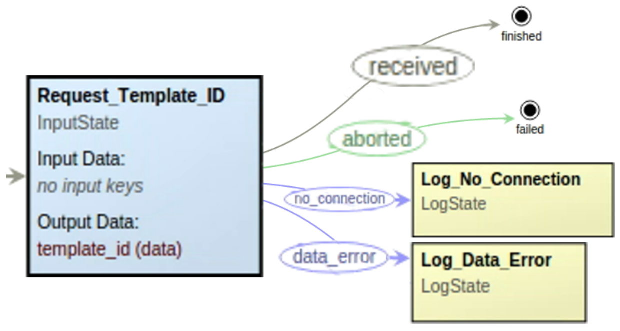
Operator	**Role Description**
Supervisory	The supervisor operator (or just supervisor) was responsible for coordinating the human-robot team. This operator started and stopped the execution of high-level behaviors, interacted with the active behavior, and adjusted its autonomy via FlexBE's GUI. The supervisor also communicated with the other operators verbally, in order to keep them up to date with the progress of behavior execution and to convey requests.
Primary	The primary operator (or main operator) was responsible for monitoring and verifying actions generated by the active high-level behavior. The primary operator would also intervene and interact with the OCS in order to command the robot to execute actions such as footstep or manipulation planning, if the corresponding behavior state failed.
Auxiliary	The auxiliary operator was responsible for perception tasks, such as inserting templates or other semantic information, as requested by high-level behaviors. The auxiliary operator would also gather perception data on the OCS side in support of the primary operator's situational awareness. For Team ViGIR, the auxiliary operator also served as team lead during the runs, and was responsible for making the final decisions on tactics.
Immersed	The immersed operator used an Oculus Rift DK220 virtual reality head-mounted display to observe task execution. This permitted this operator to visually navigate the 3D scene and thus assist in situational awareness. (Team ViGIR only)

Table 1: Roles of Team ViGIR's four and Team Hector's three operators in the DRC Finals.

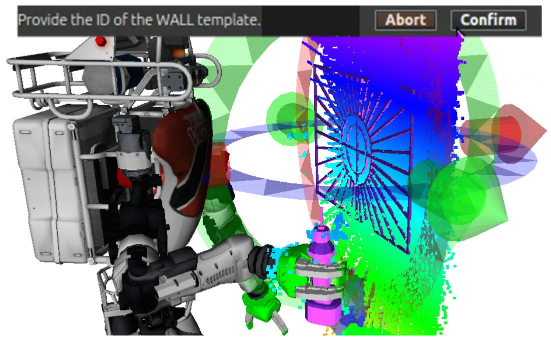
5.2 Interaction between Operators and Behavior Execution

The supervisor interacts with the active behavior via FlexBE's GUI. The interaction ranges from starting behavior execution to ending it prematurely, if necessary. In between, the supervisor monitors the autonomous transitions of the behavior's state machine, confirms the execution of transitions whose autonomy threshold is greater than the current autonomy level, and adjusts the autonomy level. Some of the non-autonomous transitions (e.g., decision points) may require that the supervisor communicates with the other operators, who have better situational awareness. If necessary, the supervisor can also force synchronization between the behavior mirror (OCS-side) and the actual behavior running on-board the robot. Finally, the supervisor is responsible for online behavior modifications (Section 4.1), which we did not use in the DRC Finals.

The interaction of the auxiliary operator with the active behavior revolves mainly around object templates. For the active high-level behavior to command the robot to approach and manipulate objects, it requires information embedded in the corresponding object templates. To this end, a specific behavior state, the `InputState` parametrized accordingly, can send a request to the operators (Fig. 9a). Specifically, the auxiliary operator is responsible for inserting the correct object template, aligning it with the sensor data, and then responding to the behavior's request (Fig. 9b), which makes the template's ID available to the `InputState`. The auxiliary operator is also responsible for confirming that a footstep plan generated at the initiative of the active behavior looks safe and reasonable, given the available sensor data.



(a) An `InputState`, parametrized to request an object template ID, and its outcomes.



(b) The request displayed to the operators. Here, the message reads "Provide the ID of the WALL template."

Figure 9: Execution of an `InputState` (left) results in a request being displayed in the screens of the primary, auxiliary, and immersed operators (right). Here, the auxiliary operator is responsible for responding.

While the primary operator can also respond to direct requests from the behavior — most notably, requests for locomotion goals (feet pose) — their interaction was indirect during the DRC Finals. In particular, if a behavior state corresponding to some action failed,[20] the supervisor has the option of asking the primary operator to perform that action, and only that action, in lieu of the state. Once the primary operator has performed the action, the supervisor can trigger the failed state's transition that corresponds to completion of that action. Then, behavior execution can continue normally.

Finally, the immersed operator did not respond to behavior requests directly. Rather, this operator's situational awareness helped the supervisor make various decisions that the active behavior requested.

6 Experimental Evaluation

In this section we describe the performance of the approach used by both Team ViGIR and Team Hector during the DRC Finals. Additionally, we present experimental evaluation obtained during systematic laboratory experiments. This evaluation focuses on showing how high-level behaviors and human supervisors can efficiently collaborate to perform manipulation tasks with help of object template information.

[20] We are assuming that the `failed` transition's autonomy threshold is such that it is not executed autonomously.

6.1 DRC Finals Evaluation

The DRC Finals held in Pomona California in June 2015, presented challenges in a wide range of areas. On one side, the robot system capabilities to complete the individual tasks designed for the challenge were put to the test. On the other side, the challenges of real-world field scenarios such as temperature, ground slope, and communication interferences among others, pushed the humanoid robots to their limits. To keep the focus on the experimental evaluation that concerns this paper, the first task of the DRC Finals which corresponded to driving a vehicle is omitted. All evaluations of manipulation tasks are considered to start after the auxiliary operator inserts the object template and are considered to be finished when the main objective is achieved.

6.1.1 Team ViGIR Open Door (Day 1)

During day one, Team ViGIR opened the door using a combination of high-level behaviors, object template manipulation, and teleoperation. The auxiliary operator identified the door and inserted the door template, aligning it so that the door-handle matched. The high-level behavior requested to the auxiliary operator the ID of the template to start the task. First, the high-level behavior requested from the object template server the stand pose to open the door, which was used as input to request a footstep plan for stepping towards the door. The high-level behavior commanded the robot to walk to the stand pose; the robot responded to these commands, but the response took longer than expected.

After reaching the stand pose and autonomously positioning the torso to face towards the door, the behavior's attempts at requesting an arm motion plan to the door handle's pre-grasp pose failed repeatedly. We later discovered that this was due to the unplanned[21] degradation of communications between the field computer (where FlexBE, i.e., the active behavior, was running) and the computers on-board Atlas (where the planning system was running). Regardless, the supervisor communicated the failures to all operators and asked the primary operator to take over.

The primary operator commanded the robot to move the arm to the pre-grasp pose. Misalignments from the door template prevented the robot to reach the grasp pose. Because of this, the primary operator proceeded to execute the arm motion using Cartesian teleoperation. Once the robot's hand grasped the door-handle, the primary operator requested the "Turn Clockwise" affordance of the door template. However, the turning motion of the door-handle was not enough to unlatch the door. So when the primary operator requested the "Push" affordance, the grasp from the door-handle was lost. The primary operator proceeded to perform Cartesian teleoperation a second time and the door was opened by pushing downwards on the door-handle. Figure 10 shows the series of images from the robot's camera view of the execution of the task. Figure 17a shows a timeline of the events required to perform the task.

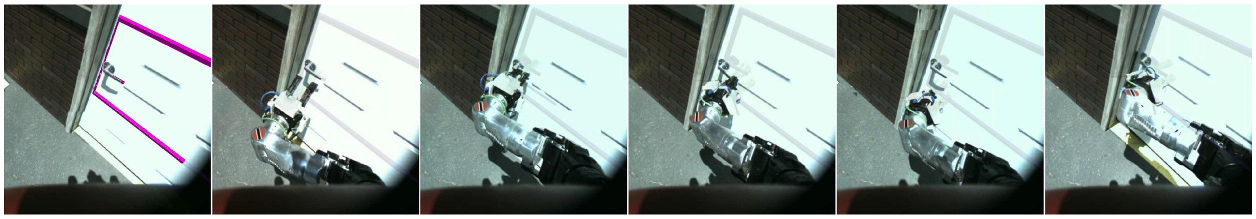


Figure 10: Robot's camera view from Door task. From left to right: Door Template aligned, Pre-grasp, Grasp, Turn Clockwise affordance, Push affordance (fails to open), door opened after Cartesian teleoperation.

As can be seen, initially, the supervisor operator was able to command the robot using high-level behaviors. However, after a failure, the primary operator intervened for the rest of the task using object template control (grasp and affordance commands) and Cartesian teleoperation. This experiment shows the cascade

[21] DARPA's planned degradation of wireless communications was between the OCS and the field computer(s).

of changes in the control approach taken by the operators to achieve the task. Given the flexibility to change approaches on the fly, it is shown how the operators were able to adapt to higher-layer system failures and use lower layers of manipulation control to open the door.

6.1.2 Team ViGIR Turn Valve (Day 1)

After traversing the opened door, the next task was to open a valve 360 degrees counter-clockwise. At this point, there was no possibility to use high-level behaviors due to communications issues; for this reason the primary and auxiliary operators performed the rest of the tasks.

The auxiliary operator inserted and aligned the template to match the pose of the real valve. The primary operator requested the stand pose from the template and the footstep plan was calculated. It was discovered that given the communication constraints, footstep plans with more than 10 steps were not able to be visualized in the OCS. For this reason, manual creation of footstep plans was required to reach the valve. Once the robot stood in front of the valve, the primary operator successfully used grasp commands to place the robot's wrist attachment (a poking-stick located in the wrist of the left hand) in-between the crossbars of the valve. Afterwards, the operator used the "Open" affordance of the valve template to turn the valve. Figure 11 shows the series of images from the robot's camera view of the execution of the task. Figure 17b shows a timeline of the events required to perform the task.

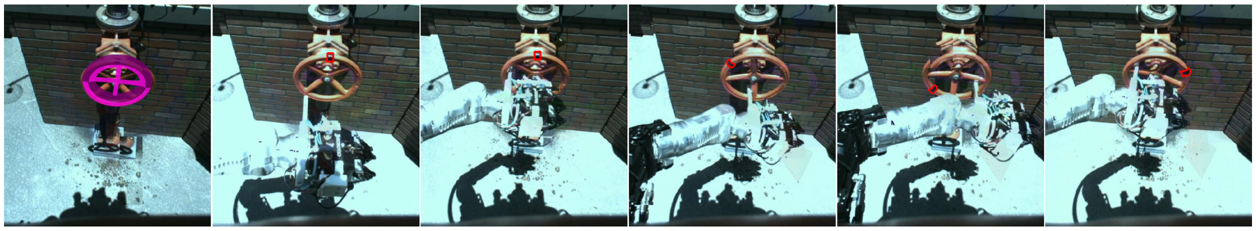


Figure 11: Robot's camera view from Valve task. From left to right: Valve Template aligned, Pre-grasp, Grasp, Open affordance 45, 135, and 270 degrees.

6.1.3 Hector Open Door (Day 1)

During day one, Team Hector also opened the door using a combination of high-level behaviors, object template manipulation, and teleoperation.

The first attempt was mainly commanded by the high-level behavior. The auxiliary operator placed the door template based on captured point cloud data. Using the template pose information, the high-level behavior was able to generate the transitions to move the left hand to the pre-grasp and afterwards to the grasp pose. The primary operator noticed a significant offset through the camera images which was likely caused by sensor noise and non-perfect LIDAR to robot frame calibration. Several iterations between the auxiliary operator to perform template adjustment and the supervisor operator to generate manual transitions in the high-level behavior were required to bring the robot's hand into the grasp pose of the door handle.

Due to a hardware failure, Team Hector was not able to continue execution of the arm motions to open the door, this forced them to request a reset. After this reset, the primary operator intervened and continued the task executing Cartesian teleoperation to re-grasp the door handle. Once the robot's hand was confirmed to have grasped the door handle, the primary operator requested the execution of the "Turn Clockwise" affordance of the Door Template. In this way, Team Hector managed to open the door.

The concept of work sharing between operators and robot worked well, as all operators had been able to provide all needed information to the behavior. This additionally shows that the high-level software architecture developed initially for the Atlas robot was successfully implemented on the THORMANG humanoid robot. The lack of recorded information from the task, prevents us from providing high-detail information; in lieu of an explicit timeline, we present in Figure 12 a series of images from the robot executing the door task.

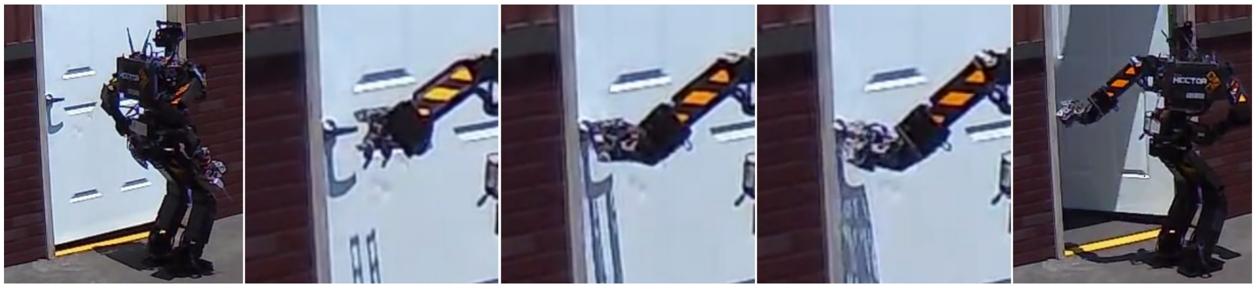


Figure 12: DRC external footage from the Door task. From left to right: Setup, Pre-grasp, Grasp, Open affordance, Door open.

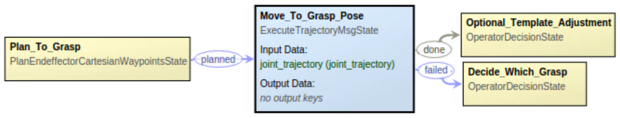
6.1.4 ViGIR Open Door (Day 2)

On day two, the behavior started by requesting that the door template be placed and aligned. The auxiliary operator responded by providing this information. The behavior was then able to request and autonomously execute a footstep plan towards the door template's stand pose. Since we had addressed the communications issue from day one (by moving FlexBE to the on-board computers), the supervisor was confident enough to switch the behavior's current autonomy level to `High`. The behavior had the robot look down towards the door handle and turn its torso. It then requested the auxiliary operator to adjust the door template, if necessary. Afterwards, the behavior autonomously moved the robot's arm to the pre-grasp pose and then the grasp pose (Fig. 13a). Once the hand moved to the grasp pose of the door template, the behavior asked for permission to proceed. The primary operator noticed that the robot's hand was not in the correct pose. After adjusting the hand (using the affordances of the door template), the supervisor gave the high-level behavior permission to proceed. Had it not been for this application of collaborative autonomy, the robot would have missed the door handle, which would have required a more involved operator intervention.

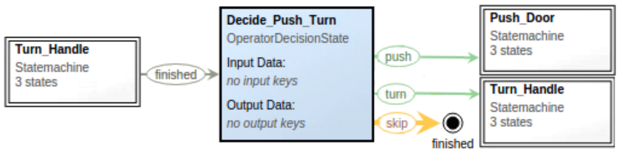
The behavior, still in `High` autonomy, executed the "Turn Clockwise" affordance of the door template. Once the affordance execution was finished, the behavior asked whether it should proceed with pushing the door or turn the handle more (Fig. 13b). The primary operator once again noticed a misalignment of the robot's hand and proceeded to adjust it using teleoperation. Then, after having communicated with the other two operators, the supervisor had the behavior repeat the turning part. The robot proceeded to turn the handle, but the motion was still not sufficient for unlatching the door; once again the supervisor had the behavior repeat the turning part. The behavior's response took longer than expected and so the primary operator completed the turning motion using the affordances of the template. The door handle unlatched and the door opened on its own (due to gravity). Thus, the supervisor had the behavior skip the execution of the "Push" affordance. Figure 13 shows FlexBE snapshots from behavior execution. Figure 17c shows a timeline of the events required to perform the task.

6.2 Laboratory Evaluation

This section describes lab experiments performed to demonstrate the potential of the approach presented in this paper. During the DRC Finals, hardware and communication issues prevented us from performing at the competitive level that our approach allows. For this reason, laboratory experimentation was performed using the same software setup as used during the DRC (with the exception of using our communications bridge). These tests were performed to compare a pure-operator execution of the task against a collaborative execution between high-level behaviors and operators. Due to a damaged leg sensor, the robot was not able to walk after the DRC, for this reason lab experimentation does not include walking or stepping.



(a) The behavior is moving the hand to the door handle's grasp pose. The current autonomy level, `High`, is higher than that of the two possible transitions. Therefore, the next state will be executed without operator intervention.



(b) Having turned the door handle, the behavior is asking whether it should turn it more, push the door, or proceed to the next step. The execution will not continue until the supervisor selects one of the three transitions.

Figure 13: Two snapshots of "Open Door" behavior execution during the second day of the DRC Finals. The top-level state machine corresponding to this behavior can be seen in Fig. 5.

6.2.1 Opening the Door (Operator Only)

We performed lab experimentation of opening the door to demonstrate that the system is capable of performing this task at an affordance level, as opposed to teleoperation as during day one of the DRC. In this case, the robot was already placed in a position where the door handle was reachable by the robot. The Door Template was inserted and aligned to the sensor data. The operator visualized the previews of the pre-grasp pose and the grasp pose to verify that the robot was able to reach both poses. Then, the operator commanded the robot to execute the arm motions for the pre-grasp pose. After reaching the pre-grasp pose, the operator needed to request the robot to set the fingers in a grasp posture before approaching the door handle. The operator then commanded the robot to move the arm to the grasp pose and setted the grasp posture that made the fingers to grasp the door handle. Once the robot had control of the door handle, the operator executed the "Turn Counter-clockwise" affordance of the Door template and the door got unlatched. Afterwards, the operator executes the "Push" affordance and the door was opened. Figure 17d shows a timeline of the events required to perform the task.

6.2.2 Opening the Door (High-level Behavior)

The same experiment as in Section 6.2.1 was also performed by a high-level behavior — monitored by the supervisor and in collaboration with the auxiliary operator. First, the behavior requests that the auxiliary operator inserts and aligns the door template. Once the template is in place, the behavior, executing in `High` autonomy level, is able to carry out all the remaining actions (pre-grasp, grasp, execution of affordances, etc.). The supervisor operator only had to confirm the few state machine transitions that had an autonomy threshold of `High` or `Full`. Figure 14 shows the series of images from the robot's camera view of the execution of the task. Figure 17e shows a timeline of the events required to perform the task and Table 2 indicates the exact task completion time. The high-level behavior, in collaboration with the auxiliary operator, opened the door twice as fast as the primary operator acting alone.

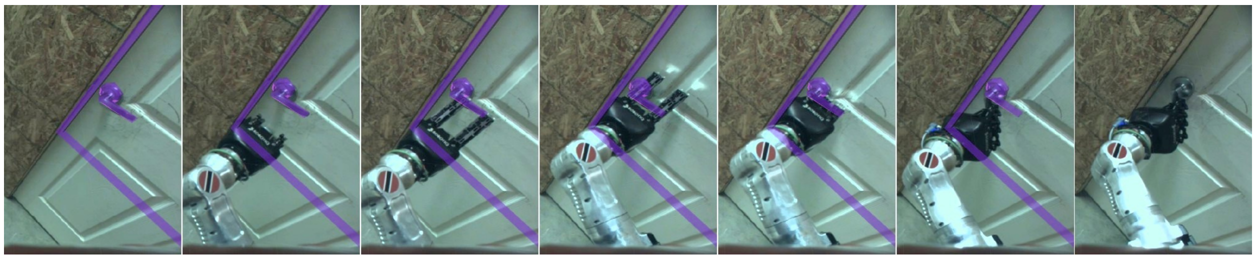


Figure 14: Robot's camera view from Door task. From left to right: Door Template aligned, Pre-grasp, Open fingers, Grasp, Close fingers, Open Clockwise affordance, Push affordance.

6.2.3 Turning the Valve (Operator Only)

For this task, the robot was placed in a position where the valve was reachable so that locomotion was not required. The operator inserted the valve template and aligned it to the sensor data. Afterwards, the operator selected the visualization of the pre-grasp pose and initiated execution of the arm motion. Then, the same procedure was done to reach the grasp pose, which put the wrist attachment of the left hand in-between the crossbars of the valve. Finally, the operator selected the “Close” affordance of the valve template to generate the circular arm motions to turn the valve. Figure 15 shows the series of images from the robot’s camera view of the execution of the task. Figure 17f shows a timeline of the events required to perform the task.

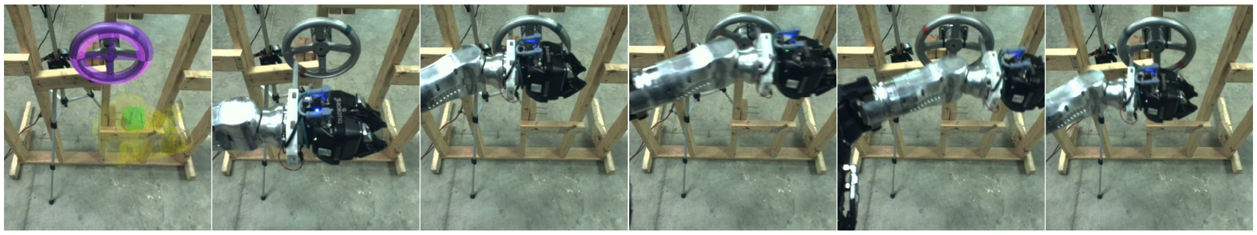


Figure 15: Robot’s camera view from Valve task. From left to right: Valve Template aligned, Pre-grasp, Grasp, Open affordance 90, 180, and 270 degrees.

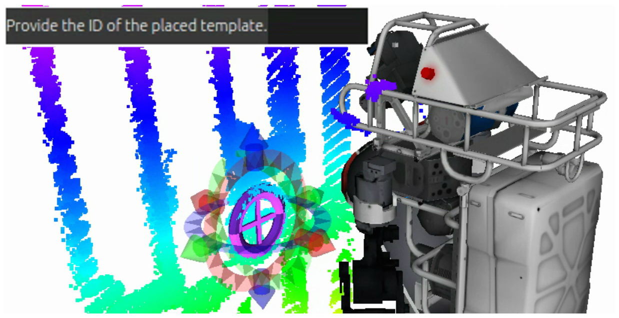
6.2.4 Turning the Valve (High-level Behavior)

The same experiment as in Section 6.2.3 was also performed by a high-level behavior — monitored by the supervisor and in collaboration with the auxiliary and primary operators. First, the behavior requests that the auxiliary operator inserts and aligns the valve template. Once the template is in place, the behavior, executing in `High` autonomy level, moves the robot’s arm to the pre-grasp pose. It then asks the primary operator to (optionally) adjust the hand’s position to ensure that the wrist attachment will be able to slide into the valve. The primary operator responded that no adjustment was necessary. Thus, the supervisor allowed behavior execution to continue. Afterwards, the behavior was able to complete the task mostly autonomously; the supervisor made a few transition confirmations and high-level decisions. Highlights from this experiment are depicted and discussed in Fig. 16. Fig. 17g shows a timeline of the events required to perform the task and Table 2 indicates the exact completion time. The high-level behavior, in collaboration with the auxiliary operator, turned the valve 150% faster compared to the primary operator acting alone.

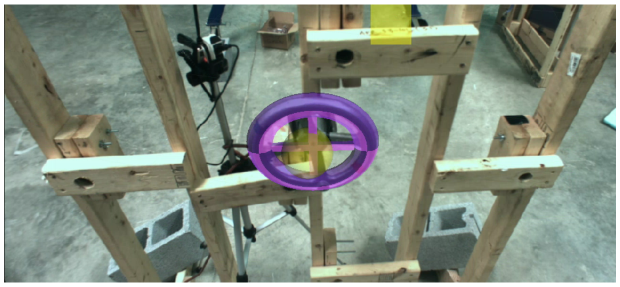
6.3 Timeline Results Analysis

Figure 17 shows a timeline of the events that happened during the DRC Finals runs and the laboratory experimentation. In this timeline it can be appreciated the periods of time where the robot was stepping and when manipulation was being performed. To fairly compare times, time starts after the robot has stopped locomotion. This timeline mainly shows when the robot was running under high-level behaviors and when it was being commanded by the human operator. Tasks that were run under a high-level behavior start with a request from the behavior (shown in yellow) to the auxiliary operator to provided the template ID of the aligned template (shown in lightgreen). The auxiliary operator responds to this request (shown in purple), and the behavior continues execution. The supervisor operator monitors the autonomy of the high-level behavior (shown in gray) and can execute priority manual transitions (shown in black).

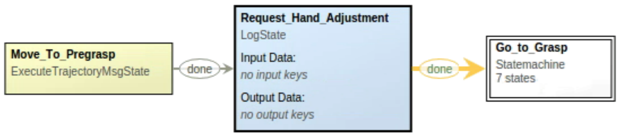
For matters of space, we will give detailed timeline description only of the events during the Door task on day two (see Figure 17c), the other tasks can be analogously analysed. During the second day, after accomplishing the drive task, a reset was requested to skip the egress task. This reset consisted of a 10 min pause; in the meanwhile, the operators prepared the high-level behavior (shown in pink). After the reset pause



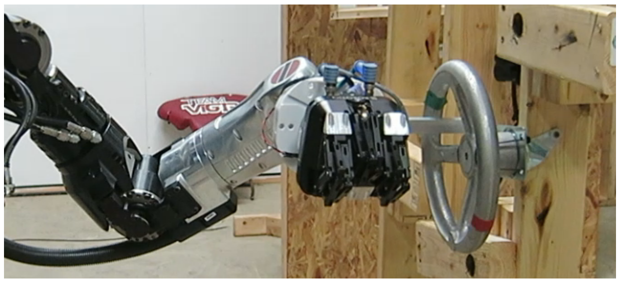
(a) The behavior is asking the auxiliary operator to insert, align, and provide the ID of the valve object template.



(b) The operator has placed the template and is now confirming the alignment using the first-person camera view.



(c) Before inserting the wrist adjustment in the valve, the behavior asks the supervisor to confirm.



(d) Once the operator confirms, the behavior proceeds to insert the wrist attachment and turn the valve.

Figure 16: "Turn Valve" behavior execution in `High` autonomy level (i.e., most transitions do not require the supervisor's permission). The auxiliary operator placed the valve object template (top). The primary operator confirmed that the wrist attachment was aligned with the gaps in the valve (bottom).

was over, the supervisor gave a manual transition to the behavior, and the robot started stepping (shown in blue). The robot autonomously walked to the stand pose of the door template and rotated the torso towards the door (upper body motions including manipulation are shown in darkgreen). The supervisor operator changed the autonomy level to `High` so only transitions with equal or higher autonomy level were required to be confirmed. The behavior reminded the auxiliary operator to consider template alignment; the auxiliary operator responded and the supervisor operator confirmed the transition. The behavior autonomously commanded the arm motions to pre-grasp and grasp pose and once again, the behavior requested for a confirmation if the robot's hand was in the correct pose. The primary operator noticed a misalignment and started generating commands using object templates (shown in cyan) to command the robot's hand to be closer to the door's handle using the "push" affordance of the door template. The supervisor operator confirmed the transition and the robot started turning the door handle. The behavior requested for confirmation to continue turning the door handle or to start pushing the door. However, the primary operator again noticed a misalignment and proceeded with Cartesian teleoperation (shown in red) to adjust the robot's hand pose. The supervisor confirmed the transition and the behavior continued turning the door handle. The behavior requested confirmation to continue turning the door handle or to start pushing the door; the supervisor confirmed to continue turning the door handle. The robot did not react to this command and the primary operator executed the step using the affordance of the door template and the door opened.

As a general view, gray color means that the task was performed using high-level behaviors, which gets reflected in minimal operator input. Whereas tasks that present red color, show a lot of teleoperation from the primary operator, meaning less autonomy. Table 2 shows the manipulation times required to perform the task. Video footage of the DRC runs and laboratory experiments can be seen here[22].

[22] `https://www.youtube.com/watch?v=5bSwwnQXfgQ&list=PL09J37oL4U7BXuuNHWE54vcMwsJlgAjmJ`

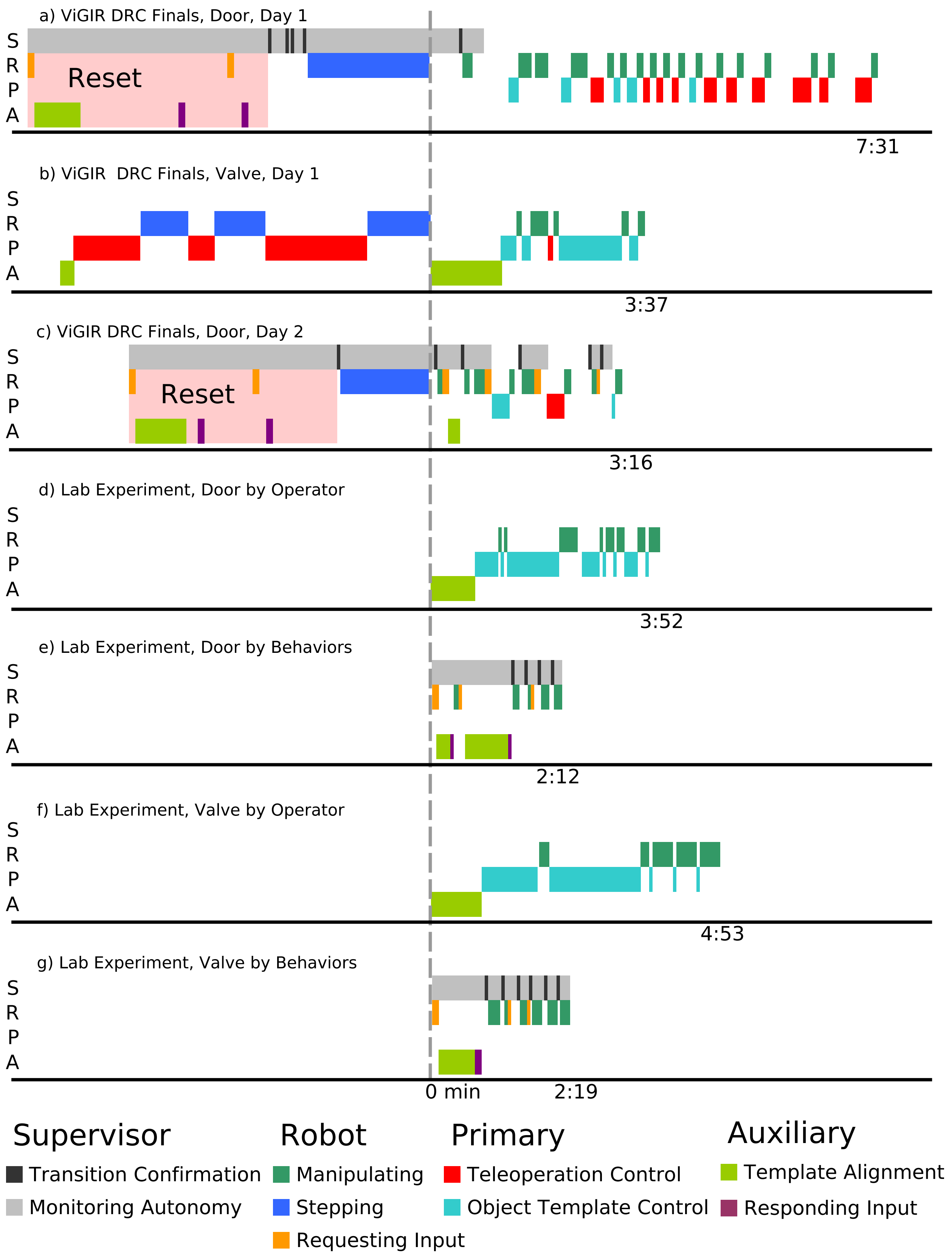


Figure 17: Timeline of actions during DRC Finals tasks and laboratory experimentation.

Time (min:sec)	Overall Time	Template Alignment	Teleoperation Control	Object Template Control	High-level Behaviors	Robot Motion
ViGIR Door Day 1	07:31	(in reset)	01:53	00:43	00:28	02:20
Lab Valve Operator	04:53	00:51	00:00	02:40	00:00	01:21
Lab Door Operator	03:52	00:45	00:00	01:57	00:00	01:00
ViGIR Valve Day 1	03:37	01:12	00:05	01:38	00:00	00:42
ViGIR Door Day 2	03:16	00:12	00:17	00:21	01:48	01:00
Lab Valve Behaviors	02:19	00:36	00:00	00:00	02:19	00:53
Lab Door Behaviors	02:12	00:57	00:00	00:00	02:12	00:31

Table 2: Time table of DRC and laboratory evaluation sub-tasks. Due to operators analysing situation, and behaviors and robot motions being simultaneously executed, the overall time does not reflect the sum of the times taken for each of the these sub-tasks.

7 Lessons Learned

This section discusses the lessons learned by Team ViGIR and Team Hector during the process of applying, using, and sharing the approach presented in this work. These lessons are based on our experience of what worked and what did not. We divide our lessons learned into three categories: Motion planning and Manipulation Control, High-level Behavior Control and Collaborative Autonomy, and Multi-team Collaboration.

7.1 Motion planning and Manipulation Control

Having the possibility to operate in the different modes of control, such as teleoperation, object template based manipulation, and high-level behaviors, provides a robust approach to account whenever failures happen from the higher-levels of control. This allowed us to operate in object template control when high-level behaviors failed and in teleoperation mode when object template interaction failed, for example, during the first day of the DRC Finals.

We identified that most of the high-level behavior failures originated from processes of lower layers of control, such as collision free motion planning, state estimation, and joint control. While generally providing a safety guard against environment collisions, planning with full collision avoidance can also fail due to spurious collision data generated from noisy sensor data. If these failures are properly considered by the higher levels of control, the states can automatically retry to execute these motion plans. In our case, whenever behavior failures persisted, the supervisor operator requested the auxiliary operator for a “world model reset” which resets the internal environment model to remove spurious collision data.

Overall control of object template manipulation involves several different steps: preview from pre-grasp and grasp poses, execution of pre-grasp, open hand, execution of grasp, close hand, attach object template, detach if necessary, select usabilities, define parameters for object affordances and their execution. These steps provide manipulation in a higher level compared to pure teleoperation, however, they are still manually executed, which can be error-prone.

Defining this information in an object template base allows abstraction to higher levels of control such as behaviors. Using a higher-level behavior to iterate through these steps allows the operators to act as observers of the task and intervene only in cases where the high-level behaviors cannot proceed.

The use of manual template alignment by the operator avoids automated object alignment as a potential source of error, but requires significant time and expertise by operators. A highly robust automated template alignment approch is thus desirable.

During robot experimentation it was discovered that the object template approach allows flexibility to adapt to situations where an object in the environment is found but no object template has been created for that object yet. This flexibility extends the use of the object template approach to achieve manipulation tasks with unknown objects, with intermediary objects, or to use objects in a way they were not designed for (Romay et al., 2015).

7.2 High-level Behavior Control and Collaborative Autonomy

The most noteworthy lesson that we learned, both during practice and at the DRC Finals, is the importance of having the logic of the high-level behaviors (i.e., the hierarchical state machines) reflect the manual workflow of the human operator(s). This allows the operator to easily follow behavior execution and, if necessary, intervene, perform a few steps, and then hand control authority back to the behavior. In terms of manipulation, one of the ways we achieved this was by designing state machines that mirrored the operator's object template-based workflow depicted in Fig. 4. We believe this is an important lesson, especially in light of the observation that it proved hard for the operators of many teams to take over when the autonomous system failed (DRC Teams, 2015).

Unlike many other teams, we did not employ a passive operator whose task was to read a script to the active operator(s). Our reasoning was that the high-level behavior's logic would play that role, as a corollary of the lesson above (same workflow). We included behavior states that would simply display messages to the operators. Thus, the active high-level behavior was able to prompt the operators (e.g., to optionally realign an object template or to visually confirm some condition). In addition, we discovered that, even if behavior execution fails completely (as in day one of the Finals), the supervisor can transform into a passive operator that directs the primary operator by reading the logic of the behavior's state machine (via FlexBE's GUI, e.g., see Fig. 5). In other words, the state machine's logic can act as a very detailed, nonlinear, visual script.

Our collaborative autonomy design led to the observation that the human operator can be seen as just another system capability from the point of view of a high-level behavior. That is, the behavior can be somewhat agnostic to whether an action was performed automatically or manually by the operator, as long as the necessary postconditions are met. This property contributed to our behaviors being quite robust to small discrepancies and errors, since the operators could easily assist.

During the finals, we discovered the need for additional "loopbacks" in our state machines to allow repeated attempts of particular states or sequences of states. While this is permitted by HFSMs in general, and FlexBE in particular, many of our specific implementations lacked the ability to easily re-try a particular sub-task as we do not allow jumping to arbitrary states within the state machine at runtime (unless behavior modifications are invoked; c.f. Section 4.1 and (Schillinger et al., 2016)). In the future, we plan to incorporate more strategic loopbacks into our state machines in the event of state failures.

Our main objective in employing high-level behavior control was to handle the DRC Finals tasks. However, thanks to FlexBE's graphical state machine composition capabilities, Team ViGIR ended up using high-level behaviors to also automate other tasks for Atlas. Examples include the startup check-out procedure (turning the pump on, calibrating the electric joints, etc.), the calibration of Atlas' hydraulic joints, as well as days worth of system identification experiments (Schappler et al., 2015). The resulting check-out and calibration behaviors were used in the DRC Finals, at the beginning of each run and after resets.

Lastly, we observed that encoding high-level tasks in the form of FlexBE state machines promotes proper software engineering practices. For example, having each state implementation perform a single action enforces modularity and the single responsibility principle. In addition, the very concept of hierarchical state machines (but also templates and affordances) lends itself to multiple layers of abstraction. Similarly, the concept of state outcomes reinforces the use of ROS actions and services, which are higher level interfaces compared to simply publishing and subscribing to ROS topics. Finally, FlexBE state machines handle both logic and data flow, but succeed in keeping the two separate.

7.3 Multi-team Collaboration

Team ViGIR's software was originally designed and implemented for the Atlas robot. Therefore, multiple changes were required to refactor different components to provide a robot agnostic architecture. Although all dependencies to Atlas specific libraries were removed, Team Hector encountered situations where software was not generic enough to cover their particular robot platform needs or features were not available. In these cases, Team Hector collaborated and contributed by adapting and improving the high-level software.

Through the additional usage by Team Hector and Team Valor, the robot agnostic software has been automatically tested thrice. Therefore, the software was quickly put to the test, and bugs were able to be identified and fixed in a short time. Developing robot-agnostic software requires that each developer considers how the software is intended to work and how is it supposed to be used by the other teams. Therefore, generic solutions have to be provided which can be adapted or configured for a particular robot system in an efficient way. Changes in software architecture have to be cleanly updated and clearly communicated to the collaborating teams, similar to what software companies standards are.

We also learned that well designed software can be used by different robot systems without loss of capabilities. For example, the OCS software is almost identical for all our robots, this way, operators can train using a particular robot platform and quickly adapt to control a different humanoid robot.

8 Conclusion

In this article, we presented an approach to operate remote robots to perform manipulation tasks using a collaborative autonomy principle where robot action-commands are shared between high-level behaviors and human operators. We approached this by allowing a human operator to interact with the remote robot in three different layers of control: Motion Planning, Object Template Manipulation, and High-level Behaviors. The motion planning backend provides joint control in different levels of commands such as joint control and Cartesian teleoperation which can then be abstracted by object template based manipulation. This provides foundation to the three main pillars of the contributions presented in this paper. First, object templates provide an interaction method for object manipulation that can be used by human operators and high-level behaviors. Second, high-level behaviors (FlexBE) provide an abstraction of the lower layers of control to allow interaction between human operators and robots at a task-level. And third, the concept of collaborative autonomy systematically brings these concepts together allowing for collaboration between high-level behaviors and human operators. We also open-sourced all of the corresponding software.

As shown by the DRC task performance in Section 6.1.4, the flexibility of changing control within these layers increases resilience in achieving remote manipulation in complex disaster response tasks. Operators are able to transit back and forth between these layers whenever the need arises, adapting the approach to solve a manipulation task on the fly. Performance of the approach was shown during the DRC Finals in two types of tasks, however, specific challenges that arose during competition prevented the approach from demonstrating the full potential of collaborative autonomy using high-level behaviors.

As shown by experiments in Sections 6.2.1 and 6.2.3, the operator has a higher level of control compared to pure Cartesian teleoperation, but there are still a significant amount of steps that the operator needs to execute manually. This requires training and practice and limits the number of people that are capable of using the system. High-level behavior experiments in Section 6.2.4 and 6.2.2 show that the approach can allow a human operator to perform remote manipulation at a task-level using high-level behaviors with minimal human input (mainly object template alignment and high-priority behavior transitions).

Furthermore, we show that the collaborative autonomy approach was successfully implemented in the two different humanoid robot platforms, from Team ViGIR and Team Hector. This opens the opportunity to extend this concept to non-humanoid robot platforms which are enabled to perform manipulation tasks.

Acknowledgments

This project was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Air Force Research Lab (AFRL) contract FA8750-12-C-0337 to TORC Robotics; Team ViGIR would like to thank TORC Robotics for their support and overall project management. Team Hector would like to thank TORC Robotics for opening their doors and providing logistical support during final testing and transportation. Team ViGIR and Team Hector would like to thank all team members; their contribution and support enabled the realization of this work. We would also like to thank DARPA and its support staff for a well run competition, Boston Dynamics, Inc. for their support with the Atlas robot, Robotis, Inc. for their support with THORMANG robot, and the Open Source Robotics Foundation (OSRF) for their support of ROS and Gazebo. The teams would also like to thank the contributors and maintainers of MoveIt! and the SMACH High-level Executive.

References

Babu, B. P. W., Du, R., Padir, T., and Gennert, M. A. (2015). Improving robustness in complex tasks for a supervisor operated humanoid. In *Humanoids*.

Berenson, D., Srinivasa, S. S., and Kuffner, J. (2011). Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*.

Birkenkampf, P., Leidner, D., and Borst, C. (2014). A knowledge-driven shared autonomy human-robot interface for tablet computers. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 152–159.

Bohren, J. and Cousins, S. (2010). The SMACH High-Level Executive [ROS News]. *Robotics Automation Magazine, IEEE*, 17(4):18–20.

Cheng, G. and Zelinsky, A. (2001). Supervised autonomy: A framework for human-robot systems development. *Autonomous Robots*, 10(3):251–266.

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

Crandall, J. and Goodrich, M. (2001). Experiments in adjustable autonomy. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 3, pages 1624–1629 vol.3.

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.

DRC Teams (2015). What happened at the DARPA Robotics Challenge? `www.cs.cmu.edu/~cga/drc/events`. Accessed: 2015-09-23.

Fallon, M. et al. (2015). An architecture for online affordance-based perception and whole-body planning. *Journal of Field Robotics*, 32(2):229–254.

Fallon, M. and Marion, P. (2016). Private communication.

Fong, T. W. and Nourbakhsh, I. (2004). Peer-to-Peer Human-Robot Interaction for Space Exploration. In *AAAI Fall Symposium*. AAAI.

Fong, T. W., Thorpe, C., and Baur, C. (1999). Collaborative Control: A Robot-Centric Model for Vehicle Teleoperation. In *AAAI 1999 Spring Symposium: Agents with Adjustable Autonomy*.

Gibson, J. J. (1977). The theory of affordances. In Shaw, R. and J. Bransford, E., editors, *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*, pages 67–82, Hilldale, USA.

Goodrich, M. A., Crandall, J. W., and Barakova, E. (2013). Teleoperation and Beyond for Assistive Humanoid Robots. *Reviews of Human Factors and Ergonomics*, 9(1):175–226.

Hart, S., Dinh, P., and Hambuchen, K. (2015). The Affordance Template ROS Package for Robot Task Programming. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on.*

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

Hebert, P. et al. (2015). Mobile manipulation and mobility as manipulation—design and algorithms of robosimian. *Journal of Field Robotics*, 32(2):255–274.

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*. Software available at `http://octomap.github.com`.

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.

Johnson, M., Bradshaw, J., Feltovich, P., Jonker, C., van Riemsdijk, B., and Sierhuis, M. (2011). The Fundamental Principle of Coactive Design: Interdependence Must Shape Autonomy. In De Vos, M., Fornara, N., Pitt, J., and Vouros, G., editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, volume 6541 of *Lecture Notes in Computer Science*, pages 172–191. Springer Berlin Heidelberg.

Johnson, M., Shrewsbury, B., Bertrand, S., Wu, T., Duran, D., Floyd, M., Abeles, P., Stephen, D., Mertins, N., Lesman, A., Carff, J., Rifenburgh, W., Kaveti, P., Straatman, W., Smith, J., Griffioen, M., Layton, B., de Boer, T., Koolen, T., Neuhaus, P., and Pratt, J. (2015). Team IHMC's Lessons Learned from the DARPA Robotics Challenge Trials. *Journal of Field Robotics*, 32(2):192–208.

Klien, G., Woods, D., Bradshaw, J., Hoffman, R., and Feltovich, P. (2004). Ten challenges for making automation a "team player" in joint human-agent activity. *Intelligent Systems, IEEE*, 19(6):91–95.

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.

Kohlbrecher, S., Romay, A., Stumpf, A., Gupta, A., von Stryk, O., Bacim, F., Bowman, D. A., Goins, A., Balasubramanian, R., and Conner, D. C. (2015). Human-robot teaming for rescue missions: Team ViGIR's approach to the 2013 DARPA Robotics Challenge Trials. *Journal of Field Robotics*, 32(3):352–377.

Koolen, T., Smith, J., Thomas, G., Bertrand, S., Carff, J., Mertins, N., Stephen, D., Abeles, P., Englsberger, J., Mccrory, S., et al. (2013). Summary of team IHMC's Virtual Robotics Challenge entry. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots.*

Leidner, D., Borst, C., and Hirzinger, G. (2012). Things are made for what they are: Solving manipulation tasks by using functional object classes. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 429–435.

Murphy, R. R. (2015). Meta-analysis of Autonomy at the DARPA Robotics Challenge Trials. *Journal of Field Robotics*, 32(2):189–191.

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.

Radford, N. A. et al. (2015). Valkyrie: NASA's First Bipedal Humanoid Robot. *Journal of Field Robotics*, 32(3):397–419.

Romay, A., Kohlbrecher, S., Conner, D. C., Stumpf, A., and von Stryk, O. (2014). Template-based Manipulation in Unstructured Environments for Supervised Semi-autonomous Humanoid Robots. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 979–986.

Romay, A., Kohlbrecher, S., Conner, D. C., and von Stryk, O. (2015). Achieving Versatile Manipulation Tasks with unknown Objects by Supervised Humanoid Robots based on Object Templates. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 249–255.

Şahin, E., Çakmak, M., Doğar, M. R., Uğur, E., and Üçoluk, G. (2007). To afford or not to afford: A new formalization of affordances toward affordance-based robot control. *Adaptive Behavior*, 15(4):447–472.

Schappler, M., Vorndamme, J., Tödtheide, A., Conner, D. C., von Stryk, O., and Haddadin, S. (2015). Modeling, Identification and Impedance Control of the Atlas Arms. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*.

Schillinger, P. (2015). An Approach for Runtime-Modifiable Behavior Control of Humanoid Rescue Robots. Master's thesis, Technische Universität Darmstadt.

Schillinger, P., Kohlbrecher, S., and von Stryk, O. (2016). Human-Robot Collaborative High-Level Control with an Application to Rescue Robotics. In *IEEE International Conference on Robotics and Automation (to appear)*, Stockholm, Sweden.

Tedrake, R. (2014). Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems. `http://drake.mit.edu`. Accessed: 2016-02-10.